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:978-0-19-864117-8). '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 ................................................................. 2

Index 9
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 Al-
gebra 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 equa-
tion 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

```r
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 dis-
turbances 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 ad-
titional 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:
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 \( \eta_t \) and \( \epsilon_t \) are i.i.d. \( \mathcal{N}(0, I_m) \) and \( \mathcal{N}(0, I_d) \), respectively, and \( \alpha_t \) denotes the state vector.

The parameters admit the following dimensions:

- \( a_t \in \mathbb{R}^m \)
- \( d_t \in \mathbb{R}^{m \times m} \)
- \( T_t \in \mathbb{R}^{m \times m} \)
- \( H_t \in \mathbb{R}^{m \times m} \)
- \( y_t \in \mathbb{R}^d \)
- \( c_t \in \mathbb{R}^d \)
- \( Z_t \in \mathbb{R}^{d \times m} \)
- \( G_t \in \mathbb{R}^{d \times d} \)
- \( \eta_t \in \mathbb{R}^m \)
- \( \epsilon_t \in \mathbb{R}^d \)

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: \( y_1, \ldots, y_n \) instead as the scalar series \( y_{1,1}, \ldots, y_{1,p}, y_{2,1}, \ldots, y_{n,p_n} \). This univariate treatment of the multivariate series has the advantage that the function of the covariance matrix, \( F_t \), becomes \( 1 \times 1 \), 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:

\[ y'_t = (y(t,1), \ldots, y(t,p)) \]

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:

\[ \hat{y}_{t,i} = c_t + Z_t \cdot a_{t,i} \]
\[ v_{t,i} = y_{t,i} - \hat{y}_{t,i} \]
\[ F_{t,i} = Z_{t,i} P_{t,i} Z_{t,i}^T + GG_{t,i} \]
\[ K_{t,i} = P_{t,i} Z_{t,i}^T F_{t,i}^{-1} \]
\[ i = 1, \ldots, p \]

Transition from time \( t \) to \( t + 1 \) occurs through the standard transition equations.

\[ \alpha_{t+1,1} = d_t + T_t \cdot \alpha_{t,p} \]
\[ P_{t+1,1} = T_t \cdot P_{t,p} \cdot T_t^T + H H_t \]

The log-likelihood at time \( t \) is given by:

\[ \log L_t = -\frac{p}{2} \log(2\pi) - \frac{1}{2} \sum_{i=1}^{p} \left( \log F_i + \frac{v_i^2}{F_i} \right) \]

Where the log-likelihood of observations is:

\[ \log L = \sum_t \log L_t \]

**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 \( \text{verbose} = \text{TRUE} \), a list with the following elements is also returned, corresponding to the filtered state variables and covariances of the Kalman filter algorithm:

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

**Log-Likelihood Values:**

When there are no missing observations (i.e. "NA" values) in argument \( y_t \), the return of function \( \text{fkf.SP} \) and the \( \logL\text{ik} \) object returned within the list of function \( \text{fkf} \) are identical. When NA’s are present, however, log-likelihood values returned by \( \text{fkf.SP} \) 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 \( y_t \) and \( d \) is the number of rows of argument \( y_t \). Under the assumption that there are missing observations, \( d \) would instead become \( d_t \), where \( d_t \leq d \). Whilst this doesn’t influence parameter estimation, because observation matrix \( y_t \) 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 \( \text{fkf} \).

**References**

Examples

```r
## 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)
```

```r
## 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, ma1), ncol = 2)
  ct <- matrix(0)
  dt <- matrix(0, nrow = 2)
  GGt <- matrix(0)
  H <- matrix(c(1, ma1), nrow = 2) * sigma
  return(list(Tt = Tt, Zt = Zt, ct = ct, dt = dt, GGt = GGt, H = H))
}
```
fkk.SP

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 <- arma2Lss(theta[c("ar1")] , theta[c("ar2")] , theta[c("ma1")] , theta[c("sigma")])
  ans <- fkk.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, ...
                    -fkk.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

fkf. SP, 2