Package: FKF.SP (via r-universe)

October 29, 2024

Title Fast Kalman Filtering Through Sequential Processing

Version 0.3.3

Description Fast and flexible Kalman filtering and smoothing 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/smoother.

License GPL-3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

RdMacros mathjaxr

Imports mathjaxr,

Suggests knitr, rmarkdown, stats, FKF, NFCP

VignetteBuilder knitr

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

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

Repository https://tomaspinall.r-universe.dev

RemoteUrl https://github.com/tomaspinall/fkf.sp

RemoteRef HEAD

RemoteSha 659fedeb90af102d3ab148ae8fa74fe8db975d51

Contents

---

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. 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,
  smoothing = 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 filtered values of the Kalman filter (see **Value**).

`smoothing` A `logical`. When `smoothing = TRUE`, Kalman smoothing is additionally performed and smoothed values returned (see **Value**).

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. Sequential processing has been empirically shown to grow linearly with respect to the dimensions of y_t, rather than exponentially as is the case with the traditional Kalman filter algorithm (Aspinall et al., 2022, P104).

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:

$$a_t \in R^m \qquad d_t \in R^m \qquad \eta_t \in R^m$$

$$
\begin{array}{lll}
T_t \in R^{m \times m} & H_t \in R^{m \times m} & \\
y_t \in R^d & c_t \in R^d & \epsilon_t \in R^d \\
Z_t \in R^{d \times m} & G_t \in R^{d \times d} &
\end{array}
$$

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:

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

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

$$F_{t,i} = Z_{t,i} P_{t,i} Z_{t,i}' + GGt_{t,i}$$

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

$$i = 1, \cdots, p$$

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:

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

Compiled Code:

`fkf.SP` wraps the C-functions `fkf_SP`, `fkf_SP_verbose` and `fkfs_SP`, which each rely upon the linear algebra subroutines of BLAS (Basic Linear Algebra Subprograms). These C-functions are called when `verbose = FALSE`, `verbose = TRUE` and `smoothing = TRUE`, respectively.

The difference in these compiled functions are in the values returned from them. The `fkfs_SP` also performs Kalman filtering and subsequently smoothing within the singular compiled C-code function.

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`, an S3 class of type 'fkf.SP' 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]` = $P_{t\|t}$.
`Pt`	A $m \times m \times (n+1)$-array containing the variance of at, i.e. `Pt[,,t]` = P_t.
`yt`	A $d \times n$-matrix containing the input observations.
`Tt`	either a $m \times m \times n$ or a $m \times m \times 1$-array, depending on the argument provided.
`Zt`	either a $d \times m \times n$ or a $d \times m \times 1$-array, depending on the argument provided.
`Ftinv`	A $d \times n$-matrix containing the scalar inverse of the prediction error variances.
`vt`	A $d \times n$-matrix containing the observation error.
`Kt`	A $m \times d \times n$-array containing the Kalman gain of state variables at each observation.
`logLik`	The log-likelihood.

In addition to the elements above, the following elements corresponding to the smoothed values output from Kalman smoothing are also returned when `smoothing = TRUE`. The `fks.SP` provides more detail regarding Kalman smoothing.

`ahatt`	A $m \times n$-matrix containing the smoothed state variables, i.e. `ahatt[,t]` = $a_{t\|n}$
`Vt`	A $m \times m \times n$-array containing the variances of `ahatt`, i.e. `Vt[,,t]` = $P_{t\|n}$

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. This is due to low bias in the log-likelihood values output by `fkf`, but does not influence parameter estimation. Further details are available within this package's vignette.

References

Aspinall, T. W., Harris, G., Gepp, A., Kelly, S., Southam, C., and Vanstone, B. (2022). *The Estimation of Commodity Pricing Models with Applications in Capital Investments*. Available Online.

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 0.2.3. '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)
```

---

fks.SP *Fast Kalman Smoother through Sequential Processing*

Description

The Kalman smoother is a backwards algorithm that is run after the Kalman filter that allows the user to refine estimates of previous states to produce "smoothed" estimates of state variables. This function performs the "Kalman smoother" algorithm using sequential processing, an approach that can substantially improve processing time over the traditional Kalman filtering/smoothing algorithms. The primary application of Kalman smoothing is in conjunction with expectation-maximization to estimate the parameters of a state space model. This function is called after running `fkf.SP`. `fks.SP` wraps the C-function `fks_SP` which relies upon the linear algebra subroutines of BLAS (Basic Linear Algebra Subprograms).

Usage

