Package 'KFKSDS'

January 28, 2015

Version 1.6

Date 2015-01-28

Title Kalman Filter, Smoother and Disturbance Smoother

Description Naive implementation of the Kalman filter, smoother and disturbance smoother for state space models.

Author Javier López-de-Lacalle

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

Depends R (>= 3.0.0), stats

Suggests dlm, dse, FKF, KFAS, numDeriv, stsm

NeedsCompilation yes

Encoding UTF-8

License GPL-2

URL http://jalobe.com

Repository CRAN

Date/Publication 2015-01-28 20:51:01

R topics documented:

KFKSDS-package 2
DS 3
KF 5
KF-interfaces 9
KFKSDS 11
KS 13
predict.stsmSS 15

Index **18**

KFKSDS-package *Kalman Filter, Smoother and Disturbance Smoother*

Description

This package provides an implementation of the Kalman filter, smoother and disturbance smoother for state space models. An interface to run implementations of the filter that are available in other packages is also provided.

Details

There are several other packages in R that perform Kalman filtering and smoothing. A review can be found in Tusell (2011). Each one has its own strengths that make them best suited to a particular context. The package **KFKSDS** was developed as a tool to conduct the work described in López-de-Lacalle (2013a, 2013b). Within this framework, the package is useful as a development tool as well as for debugging and testing purposes.

The implementation is naive in that it is a direct transcription of the equations of the filter and the smoother that can be found in many textbooks. A square root filter is not considered in order to deal with potential numerical problems that may arise in the original equations.

The package includes a wrapper function that links to functions from the following packages: **dlm**, **dse**, **FKF**, **KFAS**

and **stats**. The original interface provided by each package is recommended as they sometimes provide further capabilities or options. Nevertheless, the wrapper function provided in this package is useful for debugging and testing since it allows running different implementations of the filter through a unified interface.

A useful utility of the package is that it computes the analytical derivatives of some of the elements of the filter and the smoother with respect to the parameters of the model. In particular, the necessary elements to compute the analytical derivatives of the time domain log-likelihood function are returned. This is especially useful when it comes to maximizing the likelihood function.

In some models, the Kalman filter and the smoother are expected to converge to a steady state. Some optional parameters can be defined in order to assess at each iteration of the filter whether a steady state has been reached. When the steady state is reached, the values from the last iteration are used in the remaining iterations of the filter and smoother. Thus, the number of matrix operations can be reduced substantially.

Some parts of the code are implemented in the C language where the matrix operations are handled through the GNU Scientific Library.

The package **stsm** is a useful complement to easily create an object containing the matrices of the state space representation for the structural models defined in that package.

References

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

Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M. and Rossi, F. (2009). *GNU Scientific Library Reference Manual*. Network Theory Ltd.

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

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

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

Tusell. F. (2011). 'Kalman Filtering in R.' *Journal of Statistical Software*, **39**(2). http://www.jstatsoft.org/v39/i02/.

Author(s)

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

DS *Disturbance Smoother for State Space Models*

Description

These functions run the disturbance smoother upon the output from the Kalman filter and smoother.

Usage

```
DS(y, ss, kf, ks)
DS.deriv(ss, ksd)
```

Arguments

y	a numeric time series or vector.
ss	a list containing the matrices of the state space model.
kf	a list containing the output returned by the function KF.
ks	a list containing the output returned by the function KS.
ksd	a list containing the output returned by the function KS.deriv.

Details

See the details section and the section 'state space representation' in KF.

Value

DS returns a list containing the following elements:

epshat	smoothed estimate of the disturbance term in the observation equation.
vareps	error variance of epshat.
etahat	smoothed estimate of the disturbance term(s) in the state equation.
vareta	error variance of etahat.

DS.deriv returns a list containing the derivatives of the elements above named respectively depshat, dvareps, detahat and dvareta. The derivatives are summed over all the observations.

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

KF, KS; char2numeric in package **stsm**.

