Package 'stsm'

February 20, 2015

Version 1.7

Date 2015-01-26

Title Structural Time Series Models

Description Fit the basic structural time series model by maximum likelihood.

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

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

Depends R (>= 3.0.0), methods

Imports KFKSDS, parallel, stats

Suggests mvtnorm, numDeriv

NeedsCompilation yes

Encoding UTF-8

License GPL-2

URL http://jalobe.com

Repository CRAN

Date/Publication 2015-01-26 21:39:14

R topics documented:

`stsm-package` *Structural Time Series Models*

Description

This package provides algorithms to fit structural time series models by maximum likelihood.

Details

As witnessed in the special issue of the *Journal of Statistical Software* (Commandeur *et al.*, 2011), the prevalent procedure to fit a structural time series model is as follows: 1) choose arbitrary starting values for the parameters, 2) evaluate the log-likelihood function by means of the Kalman filter, 3) obtain a new set of parameter values that lead to a higher value of the log-likelihood function by means of the L-BFGS-B algorithm, 4) iterate the searching procedure until a predetermined degree of convergence.

Considering that there are several packages in R to run the Kalman filter (see for instance Tusell, 2011 and the documentation in package **KFKSDS**) and that the `optim` function in the **stats** package provides and interface to the L-BFGS-B and to other optimization algorithms, fitting a structural time series model may seem a simple procedure that requires little more than translating the matrices of the state space form of the model into the syntax of the chosen interface.

In practice, the process is not always that straightforward. As stated in the documentation of `StructTS`, *optimization of structural models is a lot harder than many of the references admit*. There are several details that should be taken into account when implementing the procedure described above, (López-de-Lacalle, 2013).

There are not many packages in R that provide alternative procedures to fit structural models. It is probably a consequence of the widespread believe that all that is needed to carry out and analysis with structural time series models is an implementation of the Kalman filter together with a general purpose optimization algorithm.

The package **stsm** implements specific algorithms to fit models in the framework of the basic structural time series model. The following enhancements to general purpose optimization algorithms are implemented: scoring algorithm based on analytical derivatives, maximization of the time or

frequency domain likelihood function, automatic choice of the optimal step size, concentration of a parameter, implementation of the original and a modified version of the expectation-maximization algorithm.

References

Commandeur, J.J.F., Koopman, S.J. and Ooms, M. (2011). 'Statistical Software for State Space Methods', *Journal of Statistical Software*, Vol. 41, No. 1, http://www.jstatsoft.org/v41/i01/.

Durbin, J. and Koopman, S.J. (2001). *Time Series Analysis by State Space Methods*. Oxford University Press.

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

López-de-Lacalle, J. (2013). '101 Variations on a Maximum Likelihood Procedure for a Structural Time Series Model.' Unpublished manuscript.

Tusell. F. (2011). 'Kalman Filtering in R.' *Journal of Statistical Software*, Vol. 39, No. 2. http://www.jstatsoft.org/v39/i02/.

Author(s)

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

`barrier.eval` *Barrier Term in the Likelihood Function*

Description

This function evaluates a barrier term to be added to the log-likelihood function in order to penalize for parameter values close to the boundaries.

Usage

```
barrier.eval(model, type = c("1", "2"), mu = 0.01,
  gradient = FALSE, hessian = FALSE)
```

Arguments

`model`	object of class `stsm`.
`type`	a character indicating the type of barrier term.
`mu`	the barrier multiplier. A positive value.
`gradient`	logical. If `TRUE`, the first order derivatives of the barrier term with respect to the parameters of the model are evaluated.
`hessian`	logical. If `TRUE`, the second order derivatives of the barrier term with respect to the parameters of the model are evaluated.

Details

A barrier term can be defined in order to penalize against parameter values close to the boundaries defined in `model@lower` and `model@upper`. The barrier term is added to the negative of the log-likelihood function, which is then given by:

$$-logLik + \mu \sum_i q_i(x).$$

μ is a positive barrier multiplier, the larger it is the stronger the barrier is and, hence, the search is done farther from the boundaries (Rardin, 1998 Fig. 14.8); $q_i(x)$ are the barrier functions that are defined in such a way that the barrier term grows to infinity as the parameters of the model approach to the boundaries. Two types of barrier functions are considered:

`type = "1":` $-log(par[i] - bound[i])$, for lower bound constraints.

`type = "1":` $-log(bound[i] - par[i])$, for upper bound constraints.

`type = "2":` $1/(par[i] - bound[i])$, for lower bound constraints.

`type = "2":` $1/(bound[i] - par[i])$, for upper bound constraints.

Value

A list containing the barrier term evaluated for the lower and upper bound constraints, `bl` and `bu`, respectively, and the first and second order derivatives of the barrier term, `dl1`, `du1`, `dl2`, `du2`.

References

Rardin, R.L. (1998). Section 14.5. *Optimization in Operations Research*. Prentice Hall.

See Also

`mloglik.fd`, `mloglik.td`.

Examples

```
# define local level plus seasonal model for a simulated time series
# and evaluate the barrier term for some parameter values
# the barrier term in the second model is higher since the variances
# are closer to the lower bounds (zero)
data("llmseas")
pars <- c("var1" = 30, "var2" = 1, "var3" = 10)
m1 <- stsm.model(model = "llm+seas", y = llmseas, pars = pars)
bar1 <- barrier.eval(m1, type = "2", mu = 3)
bar1$barrier

pars <- c("var1" = 0.3, "var2" = 0.01, "var3" = 0.1)
m2 <- stsm.model(model = "llm+seas", y = llmseas, pars = pars)
bar2 <- barrier.eval(m2, type = "2", mu = 3)
bar2$barrier

# the barrier term is added to the negative of the likelihood function
```

```
# that will be the objective function to be minimized,
# value of minus log-likelihood without barrier
mloglik.fd(model = m2)
# adding a barrier term
mloglik.fd(model = m2) + bar2$barrier
mloglik.fd(model = m2, barrier = list(type = "2", mu = 3))
```

`datagen.stsm` *Generate Data from a Structural Time Series Model*

Description

This function simulates data from a structural time series model defined in an object of class `stsm`.

Usage

```
datagen.stsm(n, model = list(), SigmaEV, labels, n0 = 20, freq = 1,
  old.version = FALSE)
```

Arguments

`n`	number of observations in the output time series.
`model`	a list containing the matrices of the state space for of the structural model.
`SigmaEV`	a list containing the elements `values` and `vectors`, they are respectively the eigen values and vectors of the covariance matrix of the disturbance terms in the state equation of the model.
`labels`	optional vector of characters giving the names of the unobserved components in the model.
`n0`	number of warming-up observations (they are not included in the output data).
`freq`	number of observations per unit of time, e.g. `freq = 4` for quarterly data.
`old.version`	logical. If `TRUE`, results obtained in a previous version of the package are reproduced.

Details

The matrices in the argument `model` must follow the conventions of an object of class `stsm` as returned by `char2numeric`.

For compatibility with previous versions of the package, `old.version = TRUE` generates random values from the multivariate normal distribution using the function `rmvnorm` with `pre0.9_9994 = TRUE`. `mvrnorm.version = FALSE` uses the theoretical expression that is commonly used to define random draws for the multivariate normal distribution.

Value

A list containing the output time series and the underlying components of the model.

See Also

sim-data, stsm.

Examples

```
# generate a quarterly series from a local level plus seasonal model
# the data set 'llmseas' is generated as follows (first series)
pars <- c(var1 = 300, var2 = 10, var3 = 100)
m <- stsm.model(model = "llm+seas", y = ts(seq(120), frequency = 4),
  pars = pars, nopars = NULL)
ss <- char2numeric(m)
set.seed(123)
y <- datagen.stsm(n = 120, model = list(Z = ss$Z, T = ss$T, H = ss$H, Q = ss$Q),
  n0 = 20, freq = 4, old.version = TRUE)$data

data("llmseas")
all.equal(y, llmseas)
```

---

force.defpos *Force Positive Definiteness of a Matrix*

Description

This function transforms a matrix to be positive definite.

