Package 'FKF.SP'

March 21, 2022

Title Fast Kalman Filtering Through Sequential Processing

Version 0.2.0

Description Fast and flexible Kalman filtering implementation utilizing sequential processing, designed for efficient parameter estimation through maximum likelihood estimation. Sequential processing is a univariate treatment of a multivariate series of observations and can benefit from computational efficiency over traditional Kalman filtering when independence is assumed in the variance of the disturbances of the measurement equation. Sequential processing is described in the textbook of Durbin and Koopman (2001, ISBN:978-0-19-964117-8). 'FKF.SP' was built upon the existing 'FKF' package and is, in general, a faster Kalman filter.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.2

RdMacros mathjaxr,
Rdpack

Imports mathjaxr,
Rdpack,
curl

Suggests knitr,
rmarkdown,
stats,
FKF,
NFCP

VignetteBuilder knitr

URL https://github.com/TomAspinall/FKF.SP

BugReports https://github.com/TomAspinall/FKF.SP/issues

R topics documented:

fkf.SP *Fast Kalman Filtering using Sequential Processing.*

Description

The fkf.SP function performs fast and flexible Kalman filtering using sequential processing. It is designed for efficient parameter estimation through maximum likelihood estimation. fkf.SP wraps the C-function fkf_SP which relies upon the linear algebra subroutines of BLAS (Basic Linear Algebra Subprograms). Sequential processing (SP) is a univariate treatment of a multivariate series of observations that increases computational efficiency over traditional Kalman filtering in the general case. SP takes the additional assumption that the variance of disturbances in the measurement equation are independent. fkf.SP is based from the fkf function of the FKF package but is, in general, a faster Kalman filtering method. fkf and fkf.SP share identical arguments (except for the GGt argument, see **arguments**). fkf.SP is compatible with missing observations (i.e. NA's in argument yt).

Usage

```
fkf.SP(a0, P0, dt, ct, Tt, Zt, HHt, GGt, yt, verbose = FALSE)
```

Arguments

a0	A vector giving the initial value/estimation of the state variable
P0	A matrix giving the variance of a0
dt	A matrix giving the intercept of the transition equation
ct	A matrix giving the intercept of the measurement equation
Tt	An array giving factor of the transition equation
Zt	An array giving the factor of the measurement equation
HHt	An array giving the variance of the innovations of the transition equation
GGt	A vector giving the diagonal elements of the matrix for the variance of disturbances of the measurement equation. Covariance between disturbances is not supported under the sequential processing method.
yt	A matrix containing the observations. "NA"- values are allowed
verbose	A logical. When verbose = TRUE, A list object is output, which provides the filtered state variables and variances of the Kalman filter.

Details

Parameters:

The fkf.SP function builds upon the fkf function of the FKF package by adjusting the Kalman filtering algorithm to utilize sequential processing. Sequential processing can result in significant decreases in processing time over the traditional Kalman filter algorithm. The fkf.SP and fkf functions feature highly similar arguments for compatibility purposes; only argument GGt has changed from an array type object to a vector or matrix type object. The fkf.SP function takes the additional assumption over the fkf function that the variance of the disturbances of the measurement equation are independent; a requirement of SP (see below).

Parameters can either be constant or deterministic time-varying. Assume the number of discrete time observations is n i.e. $y = y_t$ where $t = 1, \cdots, n$. Let m be the dimension of the state variable

and d the dimension of the observations. Then, the parameters admit the following classes and dimensions:

`dt`	either a $m \times n$ (time-varying) or a $m \times 1$ (constant) matrix.
`Tt`	either a $m \times m \times n$ or a $m \times m \times 1$ array.
`HHt`	either a $m \times m \times n$ or a $m \times m \times 1$ array.
`ct`	either a $d \times n$ or a $d \times 1$ matrix.
`Zt`	either a $d \times m \times n$ or a $d \times m \times 1$ array.
`GGt`	either a $d \times n$ (time-varying) or a $d \times 1$ matrix.
`yt`	a $d \times n$ matrix.

State Space Form

The following notation follows that of Koopman *et al.* (1999). The Kalman filter is characterized by the transition and measurement equations:

$$\alpha_{t+1} = d_t + T_t \cdot \alpha_t + H_t \cdot \eta_t$$

$$y_t = c_t + Z_t \cdot \alpha_t + G_t \cdot \epsilon_t$$

where η_t and ϵ_t are i.i.d. $N(0, I_m)$ and i.i.d. $N(0, I_d)$, respectively, and α_t denotes the state vector. The parameters admit the following dimensions:

$a_t \in R^m$	$d_t \in R^m$	$\eta_t \in R^m$
$T_t \in R^{m \times m}$	$H_t \in R^{m \times m}$	
$y_t \in R^d$	$c_t \in R^d$	$\epsilon_t \in R^d$
$Z_t \in R^{d \times m}$	$G_t \in R^{d \times d}$	

Note that `fkf.SP` takes as input `HHt` and `GGt` which corresponds to $H_t H_t'$ and $diag(G_t)^2$ respectively.

