Package ‘KFAS’

February 19, 2015

Version 1.0.4-1

Date 2014-06-06

Title Kalman Filter and Smoother for Exponential Family State Space Models.

Author Jouni Helske <jouni.helske@jyu.fi>

Maintainer Jouni Helske <jouni.helske@jyu.fi>

Depends R (>= 3.1.0)

Imports stats

Suggests MASS, testthat

Description Package KFAS provides functions for Kalman filtering, smoothing, forecasting and simulation of multivariate exponential family state space models with exact diffuse initialization when distributions of some or all elements of initial state vector are unknown.

License GPL (>= 2)

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-06-06 17:31:27

R topics documented:

`approxSSM` *Linear Gaussian Approximation for Exponential Family State Space Model*

Description

Function `approxSMM` computes the linear Gaussian approximation of a state space model where observations follow an exponential family distribution.

Usage

```
approxSSM(model, theta, maxiter = 50, tol = 1e-15)
```

Arguments

`model`	A non-Gaussian state space model object of class `SSModel`.
`theta`	Initial values for conditional mode theta.
`maxiter`	The maximum number of iterations used in approximation Default is 50.
`tol`	Tolerance parameter for convergence checks. Iterations are continued until $tol > abs(dev_{old} - dev_{new})/(abs(dev_{new}) + 0.1))$.

Details

The linear Gaussian approximating model is defined by

$$\tilde{y}_t = Z_t \alpha_t + \epsilon_t, \quad \epsilon_t \sim N(0, \tilde{H}_t),$$

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t, \quad \eta_t \sim N(0, Q_t),$$

and $\alpha_1 \sim N(a_1, P_1)$, where $\tilde{y}$ and $\tilde{H}$ are chosen in a way that the linear Gaussian approximating model has the same conditional mode of $\theta = Z\alpha$ given the observations y as the original non-gaussian model. Models also have a same curvature at the mode.

The approximation of the exponential family state space model is based on iterative weighted least squares method, see McCullagh and Nelder (1983) p.31 and Durbin Koopman (2012) p. 243.

Value

An object which contains the approximating Gaussian state space model with following additional components:

`thetahat`	Mode of $p(\theta\|y)$.
`iterations`	Number of iterations used.

See Also

Importance sampling of non-Gaussian state space models `importanceSSM`, construct a `SSModel` object `SSModel`, and examples in `KFAS`.

`artransform` *Mapping real valued parameters to stationary region*

Description

Function `artransform` transforms p real valued parameters to stationary region of pth order autoregressive process using parametrization suggested by Jones (1980), except the same modification is done as in `arima`.

Usage

```
artransform(param)
```

Arguments

`param`	Real valued parameters for the transformation.

Value

transformed The parameters satisfying the stationary constrains.

`boat` *Oxford-Cambridge boat race results 1829-2000*

Description

Results of the annual boat race between universities of Oxford (0) and Cambridge (1).

Format

A time series object containing 172 observations.

Source

http://www.ssfpack.com/DKbook.html

References

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

`coef.KFS` *Extract Estimated States of State Space Model*

Description

Extracts the estimates states from output of `KFS`. For non-Gaussian models without simulation, these are estimates of conditional modes of states. For Gaussian models and non-Gaussian models with importance sampling, these are estimates of conditional means of states.

Usage

```
## S3 method for class 'KFS'
coef(object, start = NULL, end = NULL, filtered = FALSE,
  ...)
```

Arguments

`object`	An object of class `KFS`.
`start`	The start time of the period of interest. Defaults to first time point of the object
`end`	The end time of the period of interest. Defaults to the last time point of the object.
`filtered`	Logical, return filtered instead of smoothed estimates of state vector. Default is `FALSE`.
`...`	Ignored.

Value

Multivariate time series containing estimates states.

`deviance.KFS` *Deviance of a State Space Model*

Description

Returns the deviance of a object of class `KFS`.

Usage

```
## S3 method for class 'KFS'
deviance(object, ...)
```

Arguments

`object`	An object of class `KFS`.
`...`	Ignored.

Value

The value of the deviance extracted from object.

`fitSSM` *Maximum Likelihood Estimation of a State Space Model*

Description

Function `fitSSM` finds the maximum likelihood estimates for unknown parameters of an arbitary state space model, given the user-defined model updating function.

Usage

```
fitSSM(model, inits, updatefn, checkfn, ...)
```

Arguments

`inits`	Initial values for `optim`
`model`	Model object of class `SSModel`.
`updatefn`	User defined function which updates the model given the parameters. Must be of form `updatefn(pars, model,...)`, i.e. must contain ellipsis `...`. If not supplied, a default function is used, which estimates the values marked as NA in time invariant covariance matrices Q and H.
`checkfn`	Optional function for model checking. If supplied, after updating the model, if `checkfn(model)` returns TRUE, -log-likelihood is computed, otherwise `.Machine$double.xmax` is returned. See examples. If not supplied, check.model=TRUE is used for checking possible NA or Inf values, see ?logLik.SSModel.
`...`	Further arguments for functions `optim`, `updatefn` and `logLik.SSModel`, such as `method='BFGS'`.

Value

A list with elements

`optim.out`	Output from function `optim`.
`model`	Model with estimated parameters.

Examples

```
# Example function for updating covariance matrices H and Q
# (also used as a default function in fitSSM)

updatefn <- function(pars,model,...){
Q<-as.matrix(model$Q[,,1])
naQd  <- which(is.na(diag(Q)))
naQnd <- which(upper.tri(Q[naQd,naQd]) & is.na(Q[naQd,naQd]))
Q[naQd,naQd][lower.tri(Q[naQd,naQd])] <- 0
diag(Q)[naQd] <- exp(0.5 * pars[1:length(naQd)])
Q[naQd,naQd][naQnd] <- pars[length(naQd)+1:length(naQnd)]
model$Q[naQd,naQd,1] <- crossprod(Q[naQd,naQd])
if(!identical(model$H,'Omitted')){
   H<-as.matrix(model$H[,,1])
   naHd  <- which(is.na(diag(H)))
   naHnd <- which(upper.tri(H[naHd,naHd]) & is.na(H[naHd,naHd]))
   H[naHd,naHd][lower.tri(H[naHd,naHd])] <- 0
   diag(H)[naHd] <- exp(0.5 * pars[length(naQd)+length(naQnd)+1:length(naHd)])
   H[naHd,naHd][naHnd] <- pars[length(naQd)+length(naQnd)+length(naHd)+1:length(naHnd)]
   model$H[naHd,naHd,1] <- crossprod(H[naHd,naHd])
 }

 model
}

# Example function for checking the validity of covariance matrices.
```

```
checkfn <- function(model){
  #test positive semidefiniteness of H and Q
  inherits(try(ldl(model$H[,,1]),TRUE),'try-error') ||
  inherits(try(ldl(model$Q[,,1]),TRUE),'try-error')
}
```

`fitted.KFS` *Extract Fitted Values of State Space Model*

Description

Extracts fitted values from output of `KFS`.