Usage

```
force.defpos(m, tol = 0.001, debug = FALSE)
```

Arguments

m	a matrix.
tol	tolerance.
debug	logical. If TRUE, it is checked that the output matrix is positive definite.

Details

The scoring algorithm maxlik.fd.scoring requires a positive definite matrix to project the gradient into the optimal direction. If that matrix happens to be non-positive definite then the matrix M is transformed as described in Pollock (1999) pp. 341-342:

$$M = M + (\mu - \kappa)I$$

where I is the identity matrix, μ is a tolerance value and κ is the smallest eigenvalue of M.

Other alternatives are discussed in Nocedal and Wright (2006) chapter 3.

Value

A positive definite matrix.

References

Nocedal, J. and Wright, J. W. (2006). *Numerical Optimization*. Springer-Verlag.

Pollock, D.S.G. (1999). *A Handbook of Time-Series Analysis Signal Processing and Dynamics*. Academic Press.

---

gdp4795 *USA Real Gross Domestic Product*

Description

Quarterly U.S. real gross domestic product for the period 1947:I-1995:III.

Usage

```
gdp4795
```

Format

A time series object.

References

Clark, P.K. (1987). 'The Cyclical Component of U.S. Economic Activity', *Quarterly Journal of Economics*, **102**(4), pp. 797-814.

---

init.vars *Initial Parameter Values*

Description

This function computes initial variance parameters to be used as starting parameter values in an optimization procedure.

Usage

```
init.vars(model, debug = FALSE)
```

Arguments

model	an object of class stsm.
debug	logical. If TRUE, the correctness if the result is double-checked.

Details

As mentioned in Harvey (1989), the frequency domain representation of the structural model suggests using a linear regression to compute initial variance parameters from which to start an optimization procedure. The variable 2π times the periodogram is regressed on the constant terms of the spectral generating function of the model.

Value

A list containing the initial variance parameters and the output of the linear regression.

References

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

See Also

`maxlik.fd`, `stsm.sgf`.

`linesearch` *Choice of the Step Size in the Scoring Algorithm*

Description

These functions are used by the scoring optimization algorithms in order to choose the step size.

Usage

```
Brent.fmin(a = 0, b, fcn, tol = .Machine$double.eps^0.25, ...)
linesearch(b, fcn, grd, ftol = 0.0001, gtol = 0.9, ...)
step.maxsize(x, xlo, xup, pd, cap = 1)
```

Arguments

`a`	a numeric, lower end point of the interval where a line search procedure will search for the optimum step size. It should be zero or a positive value.
`b`	a numeric, upper end point of the searching interval.
`fcn`	a function to be optimized with respect to the step size.
`tol`	the tolerance for convergence of the line search procedure.
`grd`	a function returning the gradient of `fcn`.
`ftol`	a numeric, tolerance of the Wolfe condition related to the value of the function.
`gtol`	a numeric, tolerance of the Wolfe condition related to the gradient.
`x`	a numeric containing the current value of the parameters.
`xlo`	a numeric, lower bounds of the parameters of the model.

xup	a numeric, upper bounds of the parameters of the model.
pd	a numeric, direction vector chosen by the scoring algorithm.
cap	the maximum step size allowed, the default is 1.
...	arguments to be passed to the objective function or the gradient.

Details

These functions are intended to be called by other functions, not to be used directly by the user. For details about how fcn and grd should be defined see the source code of maxlik.fd.scoring and maxlik.td.scoring.

The default line search procedure used by the scoring algorithms is the univariate optimization function optimize from package stats. The functions linesearch and Brent.fmin are used for debugging, didactic and experimental purposes. They provide useful information when testing the scoring algorithm and allowed easy tune of some parameters of the line search procedure. This kind of information and options are not available for example in optim from package **stats**.

Brent.fmin is a version ported directly from the R sources (procedure Brent_fmin in file 'optimize.c'). linesearch is based on Nocedal and Wright (2006) chapter 3 and Pollock (1999) Chapter 12. It can be used to test the effect and role of the Wolfe conditions.

The function step.maxsize is not a line search procedure. Given the direction vector chosen by the scoring algorithm, this function returns the upper end of the interval where the line search procedure will search for the optimum step size. It ensures that for any step size inside the interval from 0 to the returned value the updated parameter values abide to the lower and upper bounds. This approach is also used by A. Clausen in his implementation of the BFGS algorithm. The use of this function is a simple alternative to reparameterizations of the model and to the idea implemented in the L-BFGS-B algorithm in order to deal with this kind of constraints.

Value

Brent.fmin and linesearch return a list containing:

vx	a vector containing the optimal value at each iteration during the bracketing.
minimum	the optimal value found in the last iteration.
fx	the value of the function for the optimal step size.
iter	number of iterations employed by the procedure.
counts	number of calls to the objective function. For linesearch it is a two-element vector where the second records the number of calls made to the gradient.

step.maxsize returns a numeric containing the highest possible that is compatible with the arguments passed to the function (direction vector and bounds).

References

Brent, R. (1973) *Algorithms for Minimization without Derivatives*. Prentice-Hall.

Clausen, A. R code for the BFGS algorithm. http://economics.sas.upenn.edu/~clausen/computing/optim.php.

Nocedal, J. and Wright, J. W. (2006). *Numerical Optimization*. Springer-Verlag.

Pollock, D.S.G. (1999). *A Handbook of Time-Series Analysis Signal Processing and Dynamics*. Academic Press.

See Also

maxlik.fd.scoring, maxlik.td.scoring.

---

maxlik.em *Maximization of the Time Domain Likelihood Function via the Expectation-Maximization Algorithm*

Description

Maximize the time domain log-likelihood function of a structural time series model by means of the Expectation-Maximization algorithm.

Usage

```
maxlik.em(model, type = c("standard", "modified", "mix"),
  tol = 0.001, maxiter = 300, kfconv = c(0, 10, 1),
  ur.maxiter = 1000, r.interval = c(0.001, var(model@y)*0.75, 20),
  mod.steps = seq(3, maxiter, 10),
  parallel = FALSE, num.cores = NULL)
```

Arguments

model	an object of class stsm.
type	a character choosing the implementation of the algorithm, the original (standard), a modified version or a mixture of both.
tol	the convergence tolerance.
maxiter	the maximum number of iterations.
kfconv	a length-three vector with control parameters used to determine whether the Kalman filter has converged at a given iteration.
ur.maxiter	the maximum number of iterations used in the root finding procedure. Ignored with type = "standard".
r.interval	a three-length vector to control the ends of the interval in the root finding procedure.
mod.steps	a vector indicating the iterations at which the modified EM algorithm is to be run. Only for type = "mix".
parallel	logical. If TRUE the process is run in parallel by means of function mclapply in package **parallel**.
num.cores	an optional numeric. The number of processes executed in parallel if parallel is TRUE. By default it is set equal to the number of CPU cores.

Details

This function is based on the discussion given in López-de-Lacalle (2013) about the Expectation-Maximization (EM) algorithm in the context of structural time series models The previous reference includes an R package called **stsm.em** but since it uses some non-standard options it has not been distributed. A version of the most relevant procedures is provided in the function `maxlik.em`.