Sequential Processing Iteration:

Traditional Kalman filtering takes the entire observational vector y_t as the items for analysis. SP is an alternate approach that filters the elements of y_t one at a time. Sequential processing is described in the textbook of Durbin and Koopman (2001) and is described below.

Let p equal the number of observations at time t (i.e. when considering possible missing observations $p \leq d$). The SP iteration involves treating the vector series: $y_1, \cdots, y_n$ instead as the scalar series $y_{1,1}, \cdots, y_{(1,p)}, y_{2,1}, \cdots, y_{(n,p_n)}$. This univariate treatment of the multivariate series has the advantage that the function of the covariance matrix, F_t, becomes 1×1, avoiding the calculation of both the inverse and determinant of a $p \times p$ matrix. This can increase computational efficiency (especially under the case of many observations, i.e. p is large)

For any time point, the observation vector is given by:

$$y_t' = (y_{(t,1)}, \cdots, y_{(t,p)})$$

The filtering equations are written as:

$$a_{t,i+1} = a_{t,i} + K_{t,i} v_{t,i}$$

$$P_{t,i+1} = P_{t,i} - K_{t,i} F_{t,i} K_{t,i}'$$

Where:

$$\hat{y}_{t,i} = c_t + Z_t \cdot a_{t,i}$$

$$v_{t,i} = y_{t,i} - \hat{y}_{t,i}$$

$$F_{t,i} = Z_{t,i} P_{t,i} Z'_{t,i} + GGt_{t,i}$$
$$K_{t,i} = P_{t,i} Z'_{t,i} F^{-1}_{t,i}$$
$$i = 1, \cdots, p$$

Transition from time t to $t+1$ occurs through the standard transition equations.

$$\alpha_{t+1,1} = d_t + T_t \cdot \alpha_{t,p}$$

$$P_{t+1,1} = T_t \cdot P_{t,p} \cdot T'_t + HHt$$

The log-likelihood at time t is given by:

$$logL_t = -\frac{p}{2} log(2\pi) - \frac{1}{2} \sum_{i=1}^{p} (logF_i + \frac{v_i^2}{F_i})$$

Where the log-likelihood of observations is:

$$logL = \sum_{t}^{n} logL_t$$

Value

A `numeric` value corresponding to the log-likelihood calculated by the Kalman filter. Ideal for maximum likelihood estimation through optimization routines such as `optim`.

When `verbose = TRUE`, a list with the following elements is also returned, corresponding to the filtered state variables and covariances of the Kalman filter algorithm:

`att`	A $m \times n$-matrix containing the filtered state variables, i.e. `att[,t]` = $a_{t\|t}$.
`at`	A $m \times (n+1)$-matrix containing the predicted state variables, i.e. `at[,t]` = a_t.
`Ptt`	A $m \times m \times n$-array containing the variance of att, i.e. `Ptt[,,t]` = $P_{t\|t}$.
`logLik`	The log-likelihood.

Log-Likelihood Values:

When there are no missing observations (i.e. "NA" values) in argument `yt`, the return of function `fkf.SP` and the `logLik` object returned within the list of function `fkf` are identical. When NA's are present, however, log-likelihood values returned by `fkf.SP` are always higher. The log-likelihood value of the C code of `FKF` is instantiated through the calculation of the first term of the log-likelihood function, $-0.5 \times n \times d \times log(2\pi)$, where n is the number of columns of argument `yt` and d is the number of rows of argument `yt`. Under the assumption that there are missing observations, d would instead become d_t, where $d_t \leq d \forall t$. Whilst this doesn't influence parameter estimation, because observation matrix `yt` and thus the offset resulting from this is kept constant during maximum likelihood estimation, this does result in low bias of the log-likelihood values output by `fkf`.

References

Anderson, B. D. O. and Moore. J. B. (1979). *Optimal Filtering* Englewood Cliffs: Prentice-Hall.

Fahrmeir, L. and tutz, G. (1994) *Multivariate Statistical Modelling Based on Generalized Linear Models.* Berlin: Springer.

Koopman, S. J., Shephard, N., Doornik, J. A. (1999). Statistical algorithms for models in state space using SsfPack 2.2. *Econometrics Journal*, Royal Economic Society, vol. 2(1), pages 107-160.