Examples

```
# local level plus seasonal model with arbitrary parameter values
# for the 'JohnsonJohnson' time series
m <- stsm::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
ss <- stsm::char2numeric(m)

kf <- KF(m@y, ss)
ks <- KS(m@y, ss, kf)
ds <- DS(m@y, ss, kf, ks)
acf(ds$epshat, main = "ACF of smoothed disturbance")

kfd <- KF.deriv(m@y, ss)
ksd <- KS.deriv(m@y, ss, kfd)
dsd <- DS.deriv(ss, ksd)

# compare analytical and numerical derivatives
fcn <- function(x, model, type, i = 1)
{
  m <- stsm::set.pars(model, x)
  ss <- stsm::char2numeric(m)
  kf <- KF(m@y, ss)
  ks <- KS(m@y, ss, kf)
  ds <- DS(m@y, ss, kf, ks)

  switch(type,
    "epshat" = sum(ds$epshat),
    "vareps" = sum(ds$vareps))
}

d <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "epshat")
all.equal(d, dsd$depshat)

d <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "vareps")
all.equal(d, dsd$dvareps)
```

KF *Kalman Filter for State Space Models*

Description

These functions run the iterative equations of the Kalman filter for a state space model.

Usage

```
KF(y, ss, convergence = c(0.001, length(y)), t0 = 1)
KF.C(y, ss, convergence = c(0.001, length(y)), t0 = 1)
KF.deriv(y, ss, xreg = NULL, convergence = c(0.001, length(y)), t0 = 1)
KF.deriv.C(y, ss, xreg = NULL, convergence = c(0.001, length(y)),
  t0 = 1, return.all = FALSE)
```

Arguments

y	a numeric time series or vector.
ss	a list containing the matrices of the state space model.
xreg	optional matrix or list of external regressors. See details.
convergence	a numeric vector of length two to control and determine the convergence of the filter. See details below.
t0	a numeric indicating the index of the first observation at which the contributions to the likelihood are addep up.
return.all	logical. If TRUE, extended output containing elements to be used by KS.deriv is returned.

Details

The implementation is a direct transcription of the iterative equations of the filter that are summarized below. Details can be found in the references given below and in many other textbooks. The source code follows the notation used in Durbin and Koopman (2001).

The elements in the argument ss must be named in accordance with the notation given below for the state space representation. For those models defined in the package **stsm**, a convenient way to create the argument ss is by means of the function char2numeric.

The contributions to the likelihood function of the first observations may be omitted by choosing a value of t0 greater than one.

The functions with '.deriv' in the name compute the derivatives of some of the elements involved in the filter with respect to the parameters of the model.

The functions KF and KF.deriv are fully implemented in R while KF.deriv.C calls to compiled C code.

Using KF.deriv.C with return.all = FALSE returns minimal output with the elements necessary to compute the derivative of the log-likelihood function. Using return.all = TRUE further elements to be used in KS.deriv are returned.

Missing observations are allowed. If a missing value is observed after the filter has converged then all operations of the filter are run instead of using steady state values until convergence is detected again.

Parameters to control the convergence of the filter. In some models, the Kalman filter may converge to a steady state. Finding the explicit expression of the steady state values can be cumbersome in some models. Alternatively, at each iteration of the filter it can be checked whether a steady state has been reached. For it, some control parameters can be defined in the argument `convergence`. It is considered that convergence was reached when the following is observed: the change in the variance of the prediction error over the last `convergence[2]` consecutive iterations of the filter is below the tolerance value `convergence[1]`. When the steady state is reached, the values from the last iteration are used in the remaining iteration of the filter. Thus, the number of matrix operations can be reduced substantially as pointed in Harvey (1989) Sec. 3.3.4.

External regressors. A matrix of external regressors can be passed in argument `xreg`. If `xreg` is a matrix then it is assumed that the time series passed in argument `y` has been already adjusted for the effect of these regressors, that is, $y_t^{adj} = y_t - X\gamma$. If `y` is the observed series, then `xreg` should be a list containing the following elements: `xreg`, the matrix of regressors; and `coefs`, the vector of coefficients, γ, related to the regressors. The coefficients must be placed in `coefs` in the same order as the corresponding vectors are arranged by columns in `xreg`.