The traditional design of the EM algorithm in the context of structural time series models is run with `type = "standard"`. This approach is described for instance in Durbin and Koopman (2001) Section 7.3.4. A modified version introduced in López-de-Lacalle (2013) can be run using `type = "modified"`. A mixture of both approaches is also possible by setting `type = "mix"`. In that case, the modified version is run at those iterations indicated in `mod.steps` and the traditional version is run in the remaining iterations. As we do not know beforehand the number of iterations required for convergence, `mod.steps` should be defined considering up to `maxiter` possible iterations.

In pure variance models, the Kalman filter may converge to a steady state. The parameters in `kfconv` control how convergence of the filter is determined. It is considered that convergence is reached when the following is observed: the change in the variance of the prediction error over the last `kfconv[2]` consecutive iterations of the filter is below the tolerance `kfconv[1]`. The iteration at which the the Kalman smoother has converged is the iteration where the Kalman filter converged multiplied by the factor `kfconv[3]`. If provided, `kfconv[3]` should be equal or greater than unity.

The argument `r.interval` is a three-length vector. The first two elements are the initial lower and upper ends of the interval where the variance parameters are searched. The third element is the first iteration of the EM algorithm after which the initial searching interval is narrowed. A relatively width initial interval is recommended. As the algorithm makes progress, the interval is automatically narrowed according to the values and path followed in the first `r.interval[3]` and subsequent iterations.

Value

A list containing the elements:

`Mpars`	a matrix with the values of the parameters stored by row for each iteration of the procedure.
`pars`	parameter values at the local optimum or point where the algorithm stopped.
`iter`	number of iterations until convergence or stopping.

For `type = "modified"` or `type = "mix"` the element `calls.v1` is also returned. It reports, for each parameter, the number of calls that were done to the traditional version due to failure to convergence of the root finding procedure.

References

Durbin, J. and Koopman, S.J. (2001). Section 7.3.4. *Time Series Analysis by State Space Methods*. Oxford University Press.

Harvey, A. C. (1989). Section 4.2.4. *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Koopman, S.J. and Shephard, N. (1992) Exact Score for Time Series Models in State Space Form. *Biometrika* **79**(4), pp. 823-826.

Koopman, S.J. (1993). Disturbance Smoother for State Space Models. *Biometrika* **80**(1), pp. 117-126.

López-de-Lacalle, J. (2013). 'Why Does the Expectation-Maximization Algorithm Converge Slowly in Pure Variance Structural Time Series Models?' Unpublished manuscript.

Shumway, R.H. and Stoffer, D.S. (1982) An Approach to Time Series Smoothing and Forecasting Using the EM Algorithm. *Journal of Time Series Analysis* **3**(4), pp. 253-264.

See Also

maxlik.td.

Examples

```
# fit a local level plus seasonal model to a simulated sample series
# using the three versions of the EM algorithm
# the same solution is found by all versions (up to a tolerance)
# the modified version converges in fewer iterations, yet it involves
# more computations
data("llmseas")
m <- stsm.model(model = "llm+seas", y = llmseas,
  pars = c(var1 = 1, var2 = 1, var3 = 1))

# original version
res1 <- maxlik.em(m, type = "standard",
  tol = 0.001, maxiter = 350, kfconv = c(0.01, 10, 1))
res1$pars
res1$iter

# modified version
res2 <- maxlik.em(m, type = "modified",
  tol = 0.001, maxiter = 250, kfconv = c(0.01, 10, 1),
  ur.maxiter = 1000, r.interval = c(0.001, var(m@y)*0.75, 20))
res2$pars
res2$iter
res2$calls.v1

# mixture, the modified version is run every 10 iterations
# starting in the third one
res3 <- maxlik.em(m, type = "mix",
  tol = 0.001, maxiter = 250, kfconv = c(0.01, 10, 1),
  ur.maxiter = 1000, r.interval = c(0.001, var(m@y)*0.75, 20),
  mod.steps = seq(3, 200, 10))
res3$pars
res3$iter
res3$calls.v1
```

maxlik.fd *Maximization of the Spectral Likelihood Function*

Description

Maximize the spectral log-likelihood function of a structural time series model by means of a scoring algorithm or a general purpose optimization algorithm available in optim.

Usage

```
maxlik.fd.optim(m,
  barrier = list(type = c("1", "2"), mu = 0), inf = 99999,
  method = c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG", "SANN"),
  gr = c("analytical", "numerical"), optim.control = list())

maxlik.fd.scoring(m, step = NULL,
  information = c("expected", "observed", "mix"),
  ls = list(type = "optimize", tol = .Machine$double.eps^0.25, cap = 1),
  barrier = list(type = c("1", "2"), mu = 0),
  control = list(maxit = 100, tol = 0.001, trace = FALSE, silent = FALSE),
  debug = FALSE)

maxclik.fd.scoring(m, step = NULL, information = c("expected", "observed"),
  ls = list(type = "optimize", tol = .Machine$double.eps^0.25, cap = 1),
  barrier = list(type = c("1", "2"), mu = 0),
  control = list(maxit = 100, tol = 0.001, trace = FALSE, silent = FALSE))
```

Arguments

m	an object of class stsm.
barrier	a list defining a barrier term to penalize parameter values close to the bounds m@lower and m@upper.
inf	a numeric indicating the value to be returned if the value of the log-likelihood function happens to be NA or non-finite at some iteration of the optimization procedure.
method	character indicating the method to be used by optim.
gr	character indicating whether numerical or analytical derivatives should be used.
optim.control	a list of control parameters passed to optim.
step	if it is a numeric it stands for a fixed step size, otherwise an automatic procedure is used to choose the step size.
information	the type of information about second order derivatives used to project the gradient in the scoring algorithm.
ls	control parameters for the line search procedure used to chose the step size in the scoring algorithm.

control	a list of control parameters for the scoring algorithm.
debug	logical. If TRUE, tracing information is printed for debugging purposes of the scoring algorithm.

Details

The matrix used to project the gradient may be based on expected or observed information. The former, information = "expected", uses the analytical expression of the information matrix. The latter, information = "observed", uses the analytical expression of the Hessian as in a Newton-Raphson procedure. The option information = "mix" uses a mixture of both expressions; in simulations it performed similar to the information matrix, this option may be removed in future versions of the package.

maxclik.fd.scoring maximizes the concentrated likelihood function. The parameter to be concentrated must be defined in the slot cpar of the input model m, see stsm. maxlik.fd.optim detects whether cpar is defined in the input model. In the scoring algorithm, if m@cpar is not NULL maxclik.fd.scoring should be used.

Bounds on parameters and barrier term. The lower and upper bounds within which the L-BFGS-B algorithm conducts the search are taken from the slots lower and upper defined in the input object m. As an alternative to the L-BFGS-B procedure, a barrier term can be passed as argument.

The barrier term is added to the likelihood function and acts as a penalization for parameter values close to the bounds. For details about the barrier term see barrier.eval. In the scoring algorithm, if step = NULL the procedure automatically searches the optimum step size that is compatible with the bounds on the parameters. See step.maxsize and further details below.

Control parameters for the scoring algorithm. maxit, maximum number of iterations (the default is 100); tol, tolerance to assess convergence of the algorithm (the default is 0.001); trace, logical, if TRUE, the values of the parameters at each iteration of the procedure are stored and returned in a matrix; silent, logical, if FALSE, a warning is printed if convergence is not achieved.

Choice of the step size in the scoring algorithm. If step is a numerical value, the step size is fixed to that value at all iterations of the algorithm. Otherwise, the choice of the step size in the scoring algorithm is made by means of a line search procedure specified in the argument ls. See step.maxsize for a description of the elements that can be passed through the argument ls.

External regressors If external regressors are included in the model m, starting values for their coefficients are obtained in a linear regression of the differenced series on the differenced regressors. The values in the slot pars are therefore overwritten and not used as initial values.