Usage

```
## S3 method for class 'KFS'
fitted(object, start = NULL, end = NULL, filtered = FALSE,
  ...)
```

Arguments

`object`	An object of class `KFS`.
`start`	The start time of the period of interest. Defaults to first time point of the object.
`end`	The end time of the period of interest. Defaults to the last time point of the object.
`filtered`	Logical, return filtered instead of smoothed estimates of mean vector. Default is `FALSE`.
`...`	Ignored.

Value

Multivariate time series containing fitted values.

`GlobalTemp` *Two series of average global temperature deviations for years 1880-1987*

Description

This data set contains two series of average global temperature deviations for years 1880-1987. These series are same as used in Shumway and Stoffer (2006), where they are known as HL and Folland series. For more details, see Shumway and Stoffer (2006, p. 327).

Format

A time series object containing 108 times 2 observations.

Source

http://lib.stat.cmu.edu/general/stoffer/tsa2/

References

Shumway, Robert H. and Stoffer, David S. (2006). Time Series Analysis and Its Applications: With R examples.

`hatvalues.KFS` *Extract Hat Values from KFS Output*

Description

Extract hat values from KFS output, when `KFS` was run with signal (non-Gaussian case) or mean smoothing (Gaussian case).

Usage

```
## S3 method for class 'KFS'
hatvalues(model, ...)
```

Arguments

`model`	An object of class `KFS`.
`...`	Ignored.

Details

Hat values are the diagonal elements of `V_t/H_t` where V_t is the covariance matrix of signal/mean at time t and H_t is the covariance matrix of disturbance vector ϵ of (approximating) Gaussian model at time t.

Value

Multivariate time series containing hat values.

importanceSSM *Importance Sampling of Exponential Family State Space Model*

Description

Importance Sampling of Exponential Family State Space Model.

Usage

```
importanceSSM(model, type = c("states", "signals"), filtered = FALSE,
  nsim = 1000, save.model = FALSE, theta, antithetics = FALSE,
  maxiter = 50)
```

Arguments

`model`	Exponential family state space model of class `SSModel`.
`type`	What to simulate, `'states'` or `'signals'`. Default is `'states'`
`filtered`	Simulate from $p(\alpha_t \vert y_{t-1}, ..., y_1)$ instead of $p(\alpha \vert y)$.
`nsim`	Number of independent samples. Default is 1000.
`save.model`	Return the original model with the samples. Default is FALSE.
`theta`	Initial values for conditional mode theta.
`antithetics`	Logical. If TRUE, two antithetic variables are used in simulations, one for location and another for scale. Default is FALSE.
`maxiter`	Maximum number of iterations used in linearisation. Default is 50.

Details

Function `importanceSSM` simulates states or signals of the exponential family state space model conditioned with the observations, returning the simulated samples of the states/signals with the corresponding importance weights.

Function can use two antithetic variables, one for location and other for scale, so output contains four blocks of simulated values which correlate which each other (ith block correlates negatively with (i+1)th block, and positively with (i+2)th block etc.).

Value

A list containing elements `samples`, `weights` and `model` (if `save.model==TRUE`).

`is.SSModel` *Test whether object is a valid* `SSModel` *object*

Description

Function `is.SSModel` tests whether the object is a valid `SSModel` object.

Usage

```
is.SSModel(object, na.check = FALSE, return.logical = TRUE)
```

Arguments

`object`	An object to be tested.
`na.check`	Test the system matrices for NA and infinite values. Default is `FALSE`.
`return.logical`	If `FALSE`, error is given if the the model is not a valid `SSModel` object. Otherwise logical value is returned. Defaults to `FALSE`.

Details

Note that the validity of the values in `y` and `Z` are not tested. These can contain NA values (but not infinite values), with condition that when `Z[i,,t]` contains NA value, the corresponding `y[t,i]` must also have NA value. In this case `Z[i,,t]` is not referenced in filtering and smoothing, and algorithms works properly. Note also that this does result NA values in `thetahat`, so it could be beneficial to use for example zeroes in place of NA values in Z, making first sure that the above condition is met.

Value

Logical value or nothing, depending on the value of `return.logical`.

`KFAS` *KFAS: Functions for Gaussian and Non-Gaussian State Space Models*

Description

Package KFAS contains functions for Kalman filtering, smoothing and simulation of linear state space models with exact diffuse initialization.

Details

The linear gaussian state space model is given by

$$y_t = Z_t \alpha_t + \epsilon_t,$$

$$\alpha_{t+1} = T_t \alpha_t + R_t \eta_t,$$

where $\epsilon_t \sim N(0, H_t)$, $\eta_t \sim N(0, Q_t)$ and $\alpha_1 \sim N(a_1, P_1)$ independently of each other.

All system and covariance matrices Z, H, T, R and Q can be time-varying, and partially or totally missing observations y_t are allowed.

Covariance matrices H and Q has to be positive semidefinite (although this is not checked).

Dimensions of system matrices are

Z	$p \times m \times 1$ or $p \times m \times n$ in time varying case
H	$p \times p \times 1$ or $p \times p \times n$ in time varying case (Omitted in non-gaussian models)
T	$m \times m \times 1$ or $m \times m \times n$ in time varying case
R	$m \times k \times 1$ or $m \times k \times n$ in time varying case
Q	$k \times k \times 1$ or $k \times k \times n$ in time varying case
u	$n \times p$ (Omitted in gaussian models)

In case of any of the series in model is defined as non-gaussian, the observation equation is of form

$$\prod_i^p p_i(y_{i,t} | \theta_t)$$

with $\theta_{i,t} = Z_{i,t} \alpha_t$ being one of the following:

If observations $y_{i,1}, \ldots, y_{i,n}$ are distributed as $N(\mu_t, u_t)$, then $\theta_t = \mu_t$. Note that now variances are defined using `u`, not `H`. If correlation between gaussian observation equations is needed, one can use $u_t = 0$ and add correlating disturbances into state equation (although care is needed when making inferences as then $y_t = \theta_t$)

If observations are distributed as $Poisson(u_t \lambda_t)$, where u_t is offset term, then $\theta_t = log(u_t \lambda_t)$.

If observations are distributed as $binomial(u_t, \pi_t)$, then $\theta_t = log[\pi_t/(1 - \pi_t)]$, where π_t is the probability of success at time t.