The number of rows of the matrix of regressors must be equal to the length of the series `y`.

Column names are necessary for `KF.deriv` and are optional for `KF.deriv.C`.

Value

A list containing the following elements:

`v`	prediction error.
`f`	variance of the prediction error.
`K`	Kalman gain.
`L`	auxiliar matrices to be used by the smoother.
`a.pred`	one step ahead prediction of the state vector.
`P.pred`	covariance matrix of `a.pred`.
`a.upd`	update of `a.pred` after the observation at time t that is predicted in `a.pred` enters in the recursive filter.
`P.upd`	update of `P.pred`.
`convit`	the iteration at which the filter converged. If convergence was not observed it is set to `NULL`.
`mll`	value of the negative of the log-likelihood function.

The function `KF.C` is a simplified and faster version that records and returns only the value of negative of the log-likelihood function. It is suited to be passed as argument to `optim` in the **stats** package.

The functions that evaluate the derivatives include in the output the derivatem terms: `da.pred`, `dP.pred`, `da.upd`, `dP.upd`, `dv`, `df`, `dvof` (derivative of quotient `v` over `f`), `dK` and `dL`.

`KF.deriv.C` does not return `a.upd` and `P.upd` and their derivative terms `da.upd` and `dP.upd`. If `return.all = TRUE`, this function returns: `dvof`, `dL`, `da.pred`, `dP.pred`, which are the derivative

terms necessary to evaluate the gradient of the log-likelihood function. By default they are not returned, `return.all = FALSE`. They are in any case computed, the operations that are omitted in the latter case is the arrangement of the output from the call to compiled C code into matrices of the proper dimension containing the data in the right order.

Derivatives of the likelihood function are implemented in package **stsm**. Although the Kalman filter provides information to evaluate the likelihood function, it is not its primary objective. That's why the derivatives of the likelihood are currently part of the package **stsm**, which is specific to likelihood methods in structural time series models.

State space representation

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 m is the dimension of the state vector a and 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 $v[t]$.

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

`char2numeric` in package **stsm**.

Examples

```
# local level plus seasonal model with arbitrary parameter values
# for the 'JohnsonJohnson' time series
m <- stsm::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
ss <- stsm::char2numeric(m)

# run the Kalman filter
kf <- KF(m@y, ss)
plot(kf$a.upd[,1:2], main = "filtered state vector")

# 'KF.C' is a faster version that returns only the
# value of the negative of the likelihood function
kfc <- KF.C(m@y, ss)
all.equal(kf$mll, kfc)

# compute also derivative terms used below
kfd <- KF.deriv(m@y, ss)
all.equal(kfc, kfd$mll)
kfdc <- KF.deriv.C(m@y, ss, return.all = TRUE)
all.equal(kf$mll, kfdc$mll)

# as expected the versions that use compiled C code
# are faster that the versions written fully in R
# e.g. not including derivatives
## Not run:
system.time(for(i in seq_len(10)) kf <- KF(m@y, ss))
system.time(for(i in seq_len(10)) kfc <- KF.C(m@y, ss))
# e.g. including derivatives
system.time(for(i in seq_len(10)) kfd <- KF.deriv(m@y, ss))
system.time(for(i in seq_len(10)) kfdc <- KF.deriv.C(m@y, ss, return.all = TRUE))

## End(Not run)

# compare analytical and numerical derivatives
# they give same results up to a tolerance error
fcn <- function(x, model, type = c("v", "f"))
{
  m <- stsm::set.pars(model, x)
  ss <- stsm::char2numeric(m)
  kf <- KF(m@y, ss)
  switch(type, "v" = sum(kf$v), "f" = sum(kf$f))
}

dv <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "v")
all.equal(dv, colSums(kfd$dv), check.attributes = FALSE)
all.equal(dv, colSums(kfdc$dv), check.attributes = FALSE)
df <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "f")
all.equal(df, colSums(kfd$df), check.attributes = FALSE)
all.equal(df, colSums(kfdc$df), check.attributes = FALSE)

# compare timings in version written in R with numDeriv::grad
```

```
# no calls to compiled C code in either case
# looking at these timings, using analytical derivatives is
# expected to be useful in optimization algorithms
## Not run:
system.time(for (i in seq_len(10))
  numdv <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "v"))
system.time(for(i in seq_len(10)) kfdv <- colSums(KF.deriv(m@y, ss)$dv))

## End(Not run)

# compare timings when convergence is not checked with the case
# when steady state values are used after convergence is observed
# computation time is reduced substantially
## Not run:
n <- length(m@y)
system.time(for(i in seq_len(20)) a <- KF.deriv(m@y, ss, convergence = c(0.001, n)))
system.time(for(i in seq_len(20)) b <- KF.deriv(m@y, ss, convergence = c(0.001, 10)))
# the results are the same up to a tolerance error
all.equal(colSums(a$dv), colSums(b$dv))

## End(Not run)
```