Value

A list of class stsmFit with components:

call	an object of class call specifying the arguments passed to the function.
init	initial parameter values.
pars	parameter values at the local optimum.
m	the stsm model object updated with the optimal parameter values.
loglik	the value of the log-likelihood function at the local optimum.
convergence	convergence code returned by optim; for the scoring algorithm a logical indicating whether convergence was achieved.

iter	for maxlik.fd.optim it is a two-element vector with the number of calls made by optim to the objective function and to the gradient; for maxlik.fd.scoring it is the number of iterations employed by the scoring algorithm.
message	an empty character or a character message giving some additional information about the optimization process.
Mpars	a matrix or NULL. If control$trace = TRUE in the scoring algorithm, the path to the local optimum is traced storing by rows the parameter values at each iteration.
steps	a vector or NULL. If control$trace = TRUE in the scoring algorithm, the step size used at each iteration is stored in this vector.
ls.iter	a vector containing the number of iterations employed by the line search procedure at each step of the scoring algorithm. It is NULL if ls$type = "optimize".
ls.counts	a two-element vector containing the total number of calls to the objective function and the gradient made by the line search procedure in all the iterations. (It is NULL if ls$type = "optimize".)

Note: if m@transPars is not NULL, the elements init and pars are in terms of the auxiliar set of parameters. If the output is stored for example in an object called res, get.pars(res$model) will return the actual variance parameters.

The version based on optim, maxlik.fd.optim, returns also the element hessian containing the numerically differentiated Hessian matrix at the local optimum. Note that, if the model is parameterized in terms of an auxiliar set of parameters and gradient = "numerical" is used, the Hessian returned by optim is defined with respect to the auxiliar set of parameters, not the variances.

References

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

Nocedal, J. and Wright, J. W. (2006). *Numerical Optimization*. Springer-Verlag.

See Also

barrier.eval, mloglik.fd, stsm, optim.

Examples

```
# fit the local level plus seasonal model to a
# sample simulated series
# further examples and options can be explored in the
# script files 'sim-llmseas-ml-fd.R' and 'sim-llmseas-mcl-fd.R'
# available in the 'inst' folder of the source package
data("llmseas")

# initial parameters and 'stsm' model
initpars <- c(var1 = 1, var2 = 1, var3 = 1)
m <- stsm.model(model = "llm+seas", y = llmseas, pars = initpars)

# Newton-Raphson algorithm (analytical Hessian)
```

```
res1 <- maxlik.fd.scoring(m = m, step = NULL,
  information = "observed", control = list(maxit = 100, tol = 0.001))
res1

# Scoring algorithm (information matrix)
res2 <- maxlik.fd.scoring(m = m, step = NULL,
  information = "expected", control = list(maxit = 100, tol = 0.001))
res2

# wrapper function for 'optim()' in the 'stats' package
res3 <- maxlik.fd.optim(m, method = "L-BFGS-B", gr = "analytical")
res3

# concentrating one of the parameters
# the model must be first defined accordingly, here 'var1', i.e.,
# the variance of the disturbance in the observation equation
# is concentrated, its standard error is reported as 'NA'
mc <- stsm.model(model = "llm+seas", y = llmseas,
  pars = initpars[-1], cpar = initpars[1])
res4 <- maxclik.fd.scoring(m = mc, step = NULL,
    information = "observed", control = list(maxit = 100, tol = 0.001))
res4
```

`maxlik.td` *Maximization of the Time Domain Likelihood Function*

Description

Maximize the time domain log-likelihood function of a structural time series model by means of a scoring algorithm or a general purpose optimization algorithm available in `optim`.

Usage

```
maxlik.td.optim(m,
  KF.version = eval(formals(KFKSDS::KalmanFilter)$KF.version),
  KF.args = list(), check.KF.args = TRUE,
  barrier = list(type = c("1", "2"), mu = 0), inf = 99999,
  method = c("BFGS", "L-BFGS-B", "Nelder-Mead", "CG", "SANN", "AB-NM"),
  gr = c("numerical", "analytical"), optim.control = list())

maxlik.td.scoring(m, step = NULL,
  KF.args = list(), check.KF.args = TRUE,
  ls = list(type = "optimize", tol = .Machine$double.eps^0.25, cap = 1),
  control = list(maxit = 100, tol = 0.001, trace = FALSE, silent = FALSE),
  debug = FALSE)
```

Arguments

m	an object of class stsm.
KF.version	character indicating the implementation of the Kalman filter to be used.
KF.args	a list of parameters to be passed to the function chosen to run the Kalman filter.
check.KF.args	logical. If TRUE, the arguments passed through KF.args are checked for consistency with the interface chosen in KF.version: for maxlik.td.scoring in it is checked for consistency with interface KFKSDS.
barrier	a list defining a barrier term to penalize parameter values close to the bounds m@lower and m@upper.
inf	a numeric indicating the value to be returned if the value of the log-likelihood function happens to be NA or non-finite at some iteration of the optimization procedure.
method	character indicating the method to be used by optim.
gr	character indicating whether numerical or analytical derivatives should be used.
optim.control	a list of control parameters passed to optim.
step	if it is a numeric it stands for a fixed step size, otherwise an automatic procedure is used to choose the step size.
ls	control parameters for the line search procedure used to chose the step size in the scoring algorithm.
control	a list of control parameters for the scoring algorithm.
debug	logical. If TRUE, tracing information is printed for debugging purposes of the scoring algorithm.

Details

The function maxlik.td.optim implements the common procedure of maximum likelihood, i.e., maximization of the time domain likelihood function by means of a numerical optimization algorithm (L-BFGS-B or other algorithms available in optim). The likelihood function is evaluated by means of the Kalman filter.

The function maxlik.td.scoring implements a scoring algorithm based on the analytical expression of the information matrix of the time domain likelihood function.

López-de-Lacalle (2013) discusses several options and details that are often omitted or ignored when maximizing the likelihood function of a structural time series models. The interface maxlik.td.optim allows the user to choose some options that may affect the results or convergence of the algorithm.

A novelty compared to other implementations such as StructTS is that the optimization procedure is enhanced by means of analytical derivatives. Another enhancement is that one of the parameters can be concentrated out of the likelihood function. The parameter to be concentrated is defined in the slot cpar of the input model m, see stsm. This option has not yet been implemented in the scoring algorithm.

For details about the options than can be passed through argument KF.args see the documentation of the same argument in function KalmanFilter in package **KFKSDS**.

For further information about the scoring algorithm see the following points in the details section of maxlik.fd.scoring: *Bounds on parameters and barrier term*, *Control parameters for the scoring algorithm* and *Choice of the step size in the scoring algorithm*.

If external regressors are included in the model `m`, starting values for their coefficients are obtained in a linear regression of the differenced series on the differenced regressors. The values in the slot `pars` are therefore overwritten and not used as initial values.

Note: `ls$type = "wolfe"` is not implemented for `maxlik.td.scoring`.

Value

A list of class `stsmFit`. See the section 'Value' in `maxlik.fd`.

References

Durbin, J. and Koopman, S. J. (2001). *Time Series Analysis by State Space Methods*. Oxford University Press.

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

López-de-Lacalle, J. (2013). '101 Variations on a Maximum Likelihood Procedure for a Structural Time Series Model.' Unpublished manuscript.

Nocedal, J. and Wright, J. W. (2006). *Numerical Optimization*. Springer-Verlag.

See Also

`barrier.eval`, `mloglik.td`, `stsm`, `optim`.