```
fks.SP(FKF.SP_obj)
```

Arguments

`FKF.SP_obj`	An S3-object of class "fkf.SP", returned by `fkf.SP` when using the argument `verbose = TRUE`.

Details

`fks.SP` is typically called after the `fkf.SP` function to calculate "smoothed" estimates of state variables and their corresponding variances. Smoothed estimates are used when utilizing expectation-maximization (EM) to efficiently estimate the parameters of a state space model.

Sequential Processing Kalman smoother solution:

The `fks.SP` function uses the solution to the Kalman smoother through sequential processing provided in the textbook of Durbin and Koopman (2001).

Given a state space model has been filtered through the sequential processing Kalman filter algorithm described in `fkf.SP`, the smoother can be reformulated for the univariate series:

$$y_t' = (y_{(1,1)}, y_{(1,2)}, \cdots, y_{(1,p_1)}, y_{(2,1)}, \cdots, y_{(t,p_t)})$$

The sequential processing Kalman smoother approach iterates backwards through both observations and time, i.e.: $i = p_t, \cdots, 1$ and $t = n, \cdots, 1$, where p_t is the number of observations at time t and n is the total number of observations.

The initialisations are:

$$r_{(n,p_n)} = 0$$
$$N_{(n,p_n)} = 0$$

Then, r and N are recursively calculated through:

$$L_{t,i} = I_m - K_{t,i} Z_{t,i}$$

$$r_{(t,i-1)} = Z_{t,i}' F_{t,i}^{-1} v_{t,i} + L_{t,i}' r_{t,i}$$

$$N_{t,i-1} = Z_{t,i}' F_{t,i}^{-1} Z_{t,i} + L_{t,i}' N_{t,i} L_{t,i}$$

$$r_{t-1,p_t} = T_{t-1}' r_{t,0}$$

$$N_{t-1,p_t} = T_{t-1}' N_{t,0} T_{t-1}$$

for $i = p_t, \cdots, 1$ and $t = n, \cdots, 1$

The equations for r_{t-1}, p_t and N_{t-1,p_t} do not apply for $t = 1$

Under this formulation, the values for $r_{t,0}$ and $N_{t,0}$ are the same as the values for the smoothing quantities of r_{t-1} and N_{t-1} of the standard smoothing equations, respectively.

The standard smoothing equations for $\hat{a}_t$ and V_t are used:

$$\hat{a}_t = a_t + P_t r_{t-1}$$

$$V_t = P_t - P_t N_{t-1} P_t$$

Where:

$$a_t = a_{t,1}$$

$$P_t = P_{t,1}$$

In the equations above, $r_{t,i}$ is an $m \times 1$ vector, I_m is an $m \times m$ identity matrix, $K_{t,i}$ is an $m \times 1$ column vector, $Z_{t,i}$ is a $1 \times m$ row vector, and both $F_{t,i}^{-1}$ and $v_{t,i}$ are scalars. The reduced dimensionality of many of the variables in this formulation compared to traditional Kalman smoothing can result in increased computational efficiency.

Finally, in the formulation described above, a_t and P_t correspond to the values of `att` and `ptt` returned from the `fkf.SP` function, respectively.

Value

An S3-object of class `fks.SP`, which is a list with the following elements:

`ahatt`	A $m \times n$-matrix containing the smoothed state variables, i.e. `ahatt[,t]` = $a_{t\|n}$
`Vt`	A $m \times m \times n$-array containing the variances of `ahatt`, i.e. `Vt[,,t]` = $P_{t\|n}$

References

Aspinall, T. W., Harris, G., Gepp, A., Kelly, S., Southam, C., and Vanstone, B. (2022). *The Estimation of Commodity Pricing Models with Applications in Capital Investments*. Available Online.

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

Examples

```
### Perform Kalman Filtering and Smoothing through sequential processing:
#Nile's annual flow:
yt <- Nile

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

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(yt, na.rm = TRUE) * .5
HHt = matrix(HHt)
GGt = matrix(GGt)
yt = rbind(yt)
# Filter through the Kalman filter - sequential processing:
Nile_filtered <- fkf.SP(HHt = matrix(HHt), GGt = matrix(GGt), a0 = a0, P0 = P0, dt = dt, ct = ct,
```

```
                  Zt = Zt, Tt = Tt, yt = rbind(yt), verbose = TRUE)
# Smooth filtered values through the Kalman smoother - sequential processing:
Smoothed_Estimates <- fks.SP(Nile_filtered)
```

Index