KF-interfaces *Kalman Filter*

Description

This function provides a unified interface for implementations of the Kalman filter available in different packages.

Usage

```
KalmanFilter(y, ss,
  KF.version = c("KFKSDS", "StructTS", "KFAS", "FKF", "dlm", "dse"),
  KF.args = list(), check.args = TRUE, debug = FALSE)
make.KF.args(ss, KF.version, KF.args = list())
```

Arguments

y	a numeric time series or vector.
ss	a list containing the matrices of the state space model.
KF.version	a character string indicating the implementation to be used.
KF.args	a list containing optional arguments to be passed to the function that runs the Kalman filter.
check.args	logical. If TRUE, it is checked that the elements of KF.args are valid for the selected interface KF.version.
debug	logical. Currently ignored.

Details

For some purposes such as testing, debugging or development of extensions, this function provides a useful unified interface for different implementations of the Kalman filter.

In a production environment, the use of the original interfaces provided by each package is recommended since they sometimes provide further options or may incorporate further capabilities in the latest updates.

The elements in the argument `ss` must be named in accordance with the notation given in `KF`. The function `char2numeric` in package **stsm** is a convenient way to create the argument `ss` for those models already defined in that package.

If `KF.args` is empty, default values are defined depending on the interface selected in `KF.version`. The function `make.KF.args` set default values for those arguments that are explicitly defined in `KF.args`. It also checks that the arguments passed through `KF.args` are correct and a warning is given if any of them does not apply to the selected interface `KF.version`.

Argument `KF.version`: the option `StructTS` applies the Kalman filter as in the function `StructTS` of the **stats** package. The remaining possible values for this argument are the names of the package that contains the Kalman filter interface.

Notes: (1) The package **sppir** is no longer maintained on CRAN and is not currently available here as an option. For old versions see sspir.

(2) `KF.version="dse"` requires manually loading the package **dse**.

Value

A list containing the output returned by each interface and the value of the negative of the log-likelihood function in the element `mloglik`.

References

Dethlefsen, C., Lundbye-Christensen, S. and Christensen A. L. (2012) R package version 0.2.10. **sspir***: State Space Models in* R, http://CRAN.R-project.org/package=sspir.

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

Gilbert, P. D. (2013) R package version 2013.3-2. **dse** *Brief User's Guide: Dynamic Systems Estimation*, http://CRAN.R-project.org/package=dse.

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

Helske, J. (2012). R package version 0.9.11. **KFAS***: Kalman Filter and Smoother for Exponential family state space models.*, http://CRAN.R-project.org/package=KFAS.

Luethi, D., Erb, P. and Otziger, S. (2012) R package version 0.1.2. **FKF***: Fast Kalman Filter*, http://CRAN.R-project.org/package=FKF.

Petris, G. (2013) R package version 1.1-3. **dlm** *An R Package for Dynamic Linear Models*, http://CRAN.R-project.org/package=dlm.

See Also

`KF`; `char2numeric` in package **stsm**.