Examples

```
# replicate maximum likelihood procedure as implemented in 'stats::StructTS'
res0 <- stats::StructTS(log(AirPassengers), type = "BSM")
mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
  transPars = "StructTS")
res1 <- maxlik.td.optim(mairp, KF.version = "KFKSDS",
  KF.args = list(P0cov = TRUE), method = "L-BFGS-B", gr = "numerical")
mairp1 <- set.pars(mairp, pmax(res1$par, .Machine$double.eps))
round(get.pars(mairp1), 6)
all.equal(get.pars(mairp1), res0$coef[c(4,1:3)],
  tol = 1e-04, check.attributes = FALSE)

# smoothed components
ss1 <- char2numeric(mairp1, P0cov = TRUE)
kf1 <- KFKSDS::KF(mairp1@y, ss1)
ks1 <- KFKSDS::KS(mairp1@y, ss1, kf1)
plot(tsSmooth(res0)[,c(1,3)])
plot(ks1$ahat[,c(1,3)])

# the scoring algorithm reaches another local optimum where
# the seasonal component is more homoscedastic
mairp <- stsm.model(model = "BSM", y = log(AirPassengers),
  transPars = NULL)
res2 <- maxlik.td.scoring(mairp, step = NULL,
  ls = list(type = "optimize", tol = .Machine$double.eps^0.25, cap = 1),
  control = list(maxit = 100, tol = 0.001, trace = FALSE), debug = FALSE)
```

```
round(res2$par, 6)

mairp2 <- set.pars(mairp, res2$par)
ss2 <- char2numeric(mairp2, P0cov = FALSE)
kf2 <- KFKSDS::KF(mairp2@y, ss2)
ks2 <- KFKSDS::KS(mairp2@y, ss2, kf2)
plot(ks2$ahat[,c(1,3)])
```

`method-logLik` *Extract Log-Likelihood*

Description

This method returns the log-likelihood value of a model defined in a `stsm` object.

Usage

```
## S3 method for class 'stsm'
logLik(object, domain = c("frequency", "time"), xreg = NULL,
  td.args = list(P0cov = FALSE, t0 = 1,
    KF.version = eval(formals(KFKSDS::KalmanFilter)$KF.version)),
  check.td.args = TRUE,
  barrier = list(type = c("1", "2"), mu = 0),
  inf = 99999, ...)
```

Arguments

`object`	an object of class `stsm`.
`domain`	a character indicating whether the time domain or the spectral likelihood should be evaluated.
`xreg`	optional matrix of external regressors with the same number of rows as the length of the input time series `m@y`.
`td.args`	a list containing arguments to be passed to `mloglik.td`. Only for `domain = "time"`.
`check.td.args`	logical. If `TRUE`, argument `td.args` is checked for consistency with `td.args$KF.version`. Only for `domain = "time"`.
`barrier`	a list defining a barrier term to penalize parameter values close to the bounds `m@lower` and `m@upper`.
`inf`	a numeric indicating the value to be returned if the value of the log-likelihood function happens to be `NA` or non-finite.
`...`	additional arguments. Currently ignored.

Value

An object of class `logLik` containing the value of the log-likelihood function for the given model and the attribute `df`, the number of estimated parameters.

See Also

mloglik.fd, mloglik.td, mle, KalmanFilter, stsm.

---

methods-stsmFit *Methods to Extract Information from a Fitted* stsm *Model Object*

Description

Common methods to print and display results for an object of class stsm or stsmFit returned by functions maxlik.fd and maxlik.fd.

Usage

```
## S3 method for class 'stsmFit'
coef(object, ...)
## S3 method for class 'stsmFit'
print(x, digits = max(3L, getOption("digits") - 3L),
  vcov.type = c("hessian", "infomat", "OPG", "sandwich", "optimHessian"), ...)
## S3 method for class 'stsmFit'
fitted(object, std.rediduals = TRUE, version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmFit'
residuals(object, standardised = FALSE, version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmComponents'
plot(x, ...)
## S3 method for class 'stsm'
predict(object, n.ahead = 1L, se.fit = TRUE,
  version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmFit'
predict(object, n.ahead = 1L, se.fit = TRUE,
  version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmPredict'
plot(x, ...)
## S3 method for class 'stsm'
tsSmooth(object, version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmFit'
tsSmooth(object, version = c("KFKSDS", "stats"), ...)
## S3 method for class 'stsmSmooth'
plot(x, ...)
## S3 method for class 'stsmFit'
tsdiag(object, gof.lag = 10L, ...)
```

Arguments

object	an object of class stsm or a stsmFit list.
x	a stsmFit list. For plot methods it is a list of class stsmComponents, stsmPredict or stsmSmooth.

digits	minimal number of significant digits, see print.default.
vcov.type	a character indicating the type of covariance matrix to be used to compute the standard errors of the parameter estimates.
version	a character indicating whether the Kalman filter and smoother functions from package **KFKSDS** or from the **stats** package should be used.
std.rediduals	logical. If TRUE residuals are standardised.
standardised	logical. If TRUE standardised are returned.
n.ahead	a numeric, number of observations ahead to perform prediction.
se.fit	logical. If TRUE standard errors of the predictions are returned.
gof.lag	numeric, number of lag autocorrelation coefficients to which apply the Box test.
type	A character. Type of information used to compute the covariance matrix of the parameters of the fitted model: information matrix, Hessian or a mixture of them.
...	additional arguments to be passed to the functions called in these methods.

Details

These methods are based on those with the same name available for the output returned by StructTS in the **stats** package.

These methods are originally intended to provide summary information from a model fitted by maximum likelihood. Thus, the most natural input for them is a stsmFit list returned by maxlik.fd and maxlik.td. Nevertheless, as the information and the data required by these methods are available in the slots of a stsm object, they can also be applied directly on an object of class stsm. This can be useful, for example, when we know a set of parameter estimates that was obtained from a method other than maxlik.fd or maxlik.td. By simply updating the slot pars of the stsm object, the residuals and the filtered and smoothed components are readily available through these methods.

In most cases, the ellipsis, ..., is kept in the definitions of the methods just because it is part of the parent method. It has some functionality nonetheless. For the methods fitted.stsm and predict.stsm it can be used to pass argument P0cov to function char2numeric. It can also be used to pass graphical parameters to par in method plot.stsmComponents and plot.stsmSmooth or to plot in plot.stsmPredict and plot.stsmSmooth.

By default in method fitted, std.rediduals = TRUE so that it behaves as in previous versions of the package where this argument was not available. In method rediduals, by default standardised = FALSE because it is more convenient when used in package **tsoutliers** (this argument does not need to be explicitly defined and hence the method residuals is used in the same way both on arima models and for stsm).

Value

The following information is returned by these methods:

coef.stsmFit, print.stsmFit
: optimal parameter values.

fitted.stsm, fitted.stsmFit
: an list of class stsmComponents containing the filtered components and residuals.

`residuals.stsmFit`
: residuals in the fitted model.

`plot.stsmComponents`
: plot of the filtered components.

`predict.stsm, predict.stsmFit`
: predictions of the input time series and standard errors.

`plot.stsmPredict`
: plot of the predictions.

`tsSmooth.stsm, tsSmooth.stsmFit`
: smoothed components.

`plot.stsmSmooth`
: plot of the smoothed components.

`tsdiag.stsm, tsdiag.stsmFit`
: plot of diagnostic tests.

See Also

`confint.stsmFit`, `maxlik.fd`, `maxlik.td`, `vcov.stsmFit`, `stsm`.

Examples

```
# fit the local level plus seasonal model to a
# sample simulated series
data("llmseas")
m <- stsm.model(model = "llm+seas", y = llmseas)
res <- maxlik.fd.scoring(m = m, step = NULL,
  information = "expected", control = list(maxit = 100, tol = 0.001))
print(res)

#diagnostic
tsdiag(res)

# display estimated components with 95% confidence bands
comps <- tsSmooth(res)
plot(comps)
title(main = "smoothed trend and seasonal components")

# plot predictions eight periods ahead
pred <- predict(res, n.ahead = 8, se.fit = TRUE)
plot(pred)
```

---

methods-vcov-confint *Variance-covariance Matrix for a Fitted* `stsm` *Model Object*

Description

The method `vcov` computes the variance-covariance matrix of the parameters fitted in a structural time series model. This matrix is used to compute confidence intervals for those parameters returned by the `coef.stsmFit` method.