Durbin, James, and Siem Jan Koopman (2001). *Time series analysis by state space methods.* Oxford university press.

David Luethi, Philipp Erb and Simon Otziger (2018). FKF: Fast Kalman Filter. R package version 0.1.5. https://CRAN.R-project.org/package=FKF

Examples

```
## <------------------------------------------------------------------------------
##Example 1 - Filter a state space model - Nile data
## <------------------------------------------------------------------------------

# Observations must be a matrix:
yt <- rbind(datasets::Nile)

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- matrix(1)
a0 <- yt[1]   # Estimation of the first year flow
P0 <- matrix(100)       # Variance of 'a0'
## These can be estimated through MLE:
GGt <- matrix(15000)
HHt <- matrix(1300)

# 'verbose' returns the filtered values:
output <- fkf.SP(a0 = a0, P0 = P0, dt = dt, ct = ct,
               Tt = Tt, Zt = Zt, HHt = HHt, GGt = GGt,
               yt = yt, verbose = TRUE)


## <------------------------------------------------------------------------------
##Example 2 - ARMA(2,1) model estimation.
## <------------------------------------------------------------------------------

#Length of series
n <- 1000

#AR parameters
AR <- c(ar1 = 0.6, ar2 = 0.2, ma1 = -0.2, sigma = sqrt(0.2))

## Sample from an ARMA(2, 1) process
a <- stats::arima.sim(model = list(ar = AR[c("ar1", "ar2")], ma = AR["ma1"]), n = n,
innov = rnorm(n) * AR["sigma"])

##State space representation of the four ARMA parameters
arma21ss <- function(ar1, ar2, ma1, sigma) {
Tt <- matrix(c(ar1, ar2, 1, 0), ncol = 2)
Zt <- matrix(c(1, 0), ncol = 2)
ct <- matrix(0)
dt <- matrix(0, nrow = 2)
GGt <- matrix(0)
H <- matrix(c(1, ma1), nrow = 2) * sigma
```

```
HHt <- H %*% t(H)
a0 <- c(0, 0)
P0 <- matrix(1e6, nrow = 2, ncol = 2)
return(list(a0 = a0, P0 = P0, ct = ct, dt = dt, Zt = Zt, Tt = Tt, GGt = GGt,
            HHt = HHt))
            }

## The objective function passed to 'optim'
objective <- function(theta, yt) {
sp <- arma21ss(theta["ar1"], theta["ar2"], theta["ma1"], theta["sigma"])
 ans <- fkf.SP(a0 = sp$a0, P0 = sp$P0, dt = sp$dt, ct = sp$ct, Tt = sp$Tt,
               Zt = sp$Zt, HHt = sp$HHt, GGt = sp$GGt, yt = yt)
 return(-ans)
}

## Parameter estimation - maximum likelihood estimation:
theta <- c(ar = c(0, 0), ma1 = 0, sigma = 1)
ARMA_MLE <- optim(theta, objective, yt = rbind(a), hessian = TRUE)


## <------------------------------------------------------------------------------
#Example 3 - Nile Model Estimation:
## <------------------------------------------------------------------------------

#Nile's annual flow:
yt <- rbind(Nile)

##Incomplete Nile Data - two NA's are present:
yt[c(3, 10)] <- NA

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- matrix(1)
a0 <- yt[1]   # Estimation of the first year flow
P0 <- matrix(100)       # Variance of 'a0'

## Parameter estimation - maximum likelihood estimation:
##Unknown parameters initial estimates:
GGt <- HHt <- var(c(yt), na.rm = TRUE) * .5
#Perform maximum likelihood estimation
Nile_MLE <- optim(c(HHt = HHt, GGt = GGt),
                fn = function(par, ...)
                -fkf.SP(HHt = matrix(par[1]), GGt = matrix(par[2]), ...),
                yt = yt, a0 = a0, P0 = P0, dt = dt, ct = ct,
                Zt = Zt, Tt = Tt)
## <------------------------------------------------------------------------------
#Example 4 - Dimensionless Treering Example:
## <------------------------------------------------------------------------------


## tree-ring widths in dimensionless units
y <- treering

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- matrix(1)
a0 <- y[1]            # Estimation of the first width
```

```
P0 <- matrix(100)     # Variance of 'a0'

## Parameter estimation - maximum likelihood estimation:
Treering_MLE <- optim(c(HHt = var(y, na.rm = TRUE) * .5,
                 GGt = var(y, na.rm = TRUE) * .5),
                 fn = function(par, ...)
                -fkf.SP(HHt = array(par[1],c(1,1,1)), GGt = matrix(par[2]), ...),
                 yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct,
                 Zt = Zt, Tt = Tt)
```

Index