Examples

```
# state space representation of a structural time series model
# with arbitrary parameter values
require("stsm")
m <- stsm::stsm.model(model = "BSM", y = AirPassengers, transP = "StructTS",
  pars = c("var1" = 30, "var2" = 15, "var3" = 1, "var4" = 12))
ss <- stsm::char2numeric(m)
# value of the likelihood using different interfaces and options
KalmanFilter(y = m@y, ss = ss, KF.version = "KFKSDS", KF.args = list(P0cov = FALSE))$mloglik
KalmanFilter(y = m@y, ss = ss, KF.version = "KFKSDS", KF.args = list(P0cov = TRUE))$mloglik
# 'StructTS' does not include some constants
KalmanFilter(y = m@y, ss = ss, KF.version = "StructTS")$mloglik
```

KFKSDS *Kalman Filter, Smoother and Disturbance Smoother*

Description

Kalman filter, smoother and disturbance smoother.

Usage

```
KSDS(y, ss, kf)
KSDS.deriv(y, ss, kf)
KFKSDS.deriv.C(y, ss)
KFKSDS.deriv.steady.C(y, ss, convergence = c(0.001, 10, 1.2))
```

Arguments

y	a numeric time series or vector.
ss	a list containing the matrices of the state space model.
kf	a list containing the output returned by the function KF.
convergence	a numeric vector of length three to control and determine the convergence of the filter and smoother. See details below.

Details

See the details section and the section 'state space representation' in KF.

The iteration at which the the Kalman smoother converges is determined as the iteration where the Kalman filter converged multiplied by the factor convergence[3]. It should be equal or greater than unity. It can be omitted by setting it equal to one.

These interfaces are fully implemented in R.

In these functions, the Kalman smoother and the disturbance smoother are run in a single loop instead of running two sepate loops over the same observations.

These functions return a relatively extended output than can be used, for example, to implement an expectation-maximization algorithm.

Value

The function KSDS returns a list containing the following elements:

ahat	smoothed state disturbance.
varhat	covariance matrix of ahat.
r	weighted sum of innovations used to obtain ahat.
N	intermediate matrix used to obtain varahat
epshat	smoothed estimate of the disturbance term in the observation equation.
vareps	error variance of epshat.
etahat	smoothed estimate of the disturbance term(s) in the state equation.
vareta	error variance of etahat.

The function KSDS-deriv returns also dahat, dvarahat, dr, dN which are the derivatives referred to the elements defined above.

The functions KFKSDS.deriv.C and KFKSDS.deriv.steady.C return a list containing the elements already defined above: epshat, vareps, etahat, vareta, r, N, dr, dN, dahat and dvareps.

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

KF, KS; char2numeric in package **stsm**.

Examples

```
# See tests comparing the analytical derivatives returned by
# these functions and the numerical derivatives evaluated with
# function 'numDeriv::grad' in the folder 'KFKSDS/inst/tests'
# of the source package

# local level plus seasonal model with arbitrary parameter values
# for the 'JohnsonJohnson' time series
m <- stsm::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
ss <- stsm::char2numeric(m)
kfd <- KF.deriv(m@y, ss)
ksds <- KSDS(m@y, ss, kfd)
da <- KSDS.deriv(m@y, ss, kfd)
db <- KFKSDS.deriv.C(m@y, ss)
# the ouput is the same but is returned with a different arrangement
dim(da$dahat)
dim(db$dahat)
for (i in seq_along(m@y))
```

```
  stopifnot(all.equal(da$dahat[i,,], db$dahat[,,i]))
dim(da$dN)
dim(db$dN)
for (i in seq_along(m@y))
  for (j in seq_len(3))
    stopifnot(all.equal(diag(da$dN[i,,,j]), db$dN[,j,i], check.attributes = FALSE))
```

KS *Kalman Smoother for State Space Models*

Description

These functions run the iterative equations of the Kalman smoother for a state space model upon the output from the Kalman filter.

Usage

```
KS(y, ss, kf)
KS.deriv(y, ss, kf)
```

Arguments

y	a numeric time series or vector.
ss	a list containing the matrices of the state space model.
kf	a list containing the output returned by the Kalman filter KF.