If observations are distributed as $gamma(u_t, \mu_t)$, then $\theta_t = log(\mu_t)$, where $\mu[t]$ is the mean parameter and u is the shape parameter.

If observations are distributed as $negativebinomial(u_t, \mu_t)$ (with expected value μ_t and variance $\mu_t + \mu_t^2/u_t$, see `dbinom`), then $\theta_t = log[\mu_t]$.

For exponential family models $u_t = 1$ as a default. For completely gaussian models, parameter is omitted.

For the unknown elements of initial state vector a_1, KFS uses exact diffuse initialization by Koopman and Durbin (2000, 2001, 2003), where the unknown initial states are set to have a zero mean and infinite variance, so

$$P_1 = P_{*,1} + \kappa P_{\infty,1},$$

with κ going to infinity and $P_{\infty,1}$ being diagonal matrix with ones on diagonal elements corresponding to unknown initial states.

Diffuse phase is continued until rank of $P_{\infty,t}$ becomes zero. Rank of P_{∞} decreases by 1, if $F_{\infty} > tol > 0$. Usually the number of diffuse time points equals the number unknown elements of initial state vector, but missing observations or time-varying Z can affect this. See Koopman and Durbin (2000, 2001, 2003) for details for exact diffuse and non-diffuse filtering.

To lessen the notation and storage space, KFAS uses letters P, F and K for non-diffuse part of the corresponding matrices, omitting the asterisk in diffuse phase.

All functions of KFAS use the univariate approach (also known as sequential processing, see Anderson and Moore (1979)) which is from Koopman and Durbin (2000, 2001). In univariate approach the observations are introduced one element at the time. Therefore the prediction error variance matrices F and Finf does not need to be non-singular, as there is no matrix inversions in univariate approach algorithm. This provides more stable and possibly more faster filtering and smoothing than normal multivariate Kalman filter algorithm. If covariance matrix H is not diagonal, it is possible to transform the model by either using LDL decomposition on H, or augmenting the state vector with ϵ disturbances. See `transformSSM` for more details.

References

Koopman, S.J. and Durbin J. (2000). Fast filtering and smoothing for non-stationary time series models, Journal of American Statistical Assosiation, 92, 1630-38.

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

Koopman, S.J. and Durbin J. (2003). Filtering and smoothing of state vector for diffuse state space models, Journal of Time Series Analysis, Vol. 24, No. 1.

#' Shumway, Robert H. and Stoffer, David S. (2006). Time Series Analysis and Its Applications: With R examples.