Usage

```
## S3 method for class 'stsmFit'
vcov(object,
  type = c("hessian", "infomat", "OPG", "sandwich", "optimHessian"), ...)
## S3 method for class 'stsm'
vcov(object,
  type = c("hessian", "infomat", "OPG", "sandwich"),
  domain = c("frequency", "time"), ...)
## S3 method for class 'stsmFit'
confint(object, parm, level = 0.95,
  type = c("vcov", "bootstrap"),
  vcov.type = c("hessian", "infomat", "OPG", "sandwich", "optimHessian"),
  breps = 100, ...)
```

Arguments

`object`	a `stsmFit` list or object of class `stsm`.
`type`	a character. In `vcov`, it is the type of covariance matrix. In `confit`, it is the type of confidence intervals: based on the covariance matrix of the estimated parameters or on a bootstrap procedure.
`domain`	a character indicating whether the covariance matrix is obtained upon the frequency or time domain likelihood function.
`parm`	character indicating the name of the parameter on to obtain the confidence interval. If missing, all parameters are considered.
`level`	the confidence level.
`vcov.type`	a character indicating the type of covariance matrix. Ignored if `type = "bootstrap"`.
`breps`	number of bootstrap replicates. Ignored if `type "vcov"`.
`...`	additional arguments to be passed to the functions called in these methods. Currently ignored.

Details

The following estimators of the covariance matrix of parameter estimates are available (Davidson and MacKinnon (2004), Section 10.4):

- `hessian`: the inverse of the analytical Hessian.
- `infomat`: the inverse of the analytical expression for the information matrix.
- `OPG`: the inverse of the outer product of the analytical gradient. Also known as the BHHH estimator since it was proposed by Berndt, Hall, Hall and Hausman (1974). This method requires only first order derivatives. It tends to be less reliable in small samples.
- `sandwich`: the sandwich estimator defined as: $H^{-1}(G'G)H^{-1}$, where G is the gradient vector and H is the Hessian matrix. It requires more computations and may be unreliable in small samples. However, contrary to the previous methods, it is valid when the information matrix equality does not hold for example due to misspecification of the model.
- `optimHessian`: the inverse of the numerical Hessian returned by `optim`.

The natural input for the method method vcov is a stsmFit list returned by maxlik.fd or maxlik.td. However, vcov can be also applied directly on a stsm model object. This is useful for example when maximum likelihood parameter estimates are found by means of optim using the functions maxlik.fd.optim or maxlik.fd.optim. In that case, only the covariance matrix based on the numerical Hessian returned by optim would be available. Updating the slot pars of a stsm model object with the parameter values obtained from other algorithm is a convenient solution to obtain the covariance matrix based on other methods available in vcov.stsmFit.

For the time domain likelihood function the covariance matrix of the initial state vector is considered diagonal, P0cov = FALSE.

The analytical Hessian for the time domain version is not available, the information matrix is used instead.

By default, vcov.type = "infomat" for the time domain likelihood function and vcov.type = "hessian" for the frequency domain likelihood function.

Confidence intervals can either be computed upon the covariance matrix of the parameter estimates (type = "vcov") or by means of bootstrapping (type = "bootstrap"). The bootstrap approach takes advantage of the following result (Harvey (1989) eq. (4.3.25)):

$$4\pi I(\lambda_j)/g(\lambda_j) \sim \chi^2_2, \text{ for } j \neq 0, n/2 \text{ (for n even)}$$

$$2\pi I(\lambda_j)/g(\lambda_j) \sim \chi^2_1, \text{ for } j = 0, n/2 \text{ (for n even)}$$

where $I(\lambda_j)$ and $g(\lambda_j)$ are respectively the periodogram and the spectral generating function at frequency λ_j. Upon this result, bootstrap replicates of the periodogram are generated and for each of them parameter estimates are obtained maximizing the spectral likelihood function. The quantiles of the bootstrapped parameter estimates are the confidence interval. Dahlhaus and Janas (1996) studied the properties of the frequency domain bootstrap which has been applied, among others, in Koopman and Wong (2006). An advantage of the bootstrap method is that it yields confidence intervals within the bounds of the parameters, i.e., positive variances. This procedure is computationally intensive and requires some time to run, especially for large breps.

Value

vcov.stsm, vcov.stsmFit
: return the covariance matrix of the parameters of the model.

confint.stsmFit
: returns a matrix containing confidence intervals for the parameters of the model.

References

Berndt, E. R., Hall, B. H., Hall, R. E. and Hausman, J. A. (1974). 'Estimation and inference in nonlinear structural models'. *Annals of Economic and Social Measurement*, **3**, pp. 653-65.

Dahlhaus, R. and Janas, D. (1996). 'A Frequency Domain Bootstrap for Ratio Statistics in Time Series Analysis'. *Annals of Statistics*, **24**(5), pp. 1934-1963.

Davidson, R. and MacKinnon, J. G. (2004). Section 10.4. *Econometric Theory and Methods*. Oxford University Press.

Koopman, S. J. and Wong, S. Y. (2006). 'Extracting Business Cycles using Semi-Parametric Time-varying Spectra with Applications to US Macroeconomic Time Series'. Tinbergen Institute Discussion Papers, No. 2006-105/4. http://papers.tinbergen.nl/06105.pdf

See Also

maxlik.fd, maxlik.td, methods-stsmFit, stsm.

Examples

```
## Not run:
data("llmseas")
# fit the local level plus seasonal model with default arguments
# using the Newton-Raphson algorithm
m <- stsm.model(model = "llm+seas", y = llmseas)
res <- maxlik.fd.scoring(m = m, information = "observed")
coef(res)
# confidence intervals for parameter estimates ...
# ... based on the covariance matrix of parameter estimates
# gives a warning since the lower limit of the confidence interval
# for parameter 'var2' was forced to be non-negative (fixed to 0)
civcov <- confint(res, type = "vcov", vcov.type = "hessian")
civcov
# ... based on bootstrapping the periodogram
# NOTE: this will take a while to run
set.seed(643)
ciboot <- confint(res, type = "bootstrap", breps = 100)
ciboot

## End(Not run)
```

---

mloglik.fd *Spectral Log-Likelihood Function and Derivatives*

Description

These functions evaluate the negative of the spectral log-likelihood function of a linear Gaussian state space model and its first and second order derivatives.

Usage

```
mloglik.fd(x, model,
  barrier = list(type = c("1", "2"), mu = 0), inf = 99999, xreg)

mcloglik.fd(x, model, xreg = NULL,
  barrier = list(type = c("1", "2"), mu = 0), inf = 99999)

mloglik.fd.deriv(model, xreg = NULL,
  gradient = TRUE, hessian = TRUE, infomat = TRUE, modcovgrad = TRUE,
  barrier = list(type = c("1", "2"), mu = 0),
  version = c("2", "1"))

mcloglik.fd.deriv(model, xreg = NULL,
```

```
  gradient = TRUE, hessian = TRUE, infomat = TRUE)

mloglik.fd.grad(x, model, xreg = NULL,
  barrier = list(type = c("1", "2"), mu = 0),
  inf)

mcloglik.fd.grad(x, model, xreg = NULL, inf, barrier)
```

Arguments

x	a numeric vector containing the parameters of the model. This is an auxiliar argument so that this function can be used as input to optim.
model	an object of class stsm.
xreg	optional list containing constant terms. See details.
barrier	a list defining a barrier term to penalize parameter values close to the bounds m@lower and m@upper.
inf	a numeric indicating the value to be returned if the value of the log-likelihood function happens to be NA or non-finite.
gradient	logical. If TRUE, first order derivatives of the negative of the spectral log-likelihood function are returned.
hessian	logical. If TRUE, second order derivatives of the negative of the spectral log-likelihood function are returned.
infomat	logical. If TRUE, the information matrix of the spectral log-likelihood are returned.
modcovgrad	logical. If TRUE, a mixture of the analytical expressions for the Hessian and the outer product of the gradient is used. This option is experimental and may be removed in future versions of the package.
version	a character indicating whether implementation "2" or "2" (the default) should be used.They yield the same result but are kept for debugging and comparison of timings. This argument may be removed in future versions.