Details

See the details section and the section 'state space representation' in KF.

Missing observations are allowed.

The input kf passed to KS.deriv must contain the derivative terms related to the filter that are returned by KF.deriv or KF.deriv.C.

When the Kalman filter was found to convergence at some iteration, i.e., kf$convit is not null, these functions use steady state values for N and varahat in the intermediate iterations of the smoother. For example, if the filter converged at iteration 15 in a series of length n, the equations of the smoother are run for the first iterations from observation n to $n - 15$; then the steady state values are used until there are 15 iterations remaining. In the last iterations, from observation 15 to 1 the equations of the smoother are evaluated again.

In practice, if the disturbance smoother is to be run as well, using the functions described in KFKSDS will be slightly more efficient.

Value

A list containing the following elements:

ahat	smoothed state vector.
varhat	covariance matrix of ahat.
r	weighted sum of innovations used to obtain ahat.
N	intermediate matrix used to obtain varahat

The function KS.deriv returns also the derivatives referred to each of the elements defined above, named respectively dahat, dvarahat, dr and dN.

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

KF, KSDS; char2numeric in package **stsm**.

Examples

```
# local level plus seasonal model with arbitrary parameter values
# for the 'JohnsonJohnson' time series
m <- stsm::stsm.model(model = "llm+seas", y = JohnsonJohnson,
  pars = c("var1" = 2, "var2" = 15, "var3" = 30))
ss <- stsm::char2numeric(m)

kf <- KF(m@y, ss)
ks <- KS(m@y, ss, kf)

plot(ks$ahat[,1:2], main = "smoothed state vector")

kfd <- KF.deriv(m@y, ss)
ksd <- KS.deriv(m@y, ss, kfd)
all.equal(ks$ahat, ksd$ahat)

# extended output is required if 'KF.deriv.C' is used to obtain
# the necessary elements from the filter, set return.all = TRUE
kfdc <- KF.deriv.C(m@y, ss, return.all = TRUE)
ksd <- KS.deriv(m@y, ss, kfdc)
all.equal(ks$ahat, ksd$ahat)

# compare analytical and numerical derivatives
# yield same results up to a tolerance error
fcn <- function(x, model, type, i)
{
  m <- stsm::set.pars(model, x)
```

```
  ss <- stsm::char2numeric(m)
  kf <- KF(m@y, ss)
  ks <- KS(m@y, ss, kf)
  switch(type, "ahat" = sum(ks$ahat[,i]), "r" = sum(ks$r[,i]))
}

dahat <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "ahat", i = 1)
all.equal(dahat, colSums(ksd$dahat[,1,]))
dahat <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "ahat", i = 2)
all.equal(dahat, colSums(ksd$dahat[,2,]))
dahat <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "ahat", i = 3)
all.equal(dahat, colSums(ksd$dahat[,3,]))
dr <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "r", i = 1)
all.equal(dr, colSums(ksd$dr[,1,]), check.attributes = FALSE)
dr <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "r", i = 2)
all.equal(dr, colSums(ksd$dr[,2,]), check.attributes = FALSE)
dr <- numDeriv::grad(func = fcn, x = m@pars, model = m, type = "r", i = 3)
all.equal(dr, colSums(ksd$dr[,3,]), check.attributes = FALSE)
```

`predict.stsmSS` *Kalman Filter for State Space Models*

Description

These functions run the iterative equations of the Kalman filter for a state space model.

Usage

```
## S3 method for class 'stsmSS'
predict(object, y, n.ahead = 12L, ...)
```

Arguments

`object`	a list containing the matrices of the state space model.
`y`	a numeric time series.
`n.ahead`	a numeric. The number of steps ahead to predict.
`...`	further arguments. Currently omitted.

Details

This function computes the same values as the function predict.StructTS from the **stats** package but the predictions of the components are also returned.

Value

A list containing the following elements: itempreda time series containing `n.ahead` predictions. itemsea time series containing the standard errors of `pred`. itemaa univariate or multivariate time series object containing `n.ahead` predictions for the state vector. itemPa univariate or multivariate time series object containing the square of the standard errors of `a`.