Examples

```
# Example of local level model for Nile series

modelNile<-SSModel(Nile~SSMtrend(1,Q=list(matrix(NA))),H=matrix(NA))
modelNile
modelNile<-fitSSM(inits=c(log(var(Nile)),log(var(Nile))),model=modelNile,
                  method='BFGS',control=list(REPORT=1,trace=1))$model
# Filtering and state smoothing
out<-KFS(modelNile,filtering='state',smoothing='state')
out

# Confidence and prediction intervals for the expected value and the observations.
# Note that predict uses original model object, not the output from KFS.
conf<-predict(modelNile,interval='confidence')
pred<-predict(modelNile,interval='prediction')

ts.plot(cbind(Nile,pred,conf[,-1]),col=c(1:2,3,3,4,4),
        ylab='Predicted Annual flow', main='River Nile')
```

```
# Missing observations, using same parameter estimates

y<-Nile
y[c(21:40,61:80)]<-NA
modelNile<-SSModel(y~SSMtrend(1,Q=list(modelNile$Q)),H=modelNile$H)

out<-KFS(modelNile,filtering='mean',smoothing='mean')

# Filtered and smoothed states
plot.ts(cbind(y,fitted(out,filtered=TRUE),fitted(out)), plot.type='single',
        col=1:3, ylab='Predicted Annual flow', main='River Nile')


# Example of multivariate local level model with only one state
# Two series of average global temperature deviations for years 1880-1987
# See Shumway and Stoffer (2006), p. 327 for details

data(GlobalTemp)

model<-SSModel(GlobalTemp~SSMtrend(1,Q=NA,type='common'),H=matrix(NA,2,2))

# Estimating the variance parameters
inits<-chol(cov(GlobalTemp))[c(1,4,3)]
inits[1:2]<-log(inits[1:2])
fit<-fitSSM(inits=c(0.5*log(.1),inits),model=model,method='BFGS')

out<-KFS(fit$model)

ts.plot(cbind(model$y,coef(out)),col=1:3)
legend('bottomright',legend=c(colnames(GlobalTemp), 'Smoothed signal'), col=1:3, lty=1)



# Seatbelts data
## Not run:
model<-SSModel(log(drivers)~SSMtrend(1,Q=list(NA))+
               SSMseasonal(period=12,sea.type='trigonometric',Q=NA)+
               log(PetrolPrice)+law,data=Seatbelts,H=NA)

# As trigonometric seasonal contains several disturbances which are all
# identically distributed, default behaviour of fitSSM is not enough,
# as we have constrained Q. We can either provide our own
# model updating function with fitSSM, or just use optim directly:

# option 1:
ownupdatefn<-function(pars,model,...){
  model$H[]<-exp(pars[1])
  diag(model$Q[,,1])<-exp(c(pars[2],rep(pars[3],11)))
  model #for option 2, replace this with -logLik(model) and call optim directly
}
```

```
fit<-fitSSM(inits=log(c(var(log(Seatbelts[,'drivers'])),0.001,0.0001)),
            model=model,updatefn=ownupdatefn,method='BFGS')

out<-KFS(fit$model,smoothing=c('state','mean'))
out
ts.plot(cbind(out$model$y,fitted(out)),lty=1:2,col=1:2,
main='Observations and smoothed signal with and without seasonal component')
lines(signal(out,states=c("regression","trend"))$signal,col=4,lty=1)
legend('bottomleft',
legend=c('Observations', 'Smoothed signal','Smoothed level'),
col=c(1,2,4), lty=c(1,2,1))


# Multivariate model with constant seasonal pattern,
# using the the seat belt law dummy only for the front seat passangers,
# and restricting the rank of the level component by using custom component

# note the small inconvinience in regression component,
# you must remove the intercept from the additional regression parts manually

model<-SSModel(log(cbind(front,rear))~ -1 + log(PetrolPrice) + log(kms)
               + SSMregression(~-1+law,data=Seatbelts,index=1)
               + SSMcustom(Z=diag(2),T=diag(2),R=matrix(1,2,1),
                           Q=matrix(1),P1inf=diag(2))
               + SSMseasonal(period=12,sea.type='trigonometric'),
                 data=Seatbelts,H=matrix(NA,2,2))

likfn<-function(pars,model,estimate=TRUE){
  model$H[,,1]<-exp(0.5*pars[1:2])
  model$H[1,2,1]<-model$H[2,1,1]<-tanh(pars[3])*prod(sqrt(exp(0.5*pars[1:2])))
  model$R[28:29]<-exp(pars[4:5])
  if(estimate) return(-logLik(model))
  model
}
fit<-optim(f=likfn,p=c(-7,-7,1,-1,-3),method='BFGS',model=model)
model<-likfn(fit$p,model,estimate=FALSE)
model$R[28:29,,1]%*%t(model$R[28:29,,1])
model$H

out<-KFS(model)
out
ts.plot(cbind(signal(out,states=c('custom','regression'))$signal,model$y),col=1:4)

# For confidence or prediction intervals, use predict on the original model
pred <- predict(model,states=c('custom','regression'),interval='prediction')
ts.plot(pred$front,pred$rear,model$y,col=c(1,2,2,3,4,4,5,6),lty=c(1,2,2,1,2,2,1,1))

## End(Not run)

 ## Not run:
# Poisson model
model<-SSModel(VanKilled~law+SSMtrend(1,Q=list(matrix(NA)))+
               SSMseasonal(period=12,sea.type='dummy',Q=NA),
```

```
               data=Seatbelts, distribution='poisson')

# Estimate variance parameters
fit<-fitSSM(inits=c(-4,-7,2), model=model,method='BFGS')

model<-fit$model

# use approximating model, gives posterior mode of the signal and the linear predictor
out_nosim<-KFS(model,smoothing=c('signal','mean'),nsim=0)
# State smoothing via importance sampling
out_sim<-KFS(model,smoothing=c('signal','mean'),nsim=1000)

out_nosim
out_sim

## End(Not run)

# Example of generalized linear modelling with KFS

# Same example as in ?glm
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
print(d.AD <- data.frame(treatment, outcome, counts))
glm.D93 <- glm(counts ~ outcome + treatment, family = poisson())


model<-SSModel(counts ~ outcome + treatment, data=d.AD,
               distribution = 'poisson')

out<-KFS(model)
coef(out,start=1,end=1)
coef(glm.D93)

summary(glm.D93)$cov.s
out$V[,,1]

outnosim<-KFS(model,smoothing=c('state','signal','mean'))
set.seed(1)
outsim<-KFS(model,smoothing=c('state','signal','mean'),nsim=1000)


## linear
# GLM
glm.D93$linear.predictor
# approximate model, this is the posterior mode of p(theta|y)
c(outnosim$thetahat)
# importance sampling on theta,  gives E(theta|y)
c(outsim$thetahat)



## predictions on response scale
```

```
# GLM
fitted(glm.D93)
# approximate model with backtransform, equals GLM
c(fitted(outnosim))
# importance sampling on exp(theta)
fitted(outsim)

# prediction variances on link scale
# GLM
as.numeric(predict(glm.D93,type='link',se.fit=TRUE)$se.fit^2)
# approx, equals to GLM results
c(outnosim$V_theta)
# importance sampling on theta
c(outsim$V_theta)


# prediction variances on response scale
# GLM
as.numeric(predict(glm.D93,type='response',se.fit=TRUE)$se.fit^2)
# approx, equals to GLM results
c(outnosim$V_mu)
# importance sampling on theta
c(outsim$V_mu)

## Not run:
data(sexratio)
model<-SSModel(Male~SSMtrend(1,Q=list(NA)),u=sexratio[,'Total'],data=sexratio,
               distribution='binomial')
fit<-fitSSM(model,inits=-15,method='BFGS',control=list(trace=1,REPORT=1))
fit$model$Q #1.107652e-06

# Computing confidence intervals in response scale
# Uses importance sampling on response scale (4000 samples with antithetics)

pred<-predict(fit$model,type='response',interval='conf',nsim=1000)

ts.plot(cbind(model$y/model$u,pred),col=c(1,2,3,3),lty=c(1,1,2,2))

# Now with sex ratio instead of the probabilities:
imp<-importanceSSM(fit$model,nsim=1000,antithetics=TRUE)
sexratio.smooth<-numeric(length(model$y))
sexratio.ci<-matrix(0,length(model$y),2)
w<-imp$w/sum(imp$w)
for(i in 1:length(model$y)){
 sexr<-exp(imp$sample[i,1,])
 sexratio.smooth[i]<-sum(sexr*w)
 oo<-order(sexr)
 sexratio.ci[i,]<-c(sexr[oo][which.min(abs(cumsum(w[oo]) - 0.05))],
                    + sexr[oo][which.min(abs(cumsum(w[oo]) - 0.95))])
}

# Same by direct transformation:
out<-KFS(fit$model,smoothing='signal',nsim=1000)
```

```
sexratio.smooth2 <- exp(out$thetahat)
sexratio.ci2<-exp(c(out$thetahat)
                  + qnorm(0.025) * sqrt(drop(out$V_theta))%o%c(1, -1))

ts.plot(cbind(sexratio.smooth,sexratio.ci,sexratio.smooth2,sexratio.ci2),
        col=c(1,1,1,2,2,2),lty=c(1,2,2,1,2,2))

## End(Not run)
# Example of Cubic spline smoothing
## Not run:
require(MASS)
data(mcycle)

model<-SSModel(accel~-1+SSMcustom(Z=matrix(c(1,0),1,2),
                                 T=array(diag(2),c(2,2,nrow(mcycle))),
                                 Q=array(0,c(2,2,nrow(mcycle))),
                                 P1inf=diag(2),P1=diag(0,2)),data=mcycle)

model$T[1,2,]<-c(diff(mcycle$times),1)
model$Q[1,1,]<-c(diff(mcycle$times),1)^3/3
model$Q[1,2,]<-model$Q[2,1,]<-c(diff(mcycle$times),1)^2/2
model$Q[2,2,]<-c(diff(mcycle$times),1)


updatefn<-function(pars,model,...){
  model$H[]<-exp(pars[1])
  model$Q[]<-model$Q[]*exp(pars[2])
  model
}

fit<-fitSSM(model,inits=c(4,4),updatefn=updatefn,method="BFGS")

pred<-predict(fit$model,interval="conf",level=0.95)
plot(x=mcycle$times,y=mcycle$accel,pch=19)
lines(x=mcycle$times,y=pred[,1])
lines(x=mcycle$times,y=pred[,2],lty=2)
lines(x=mcycle$times,y=pred[,3],lty=2)

## End(Not run)
```

KFS	*Kalman Filter and Smoother with Exact Diffuse Initialization for Exponential Family State Space Models*

Description

Performs Kalman filtering and smoothing with exact diffuse initialization using univariate approach for exponential family state space models.