Details

The spectral log-likelihood of a linear Gaussian state space model is given by (Harvery, 1989 Section 4.3):

$$logLik = -0.5log(2\pi) - 0.5\sum_{j=0}^{n-1} log\, g(\lambda[j]) - \pi\sum_{j=0}^{n-1} I(\lambda[j])/g(\lambda[j])$$

where $\lambda[j]$ is a frequency defined as $\lambda[j] = 2\pi j/n$; $I(\lambda[j])$ is the periodogram at frequency $\lambda[j]$ and $g(\lambda[j])$ is the spectral generating function of the model at frequency $\lambda[j]$.

The derivation of the spectral likelihood function defined above relies on the assumption that the process is circular (its covariance matrix is circulant). If the process is not circular the value of the likelihood is an approximation.

First and second order derivatives are computed by means of their analytical expressions. The first order derivatives of the spectral log-likelihood with respect to parameter θ are given by:

$$(d\,logLik)/(d\,\theta) = 0.5\sum_{j=0}^{n-1}((2\pi I(\lambda[j]))/(g(\lambda[j])) - 1)(1/g(\lambda[j]))dg(\lambda[j])/d\theta$$

Second order derivatives are given by:

$$(d^2\,logLik)(d\,\theta\theta') = \sum_{j=0}^{n-1}((2\pi I(\lambda[j]))/(g(\lambda[j])) - 1)1/(2g(\lambda[j]))(d^2 g(\lambda[j]))/(d\theta d\theta') -$$

$$2\sum_{j=0}^{n-1}((4\pi I(\lambda[j]))(g(\lambda[j])) - 1)(\frac{1}{2g(\lambda[j])})^2(dg(\lambda[j]))/(d\theta)(dg(\lambda[j]))/(d\theta')$$

The argument `x` is an auxiliar vector that is necessary in some contexts. For example, the input to function `optim` must contain as first argument the vector of parameters where optimization is performed. If it is not required or is redundant information contained in `model@pars` it can be set to `NULL`.

The functions `mcloglik.fd`, `mcloglik.fd.deriv` and `mcloglik.fd.grad` use the expressions for the spectral log-likelihood function where the parameter specified in `model@cpar` is concentrated out of the likelihood function.

For further information about the barrier term see *Bounds on parameters and barrier term* in the details section in `maxlik.fd.scoring`.

Arguments `inf` and `barrier` are not used by `mloglik.fd.grad` and `mcloglik.fd.grad` but they are needed in `maxlik.fd.optim`, where this function is passed as the gradient to be used by `optim` including the arguments `inf` and `barrier`.

Argument `xreg`. It is an optional list of constant terms. It is used by `maxlik.fd.optim` when analytical derivatives are employed and by `maxlik.fd.scoring`. It avoids computing some constant terms each time the function `mloglik.fd.grad` is called.

The list `xreg` should contain an element called `dxreg`, the external regressors differeneced by means of the differencing filter that renders stationarity in the model and the element `fft.xreg`, the Fourier transform of each regressor in `dxreg`.

The list `xreg` is not used by `mloglik.fd`. It is necessary to define this argument in the prototype of the function because when this function is passed to `optim` along with `mloglik.fd.grad`, the argument `xreg` is passed to `mloglik.fd` when it is defined in `optim` as an argument to be passed to `mloglik.fd.grad`.

Argument `xreg` is not currently implemented in functions with concentration of a parameter, `mloglik.fd`, `mcloglik.fd.deriv` and `mcloglik.fd.grad`.

Note: `modcovgrad` is not available when external regressors are defined in the input model, `model`.

Value

`mloglik.fd` returns a numeric value of the negative of the spectral log-likelihood evaluated at the parameter values defined in the input model `model` or at `x` if this argument is not `NULL`. If the value

happens to be NA or non-finite the value of argument inf is returned. This function is suited to be passed as the objective function to optim.

mloglik.fd.deriv returns a list containing a vector of the first order derivatives of the negative of the spectral likelihood and a matrix for the second order derivatives. Those derivative terms that are not requested by setting the corresponding argument to FALSE are set to NULL in the output list.

mloglik.fd.grad returns a numeric vector containing the gradient. This function is suited to be passed as the gradient function to optim.

mcloglik.fd, mcloglik.fd.deriv and mcloglik.fd.grad return the value of the same information as the other functions but for the concentrated likelihood function.

Note

mcloglik.fd.deriv is not currently implemented for model with non-null model@transPars or with a barrier term.

References

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

See Also

barrier.eval, logLik, maxlik.fd, stsm.

Examples

```
# define the local level model for Nile time series
pars <- c("var1" = 11000, "var2" = 1700)
m <- stsm.model(model = "local-level", y = Nile, pars = pars)
# 'mloglik.fd' returns the negative of the log-likelihood function
mloglik.fd(model = m)
# 'logLik' returns the value of the log-likelihood function
logLik(object = m, domain = "frequency")

# compare analytical and numerical derivatives
# more tests in file 'test-derivatives-mloglik-fd.R' in the
# folder 'inst' of the source package
system.time(da <- mloglik.fd.deriv(m, gradient = TRUE, hessian = TRUE))
dgn <- numDeriv::grad(func = mloglik.fd, x = m@pars, model = m)
dhn <- numDeriv::hessian(func = mloglik.fd, x = m@pars, model = m)
all.equal(as.vector(da$gradient), dgn)
all.equal(da$hessian, dhn)

# the same as above for the local level plus seasonal model and
# a sample simulated series
data("llmseas")
m <- stsm.model(model = "llm+seas", y = llmseas)
system.time(a <- mloglik.fd.deriv(model = m, gradient = TRUE, hessian = TRUE))
system.time(g <- numDeriv::grad(func = mloglik.fd, x = m@pars, model = m))
system.time(h <- numDeriv::hessian(func = mloglik.fd, x = m@pars, model = m))
```

```
all.equal(a$gradient, g, check.attributes = FALSE)
all.equal(a$hessian, h, check.attributes = FALSE)
```

---

mloglik.td *Time Domain Log-Likelihood Function and Derivatives*

Description

This function evaluates the negative of the time domain log-likelihood function of a linear Gaussian state space model by means of the Kalman filter.