References

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

Examples

```
## local level model
## Nile time series
require("stsm")
y <- Nile
m <- stsm::stsm.model(model = "local-level", y = y, transPars = "StructTS")
fit <- StructTS(y, "level")
m <- stsm::set.pars(m, as.vector(fit$coef[c(2,1)]) * 100 / var(y))
ss <- stsm::char2numeric(m, P0cov = TRUE)
res <- predict(ss, y, 5)

# display forecasts and confidence intervals
plot(cbind(y, res$pred), type = "n", plot.type = "single")
lines(y)
lines(res$pred, col = "blue")
lines(res$pred + 2 * res$se, col = "red", lty = 2)
lines(res$pred - 2 * res$se, col = "red", lty = 2)

# for the whole series, the above is the same as "predict.StructTS"
all.equal(res$pred, predict(fit, 5)$pred)
all.equal(res$se, predict(fit, 5)$se)

## basic Structural model
## AirPassengers time series (in logarithms)
y <- log(AirPassengers)
m <- stsm::stsm.model(model = "BSM", y = y, transPars = "StructTS")
fit <- StructTS(y, "BSM")
m <- stsm::set.pars(m, as.vector(fit$coef[c(4,1:3)]) * 100 / var(y))
ss <- stsm::char2numeric(m, P0cov = TRUE)
res <- predict(ss, y, 12)

all.equal(res$pred, predict(fit, 12)$pred)
all.equal(res$se, predict(fit, 12)$se)

# forecasts and confidence intervals for the series
# scaled back to original scale
expy <- exp(y)
plot(cbind(expy, exp(res$pred + 2 * res$se)), type = "n", plot.type = "single")
lines(expy)
lines(exp(res$pred), col = "blue")
lines(exp(res$pred + 2 * res$se), col = "red", lty = 2)
lines(exp(res$pred - 2 * res$se), col = "red", lty = 2)

# forecasts for the trend component
# the aproach in StructTS() seems to seasonal fluctuations in the trend
# see the "stsm" package for a more flexible interface for maximum likelihood
# procedures to fit a structural time series model
```

```
trend <- exp(fitted(fit)[,1])
plot(cbind(trend, AirPassengers), type = "n", plot.type = "single")
lines(AirPassengers, col = "gray")
lines(trend)
lines(exp(res$a[,1]), col = "blue")
lines(exp(res$a[,1] + 2 * sqrt(res$P[,1])), col = "red", lty = 2)
lines(exp(res$a[,1] - 2 * sqrt(res$P[,1])), col = "red", lty = 2)

# forecasts for the seasonal component
seas <- exp(fitted(fit)[,3])
plot(cbind(seas, exp(res$a[,3]) + 2 * sqrt(res$P[,3])),
  type = "n", plot.type = "single")
lines(seas)
lines(exp(res$a[,3]), col = "blue")
lines(exp(res$a[,3] + 2 * sqrt(res$P[,3])), col = "red", lty = 2)
lines(exp(res$a[,3] - 2 * sqrt(res$P[,3])), col = "red", lty = 2)
```

Index

∗Topic **package, ts**
KFKSDS-package, 2
∗Topic **ts, model**
DS, 3
KF, 5
KF-interfaces, 9
KFKSDS, 11
KS, 13
predict.stsmSS, 15

char2numeric, *4, 5*, *7*, *10*, *12*, *14*

DS, 3

KalmanFilter *(*KF-interfaces*)*, 9
KF, *3, 4*, 5, *10–14*
KF-interfaces, 9
KF.deriv, *13*
KF.deriv.C, *13*
KFKSDS, 11, *13*
KFKSDS-package, 2
KS, *4*, *12*, 13
KS.deriv, *5*
KSDS, *14*
KSDS *(*KFKSDS*)*, 11

make.KF.args *(*KF-interfaces*)*, 9

optim, *6*

predict.StructTS, *15*
predict.stsmSS, 15