Usage

```
KFS(model, filtering, smoothing, simplify = TRUE, transform = c("ldl",
  "augment"), nsim = 0, theta, maxiter = 50, convtol = 1e-15)
```

Arguments

`model`	Object of class `SSModel`.
`filtering`	Types of filtering. Possible choices are 'state', 'signal', 'mean', and 'none'. Default is 'state' for Gaussian and 'none' for non-Gaussian models. Multiple values are allowed. Note that for Gaussian models, signal is mean. Note that filtering for non-Gaussian models with importance sampling can be very slow with large models. Also in approximating mean filtering only diagonals of P_mu are returned.
`smoothing`	Types of smoothing. Possible choices are 'state', 'signal', 'mean', 'disturbance' and 'none'. Default is 'state' and 'mean'. For non-Gaussian models, option 'disturbance' is not supported, and for Gaussian models option 'mean' is identical to 'signal'. Multiple values are allowed.
`simplify`	If FALSE and model is completely Gaussian, KFS returns some generally not so interesting variables from filtering and smoothing. Default is TRUE.
`transform`	How to transform the model in case of non-diagonal covariance matrix H. Defaults to `'ldl'`. See function `transformSSM` for details.
`nsim`	The number of independent samples. Only used for non-Gaussian model. Default is 0, which computes the approximating Gaussian model by `approxSSM` and performs the usual Gaussian smoothing so that the smoothed state estimates equals to the conditional mode of $p(\alpha_t\|y)$.
`theta`	Initial values for conditional mode theta. Only used for non-Gaussian model.
`maxiter`	The maximum number of iterations used in approximation Default is 50. Only used for non-Gaussian model.
`convtol`	Tolerance parameter for convergence checks for Gaussian approximation. Iterations are continued until $tol > abs(dev_{old} - dev_{new})/(abs(dev_{new}) + 0.1))$.

Details

Notice that in case of multivariate observations, `v`, `F`, `Finf`, `K` and `Kinf` are usually not the same as those calculated in usual multivariate Kalman filter. As filtering is done one observation element at the time, the elements of prediction error v_t are uncorrelated, and `F`, `Finf`, `K` and `Kinf` contain only the diagonal elemens of the corresponding covariance matrices.

In rare cases of a diffuse initialization phase with highly correlated states, cumulative rounding errors in computing `Finf` and `Pinf` can sometimes cause the diffuse phase end too early. Changing the tolerance parameter `tol` of the model (see `SSModel`) to smaller (or larger) should help.

In case of non-Gaussian models with `nsim=0`, the smoothed estimates relate the conditional mode of $p(\alpha|y)$, and are equivalent with the results from generalized linear models. When using importance sampling (`nsim>0`), results correspond to the conditional mean.

Value

What KFS returns depends on the arguments `filtering`, `smoothed` and `simplify`, and whether the model is Gaussian or not:

`model`	Original state space model.
`KFS_transform`	Type of H after possible transformation.
`logLik`	Value of the log-likelihood function. Only computed for Gaussian models.
`a`	One step predictions of states, $a_t = E(\alpha_t \| y_{t-1}, \ldots, y_1)$.
`P`	Covariance matrices (of the non-diffuse parts) of predicted states, $P_t = Cov(\alpha_t \| y_{t-1}, \ldots, y_1)$.
`Pinf`	Diffuse part of P_t. Only returned for Gaussian models.
`t`	Filtered estimates of signals, $E(Z_t \alpha_t \| y_{t-1}, \ldots, y_1)$.
`P_theta`	Covariances $Var(Z[t]\alpha_t \| y_{t-1}, \ldots, y_1)$..
`m`	Filtered estimates of $f(\theta_t) \| y_{t-1}, \ldots, y_1)$, where f is the inverse link function.
`P_mu`	Covariances $Cov(f(\theta_t) \| y_{t-1}, \ldots, y_1)$. If `nsim=0`, only diagonal elements (variances) are computed, using the delta method.
`alphahat`	Smoothed estimates of states, $E(\alpha_t \| y_1, \ldots, y_n)$.
`V`	Covariances $Var(\alpha_t \| y_1, \ldots, y_n)$.
`thetahat`	Smoothed estimates of signals, $E(Z_t \alpha_t \| y_1, \ldots, y_n)$.
`V_theta`	Covariances $Var(Z[t]\alpha_t \| y_1, \ldots, y_n)$..
`muhat`	Smoothed estimates of $f(\theta_t) \| y_1, \ldots, y_n)$, where f is the inverse link function.
`V_mu`	Covariances $Cov(f(\theta_t) \| y_1, \ldots, y_n)$. If `nsim=0`, only diagonal elements (variances) are computed, using the delta method.
`etahat`	Smoothed disturbance terms $E(\eta_t \| y_1, \ldots, y_n)$.
`V_eta`	Covariances $Var(\eta_t \| y_1, \ldots, y_n)$.
`epshat`	Smoothed disturbance terms $E(\epsilon_{t,i} \| y_1, \ldots, y_n)$. Note that due to the possible diagonalization these are on transformed scale.
`V_eps`	Diagonal elements of $Var(\epsilon_t \| y_1, \ldots, y_n)$. Note that due to the diagonalization the off-diagonal elements are zero.
`iterations`	The number of iterations used in linearization of non-Gaussian model.
`v`	Prediction errors $v_{t,i} = y_{t,i} - Z_{i,t} a_{t,i}, i = 1, \ldots, p$, where $a_{t,i} = E(\alpha_t \| y_{t,i-1}, \ldots, y_{t,1}, \ldots, y_{1,1})$. Only returned for Gaussian models.
`F`	Prediction error variances $Var(v_{t,i})$. Only returned for Gaussian models.
`Finf`	Diffuse part of F_t. Only returned for Gaussian models.
`d`	The last index of diffuse phase, i.e. the non-diffuse phase began from time $d+1$. Only returned for Gaussian models.
`j`	The index of last $y_{i,t}$ of diffuse phase. Only returned for Gaussian models.

In addition, if argument `simplify=FALSE`, list contains following components:

`K`	Covariances $Cov(\alpha_{t,i}, y_{t,i} \| y_{t,i-1}, \ldots, y_{t,1}, y_{t-1}, \ldots, y_1), \quad i = 1, \ldots, p$.
`Kinf`	Diffuse part of K_t.

`r`	Weighted sums of innovations $v_{t+1}, \ldots, v_n$. Notice that in literature t in r_t goes from $0, \ldots, n$. Here $t = 1, \ldots, n+1$. Same applies to all r and N variables.
`r0, r1`	Diffuse phase decomposition of r_t.
`N`	Covariances $Var(r_t)$.
`N0, N1, N2`	Diffuse phase decomposition of N_t.