Usage

```
mloglik.td(x, model,
  KF.version = eval(formals(KFKSDS::KalmanFilter)$KF.version),
  KF.args = list(), check.KF.args = TRUE,
  barrier = list(type = c("1", "2"), mu = 0), inf = 99999)

KFconvar(model, P0cov = FALSE, barrier = list(type = "1", mu = 0), debug = TRUE)

mloglik.td.deriv(model, gradient = TRUE, infomat = TRUE,
  KF.args = list(), version = c("1", "2"), kfres = NULL,
  convergence = c(0.001, length(model@y)))

mloglik.td.grad(x, model, KF.version, KF.args = list(),
  convergence = c(0.001, length(model@y)),
  check.KF.args, barrier, inf)
```

Arguments

x	a numeric vector containing the parameters of the model. This is an auxiliar argument so that this function can be used as input to optim.
model	an object of class stsm.
KF.version	character indicating the implementation of the Kalman filter to be used.
KF.args	a list of parameters to be passed to the function chosen to run the Kalman filter.
check.KF.args	logical. If TRUE, the elements passed in argument KF.args are checked for consistency with KF.version.
barrier	a list defining a barrier term to penalize parameter values close to the bounds m@lower and m@upper.
inf	a numeric indicating the value to be returned if the value of the log-likelihood function happens to be NA or non-finite.
P0cov	logical. If TRUE, values outside the diagonal of the covariance matrix of the initial state vector are set equal to the values in the diagonal, as done in StructTS.
debug	logical. If TRUE, the correctness of the result is double-checked.

`gradient`	logical. If `TRUE`, first order derivatives of the negative of the spectral log-likelihood function are returned.
`infomat`	logical. If `TRUE`, the information matrix of the spectral log-likelihood are returned.
`version`	a character indicating whether implementation `"2"` or `"2"` (the default) should be used.They yield the same result but are kept for debugging and comparison of timings. This argument may be removed in future versions.
`kfres`	optional list containing the elements involved in the Kalman filter as returned by `KF.deriv`.
`convergence`	a numeric vector containing two parameters to control the convergence of the Kalman filter. See `KF`.

Details

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

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

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

for $t = 1, \ldots, n$ and $a[1] \sim N(a0, P0)$. 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 r is the number of variance parameters in the state vector.

The Kalman filtering recursions for the model above are:

Prediction

$$a[t] = Ta[t-1]$$

$$P[t] = TP[t-1]T' + RVR'$$

$$v[t] = y[t] - Za[t]$$

$$F[t] = ZP[t]Z' + H$$

Updating

$$K[t] = P[t]Z'F[t]^{-1}$$

$$a[t] = a[t] + K[t]v[t]$$

$$P[t] = P[t] - K[t]ZP[t]'$$

for $t = 2, \ldots, n$, starting with $a[1]$ and $P[1]$ equal to `a0` and `P0`. $v[t]$ is the prediction error at observation in time t and $F[t]$ is the variance of $v[t]$.

The log-likelihood of the model for a given set of parameter values is:

$$logLik = -0.5log(2\pi) - 0.5\sum_{t=1}^{n} logF[t] + v[t]^2/F[t]$$

For details about the options than can be passed through argument `KF.args` see the documentation of the same argument of function `KalmanFilter` in package **KFKSDS**. For `mloglik.td.deriv`,

the only element that is used if provided in KF.args is P0cov (a logical indicating whether the covariance matrix of the initial state vector diagonal or not).

The argument x is an auxiliar vector that is necessary in some contexts. For example, the input to function optim must contain as first argument the vector of parameters where optimization is performed. If it is not required or is redundant information contained in model@pars it can be set to NULL.

For further information about the barrier term see *Bounds on parameters and barrier term* in the details section in maxlik.fd.scoring.

KFconvar evaluates the concentrated likelihood function. The likelihood is concentrated with respect to the parameter defined in model@cpar. The optimal value of the parameter that is concentrated out of the likelihood is:

$$s2 = (1/n)\sum_{t=1}^{n} v[t]/F[t]$$

and the concentrated likelihood function is given by:

$$clogLik = (n/2)log(2\pi + 1) + 0.5\sum_{t=1}^{n} log(f[t]) + (n/2)log(s2).$$

The gradient and the information matrix are calculated upon their corresponding analytical expressions.

Arguments KF.version, check.KF.args, barrier and inf are not used by mloglik.td.grad but they are needed in maxlik.td.optim, where this function is passed as the gradient to be used by optim including the arguments inf and barrier.

Value

The minus log-likelihood function evaluated at the parameter values defined in the input model model or at x if this argument is not NULL. If the value happens to be NA or non-finite the value of argument inf is returned. This function is suited to be passed as the objective function to optim.

KFconvar returns a list containing the element mll, the negative of the concentrated minus log-likelihood function and the element cpar, the optimal value of the parameter that is concentrated out of the likelihood.

mloglik.td.deriv returns a list containing a vector of the first order derivatives of the negative of the time domain likelihood function and a matrix for the information matrix. They are set to NULL if any of them are not requested.

mloglik.td.grad returns a numeric vector containing the gradient. This function is suited to be passed as the gradient function to optim.

References

Durbin, J. and Koopman, S. J. (2001). *Time Series Analysis by State Space Methods*. Oxford University Press.

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

See Also

barrier.eval, logLik, maxlik.td, stsm, KalmanFilter, KalmanLike.

Examples

```
# local level plus seasonal model for a sample simulated series
data("llmseas")
m <- stsm.model(model = "llm+seas", y = llmseas,
  pars = c("var1" = 300, "var2" = 10, "var3" = 100))
# evaluate the time domain likelihood function using
# excluding the contributions of the first 8 observations
mloglik.td(model = m, KF.version = "KFKSDS", KF.args = list(t0 = 9))

# compare analytical and numerical derivatives
# identical gradient up to a tolerance
a <- mloglik.td.deriv(m, infomat = TRUE)
g <- numDeriv::grad(func = mloglik.td, x = m@pars,
  model = m, KF.version = "KFKSDS")
h <- numDeriv::hessian(func = mloglik.td, x = m@pars,
  model = m, KF.version = "KFKSDS")
all.equal(a$gradient, g, check.attributes = FALSE)
```

---

sim-data *Simulated Data*

Description

Time series simulated from the local level plus seasonal model.

Usage

```
llmseas
```

Format

A time series of length 120.

Source

The scripts to generate this and other series used in the vignette are stored in the files 'stsm/inst/datagen-llm.R' and 'stsm/inst/datagen-llmseas.R'. The first series returned by the latter script is the series that is loaded by data("llmseas").

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, 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`.

References

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

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 class stsm, 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.

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 class `stsm`, 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, 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.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("xreg1" = 10), xreg = xreg)
m
m <- 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(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`, `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 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`.

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' and let the the remaining parameters
# be defined by default in the slots '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 'nopars'
# '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 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, 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))
```

---

stsmFit *Interface to Different Fitting Procedures*

Description

This function is an interface to the methods available in the package for fitting a structural time series model (maximum likelihood in the time and frequency domain via different optimization algorithms).

Usage

```
stsmFit(x, stsm.method = c("maxlik.fd.scoring", "maxlik.td.scoring",
  "maxlik.fd.optim", "maxlik.td.optim"), xreg = NULL, ...)
```

Arguments

x	an object of class stsm.
stsm.method	a character indicating the method to be used.
...	further arguments to be passed to the the function selected in stsm.method.
xreg	vector or matrix of external regresors.

Details

This interface is useful to simplify the code and reduce the number of arguments in functions that call some of those functions that can be specified through stsm.method. For example, the package **tsoutliers** uses: do.call("stsmFit", args = c(list(x = y, args.tsmethod)), where args.tsmethod is a list containing the arguments to be passed to stsmFit, which includes stsm.method. Thus, the code is simplified since no switch or if statements are necessary; the number of arguments is also reduced since those that are passed tot the function specified in stsm.method are gathered in a list.

The external regressors can be defined in the input object x of class stsm. This is the way the recommend specification for functions maxlik.td and maxlik.fd. This interface allows defining xreg as an argument passed to this function because it simplifies the code in some functions of package **tsoutliers**. If xreg and x@xreg are both not NULL and error is returned.

Value

A list of class stsmFit. See the section 'Value' in maxlik.fd.

See Also

maxlik.fd.scoring, maxlik.td.scoring, maxlik.fd.optim and maxlik.td.optim.

Index