References

Koopman, S.J. and Durbin J. (2000). Fast filtering and smoothing for non-stationary time series models, Journal of American Statistical Assosiation, 92, 1630-38.

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

Koopman, S.J. and Durbin J. (2003). Filtering and smoothing of state vector for diffuse state space models, Journal of Time Series Analysis, Vol. 24, No. 1.

`ldl` *LDL Decomposition of a Matrix*

Description

Function `ldl` computes the LDL decomposition of a positive semidefinite matrix.

Usage

```
ldl(x, tol = max(abs(diag(x))) * .Machine$double.eps)
```

Arguments

`x`	Symmetrix matrix.
`tol`	Tolerance parameter for LDL decomposition, determines which diagonal values are counted as zero. Same value is used in isSymmetric function.

Value

Transformed matrix with D in diagonal, L in strictly lower diagonal and zeros on upper diagonal.

`logLik.SSModel` *Log-likelihood of the State Space Model.*

Description

Function `logLik.SSmodel` computes the log-likelihood value of a state space model.

Usage

```
## S3 method for class 'SSModel'
logLik(object, nsim = 0, antithetics = TRUE, theta,
  check.model = FALSE, transform = c("ldl", "augment"), maxiter = 50,
  seed, convtol = 1e-08, ...)
```

Arguments

`object`	State space model of class `SSModel`.
`nsim`	Number of independent samples used in estimating the log-likelihood of the non-Gaussian state space model. Default is 0, which gives good starting value for optimization. Only used for non-Gaussian model.
`antithetics`	Logical. If TRUE, two antithetic variables are used in simulations, one for location and another for scale. Default is TRUE. Only used for non-Gaussian model.
`theta`	Initial values for conditional mode theta. Only used for non-Gaussian model.
`check.model`	Logical. If TRUE, function `is.SSModel` is called before computing the likelihood. Default is FALSE.
`transform`	How to transform the model in case of non-diagonal covariance matrix H. Defaults to `'ldl'`. See function `transformSSM` for details.
`maxiter`	The maximum number of iterations used in linearisation. Default is 50. Only used for non-Gaussian model.
`seed`	The value is used as a seed via set.seed function. Only used for non-Gaussian model.
`convtol`	Tolerance parameter for convergence checks for Gaussian approximation. Iterations are continued until $tol > abs(dev_{old} - dev_{new})/(abs(dev_{new}) + 0.1))$.
`...`	Ignored.

Value

log-likelihood of the state space model.

predict.SSModel *State Space Model Predictions*

Description

Function `predict.SSModel` predicts the future observations of a state space model of class `SSModel`

Usage

```
## S3 method for class 'SSModel'
predict(object, newdata, n.ahead, interval = c("none",
  "confidence", "prediction"), level = 0.95, type = c("response", "link"),
  states = NULL, se.fit = FALSE, nsim = 0, prob = TRUE, maxiter = 50,
  ...)
```

Arguments

`object`	Object of class `SSModel`.
`newdata`	A compatible `SSModel` object to be added in the end of the old object for which the predictions are required. If omitted, predictions are either for the whole data (fitted values), or if argument `n.ahead` is given, `n.ahead` time steps ahead.
`n.ahead`	Number of steps ahead at which to predict. Only used if `newdata` is omitted. Note that when using `n.ahead`, object cannot contain time varying system matrices.
`interval`	Type of interval calculation.
`level`	Confidence level for intervals.
`type`	Scale of the prediction, `'response'` or `'link'`.
`states`	Which states are used in computing the predictions. Either a numeric vector containing the indices of the corresponding states, or a character vector defining the types of the corresponding states. Possible choices are “all”, “arima”, “custom”, “cycle”, “seasonal”, “trend”, or “regression”. These can be combined. Default is “all”.
`nsim`	Number of independent samples used in importance sampling. Used only for non-Gaussian models.
`se.fit`	If TRUE, standard errors are computed. Default is FALSE.
`prob`	if TRUE (default), the predictions in binomial case are probabilities instead of counts.
`maxiter`	The maximum number of iterations used in approximation Default is 50. Only used for non-Gaussian model.
`...`	Ignored.

Details

For non-Gaussian models, the results depend whether importance sampling is used (`nsim>0`). without simulations, the confidence intervals in response scale are computed in linear predictor scale, and then transformed to response scale. The prediction intervals are not supported. With importance sampling, the confidence intervals are computed as the empirical quantiles from the weighted sample, whereas the prediction intervals contain additional step of simulating the response variables from the sampling distribution $p(y|\theta^i)$.

If no simulations are used, the standard errors in response scale are computed using delta method.

Value

A matrix or list of matrices containing the predictions, and optionally standard errors.

Examples

```
## Not run:
set.seed(1)
x<-runif(n=100,min=1,max=3)
y<-rpois(n=100,lambda=exp(-1+x))
model<-SSModel(y~x,distribution="poisson")
xnew<-seq(0.5,3.5,by=0.1)
newdata<-SSModel(rep(NA,length(xnew))~xnew,distribution="poisson")
pred<-predict(model,newdata=newdata,interval="prediction",level=0.9,nsim=1000)
plot(x=x,y=y,pch=19,ylim=c(0,25),xlim=c(0.5,3.5))
matlines(x=xnew,y=pred,col=c(2,2,2),lty=c(1,2,2),type="l")

model<-SSModel(Nile~SSMtrend(1,Q=1469),H=15099)
pred<-predict(model,n.ahead=10,interval="prediction",level=0.9)

## End(Not run)
```

---

print.KFS *Print Ouput of Kalman Filter and Smoother*

Description

Print Ouput of Kalman Filter and Smoother

Usage

```
## S3 method for class 'KFS'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

x	output object from function KFS.
digits	minimum number of digits to be printed.
...	Ignored.

`print.SSModel` *Print SSModel Object*

Description

Print SSModel Object

Usage

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

Arguments

`x`	SSModel object
`...`	Ignored.

`residuals.KFS` *Extract Residuals of KFS output*

Description

Extract Residuals of KFS output

Usage

```
## S3 method for class 'KFS'
residuals(object, type = c("recursive", "deviance", "pearson",
  "response", "state"), ...)
```

Arguments

`object`	KFS object
`type`	Character string defining the type of residuals.
`...`	Ignored.

Details

For object of class KFS, several types of residuals can be computed:

- 'recursive': One-step ahead prediction residuals

  $$v_{t,i}),$$

  with residuals being undefined in diffuse phase. Only supported for fully Gaussian models.

- 'response': Data minus fitted values, $y - E(y)$.
- 'pearson':
$$(y_{t,i} - \theta_{t,i})/\sqrt{V(\mu)_{t,i}}, \quad i = 1, \ldots, p, t = 1, \ldots, n,$$
where $V(\mu_{t,i})$ is the variance function of the model.
- 'state': Residuals based on the smoothed disturbance terms η are defined as
$$\hat{\eta}_t, \quad t = 1, \ldots, n,$$
.
- 'deviance': Deviance residuals.

`rstandard.KFS` *Extract Standardized Residuals from KFS output*

Description

Extract Standardized Residuals from KFS output

Usage

```
## S3 method for class 'KFS'
rstandard(model, type = c("recursive", "deviance", "pearson",
  "state"), ...)
```

Arguments

`model`	KFS object
`type`	Type of residuals. See details.
`...`	Ignored.

Details

For object of class KFS, several types of standardized residuals can be computed:

- 'recursive': One-step ahead prediction residuals defined as
$$v_{t,i})/\sqrt{F_{i,t}},$$
with residuals being undefined in diffuse phase. Only supported for fully Gaussian models.
- 'pearson': Standardized Pearson residuals
$$(y_{t,i} - \theta_{t,i})/\sqrt{V(\mu)_{t,i}\phi_i\sqrt{1 - h_{t,i}}}, \quad i = 1, \ldots, p, t = 1, \ldots, n,$$
where $V(\mu_{t,i})$ is the variance function of the model, ϕ_i is the dispersion parameter and $h_{t,i}$ is the hat value. For gaussian models, these coincide with the smoothed ϵ disturbance residuals.
- 'state': Residuals based on the smoothed disturbance terms η are defined as
$$L_t^{-1}\hat{\eta}_t, \quad t = 1, \ldots, n,$$
where L_t is the lower triangular matrix from Cholesky decomposition of $V_{\eta,t}$.
- 'deviance': Deviance residuals.

`sexratio` *Number of males and females born in Finland from 1751 to 2011*

Description

A time series object containing the number of males and females born in Finland from 1751 to 2011.

Format

A time series object containing the number of males and females born in Finland from 1751 to 2011.

Source

Statistics Finland

`signal` *Extracting the Partial Signal Of a State Space Model*

Description

Function `signal` returns the signal of a state space model using only subset of states.

Usage

```
signal(object, states = "all", filtered = FALSE)
```

Arguments

`object`	Object of class `KFS`.
`states`	Which states are combined? Either a numeric vector containing the indices of the corresponding states, or a character vector defining the types of the corresponding states. Possible choices are “all”, “arima”, “custom”, “cycle”, “seasonal”, “trend”, or “regression”. These can be combined. Default is “all”.
`filtered`	If TRUE, filtered signal is used. Otherwise smoothed signal is used.

Value

`signal`	Time series object of filtered signal $Z_t a_t$ or smoothed signal $Z_t \hat{\alpha}_t$ using only the defined states.
`variance`	Cov($Z_t a_t$) or Cov($Z_t \hat{\alpha}_t$) using only the defined states. For the covariance matrices of the filtered signal, only the non-diffuse part P is used.

simulateSSM *Simulation of a gaussian State Space Model*

Description

Function `simulateSMM` simulates states, signals, disturbances or missing observations of the gaussian state space model.

Usage

```
simulateSSM(object, type = c("states", "signals", "disturbances",
  "observations", "epsilon", "eta"), filtered = FALSE, nsim = 1,
  antithetics = FALSE, conditional = TRUE)
```

Arguments

`object`	gaussian state space object.
`type`	What to simulate.
`filtered`	Simulate from $p(\alpha_t \mid y_{t-1}, ..., y_1)$ instead of $p(\alpha \mid y)$.
`nsim`	Number of independent samples. Default is 1.
`antithetics`	Use antithetic variables in simulation. Default is FALSE.
`conditional`	Simulations are conditional to data. If FALSE, the initial state α_1 is set to $\hat{\alpha}_1$ computed by `KFS`, and all the observations are removed from the model. Default is TRUE.

Details

Simulation smoother algorithm is based to article by J. Durbin and S.J. Koopman (2002).

Function can use two antithetic variables, one for location and other for scale, so output contains four blocks of simulated values which correlate which each other (ith block correlates negatively with (i+1)th block, and positively with (i+2)th block etc.).

Value

An n x k x nsim array containing the simulated series, where k is number of observations, signals, states or disturbances.

References

Durbin J. and Koopman, S.J. (2002). A simple and efficient simulation smoother for state space time series analysis, Biometrika, Volume 89, Issue 3

SSMarima *Create a State Space Model Object of Class SSModel*

Description

Function SSModel creates a state space object object of class SSModel which can be used as an input object for various functions of KFAS package.

Usage

```
SSMarima(ar = NULL, ma = NULL, d = 0, Q, stationary = TRUE, index, n,
  ynames)

SSMcustom(Z, T, R, Q, a1, P1, P1inf, index, n)

SSMcycle(period, type, Q, index, a1, P1, P1inf, n, ynames)

SSModel(formula, data, H, u, distribution, tol = .Machine$double.eps^0.5)

SSMregression(rformula, data, type, Q, index, R, a1, P1, P1inf, n, ynames)

SSMseasonal(period, sea.type = c("dummy", "trigonometric"), type, Q, index,
  a1, P1, P1inf, n, ynames)

SSMtrend(degree = 1, type, Q, index, a1, P1, P1inf, n, ynames)
```

Arguments

formula	an object of class formula containing the symbolic description of the model. The intercept term can be removed with -1 as in lm. In case of trend or differenced arima component intercept is removed automatically. Note that in order to be compatible with nonstationary elements, first level of each factor is always added to intercept, so if intercept is removed via -1, one level will be missing. See details and examples in KFAS for special functions used in model construction.
data	an optional data frame, list or environment containing the variables in the model.
H	covariance matrix or array of disturbance terms ϵ_t of observation equation. Omitted in case of non-gaussian distributions. Augment the state vector if you want to add additional noise.
u	additional parameters for non-gaussian models. See details in KFAS.
distribution	a vector of distributions of the observations. Default is rep('gaussian',p).
tol	a tolerance parameter for a diffuse phase. Smallest value of Finf not counted for zero. Defaults to .Machine$double.eps^0.5. If smoothing gives negative variances for smoothed states, try adjusting this.
index	a vector indicating for which series the corresponding components are constructed.

`type`	for cycle, seasonal, trend and regression components, character string defining if `'distinct'` or `'common'` states are used for different series.
`Q`	for arima, cycle and seasonal component, a $p \times p$ covariance matrix of the disturbances (or in the time varying case $p \times p \times n$ array), where where `p=length(index)`. For trend component, list of length `degree` containing the $p \times p$ or $p \times p \times n$ covariance matrices. For a custom component, arbitrary covariance matrix or array of disturbance terms η_t
`a1`	optional $m \times 1$ matrix giving the expected value of the initial state vector α_1.
`P1`	optional $m \times m$ matrix giving the covariance matrix of α_1. In the diffuse case the non-diffuse part of P_1.
`P1inf`	optional $m \times m$ matrix giving the diffuse part of P_1. Diagonal matrix with ones on diagonal elements which correspond to the unknown initial states.
`R`	for a custom and regression components, optional $m \times k$ system matrix or array of transition equation.
`ar`	for arima component, a numeric vector containing the autoregressive coeffients.
`ma`	for arima component, a numericvector containing the moving average coeffients.
`d`	for arima component, a degree of differencing.
`stationary`	for arima component, logical value indicating whether a stationarity of the arima part is assumed. Defaults to TRUE.
`Z`	for a custom component, system matrix or array of observation equation.
`T`	for a custom component, system matrix or array of transition equation.
`period`	for a cycle and seasonal components, the length of the cycle/seasonal pattern.
`sea.type`	for seasonal component, character string defining whether to use `'dummy'` or `'trigonometric'` form of the seasonal component.
`degree`	for trend component, integer defining the degree of the polynomial trend. 1 corresponds to local level, 2 for local linear trend and so forth.
`rformula`	for regression component, right hand side formula or list of of such formulas defining the custom regression part.
`n`	length of the series, only used internally for dimensionality check.
`ynames`	names of the times series, only used internally.

Details

Formula of the model can contain the usual regression part and additional functions defining different types of components of the model, named as `SSMarima`, `SSMcustom`, `SSMcycle`, `SSMregression`, `SSMseasonal` and `SSMtrend`.

Value

object of class `SSModel`, which is a list with the following components:

`y`	A n x p matrix containing the observations.
`Z`	A p x m x 1 or p x m x n array corresponding to the system matrix of observation equation.

H	A p x p x 1 or p x p x n array corresponding to the covariance matrix of observational disturbances epsilon.
T	A m x m x 1 or m x m x n array corresponding to the first system matrix of state equation.
R	A m x k x 1 or m x k x n array corresponding to the second system matrix of state equation.
Q	A k x k x 1 or k x k x n array corresponding to the covariance matrix of state disturbances eta
a1	A m x 1 matrix containing the expected values of the initial states.
P1	A m x m matrix containing the covariance matrix of the nondiffuse part of the initial state vector.
P1inf	A m x m matrix containing the covariance matrix of the diffuse part of the initial state vector.
u	A n x p matrix of an additional parameters in case of non-Gaussian model.
distribution	A vector of length p giving the distributions of the observations.
tol	A tolerance parameter for the diffuse phase.
call	Original call to the function.

In addition, object of class SSModel contains following attributes:

names	Names of the list components.
p, m, k, n	Integer valued scalars defining the dimensions of the model components.
state_types	Types of the states in the model.

See Also

KFAS for examples.

Examples

```
## Not run:
examplemodel<-SSModel(cbind(y1,y2,y3)  ~ x1+x2
+ SSMregression(~-1+x3+x4,data=dataset,type='common',index=c(1,3),Q=diag(c(0.05,0.1)))
+ SSMtrend(degree=1,index=1,Q=list(matrix(0.2)))
+ SSMtrend(degree=2,index=2:3,Q=list(matrix(c(0.2,0.1,0.1,0.2),2,2),diag(0.07,2)))
+ SSMcycle(period=25,Q=matrix(c(0.3,0.2,0.1,0.2,0.4,0.05,0.1,0.05,0.1),3,3))
, data=dataset, H=matrix(c(1,0.7,0.7,0.7,1,0.7,0.7,0.7,1),3,3))

## End(Not run)
```

`transformSSM` *Transform the SSModel object with multivariate observations*

Description

Function transform.SSModel transforms original model by LDL decomposition or state vector augmentation,

Usage

```
transformSSM(object, type = c("ldl", "augment"))
```

Arguments

`object`	State space model object from function SSModel.
`type`	Option `'ldl'` performs LDL decomposition for covariance matrix H_t, and multiplies the observation equation with the L_t^{-1}, so $\epsilon_t^* \sim N(0, D_t)$. Option `'augment'` adds ϵ_t to the state vector, when Q_t becomes block diagonal with blocks Q_t and H_t.

Details

As all the functions in KFAS use univariate approach, H_t, a covariance matrix of an observation equation needs to be either diagonal or zero matrix. Function transformSSM performs either the LDL decomposition of the covariance matrix of the observation equation, or augments the state vector with the disturbances of the observation equation.

In case of a LDL decomposition, the new H_t contains the diagonal part of the decomposition, whereas observations y_t and system matrices Z_t are multiplied with the inverse of L_t.

Value

`model`	Transformed model.

`[<-.SSModel` *Extract or Replace Parts of a State Space Model*

Description

S3 methods for extracting or replacing parts of objects of class `SSModel`. These methods ensure that dimensions of system matrices are not altered. `[` and `subset` and corresponding replacement methods are identical methods with different method names.

Usage

```
## S3 replacement method for class 'SSModel'
x[element, states, etas, series, times, ...] <- value

## S3 method for class 'SSModel'
x[element, states, etas, series, times, ...]

## S3 replacement method for class 'SSModel'
subset(x, element, states, etas, series, times, ...) <- value

subset(x, ...) <- value

## S3 method for class 'SSModel'
subset(x, element, states, etas, series, times, ...)
```

Arguments

x	Object of class SSModel.
element	Which element is chosen. Possible choices are 'y','Z','H','T','R','Q','a1','P1','P1inf', and 'u'.
states	Which states are chosen. Either a numeric vector containing the indices of the corresponding states, or a character vector defining the types of the corresponding states. Possible choices are "all", "arima", "custom", "cycle", "seasonal", "trend", or "regression". These can be combined. Default is "all".
etas	Which disturbances eta are chosen. Used for elements "R" and "Q".Either a numeric vector containing the indices of the corresponding etas, or a character vector defining the types of the corresponding etas. Possible choices are "all", "arima", "custom", "cycle", "seasonal", "trend", or "regression". These can be combined.
series	Numeric. Which series are chosen. Used for elements "y", "Z", and "u".
times	Numeric. Which time points are chosen.
value	A value to be assigned to x.
...	ignored.

Value

A selected subset of the chosen element or a value.

Examples

```
set.seed(1)
model<-SSModel(rnorm(10)~1)
model["H"]
model["H"]<-10
# H is still an array:
model["H"]
logLik(model)
```

```
model$H<-1
# model["H"] throws an error as H is now scalar:
model$H
logLik(model,check.model=TRUE) #with check.model=FALSE (default) R crashes!
```

Index