Package ‘FKF’

October 12, 2022

Title Fast Kalman Filter
Version 0.2.4
Description This is a fast and flexible implementation of the Kalman filter and smoother, which can deal with NAs. It is entirely written in C and relies fully on linear algebra subroutines contained in BLAS and LAPACK. Due to the speed of the filter, the fitting of high-dimensional linear state space models to large datasets becomes possible. This package also contains a plot function for the visualization of the state vector and graphical diagnostics of the residuals.
License GPL (>= 2)
Encoding UTF-8
Imports graphics
Suggests knitr, rmarkdown, testthat (>= 3.0.0)
Depends R (>= 3.5.0)
BugReports https://github.com/waternumbers/FKF/issues
NeedsCompilation yes
RoxygenNote 7.2.1
VignetteBuilder knitr
Config/testthat/edition 3
Author David Luethi [aut], Philipp Erb [aut], Simon Otziger [aut], Daniel McDonald [aut], Paul Smith [aut, cre] (<https://orcid.org/0000-0002-0034-3412>)
Maintainer Paul Smith <paul@waternumbers.co.uk>
Repository CRAN
Date/Publication 2022-10-10 19:50:02 UTC

1
**R topics documented:**

- `fkf` ........................................ 2
- `fks` ........................................ 6
- `plot.fkf` .................................. 8
- `plot.fks` .................................. 10

**Index**

<table>
<thead>
<tr>
<th>R function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fkf</code></td>
<td>2</td>
</tr>
<tr>
<td><code>fks</code></td>
<td>6</td>
</tr>
<tr>
<td><code>plot.fkf</code></td>
<td>8</td>
</tr>
<tr>
<td><code>plot.fks</code></td>
<td>10</td>
</tr>
</tbody>
</table>

---

**Description**

This function allows for fast and flexible Kalman filtering. Both, the measurement and transition equation may be multivariate and parameters are allowed to be time-varying. In addition “NA”-values in the observations are supported. `fkf` wraps the C-function `FKF` which fully relies on linear algebra subroutines contained in BLAS and LAPACK.

**Usage**

```r
fkf(a0, P0, dt, ct, Tt, Zt, HHt, GGt, yt)
```

**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 (see Details).
- `ct` A matrix giving the intercept of the measurement equation (see Details).
- `Tt` An array giving the factor of the transition equation (see Details).
- `Zt` An array giving the factor of the measurement equation (see Details).
- `HHt` An array giving the variance of the innovations of the transition equation (see Details).
- `GGt` An array giving the variance of the disturbances of the measurement equation (see Details).
- `yt` A matrix containing the observations. “NA”-values are allowed (see Details).

**Details**

**State space form**

The following notation is closest to the one of Koopman et al. The state space model is represented by the transition equation and the measurement equation. Let \( m \) be the dimension of the state variable, \( d \) be the dimension of the observations, and \( n \) the number of observations. The transition equation and the measurement equation are given by

\[
\alpha_{t+1} = d_t + T_t \cdot \alpha_t + H_t \cdot \eta_t
\]
\[ y_t = c_t + Z_t \cdot \alpha_t + G_t \cdot \epsilon_t, \]

where \( \eta_t \) and \( \epsilon_t \) are iid \( N(0, I_m) \) and iid \( N(0, I_d) \), respectively, and \( \alpha_t \) denotes the state variable.

The parameters admit the following dimensions:

\[
\begin{align*}
\alpha_t &\in \mathbb{R}^m \\
T_t &\in \mathbb{R}^{m \times m} \\
y_t &\in \mathbb{R}^d \\
Z_t &\in \mathbb{R}^{d \times m} \\
d_t &\in \mathbb{R}^m \\
H_t &\in \mathbb{R}^{m \times m} \\
e_t &\in \mathbb{R}^d \\
c_t &\in \mathbb{R}^d \\
G_t &\in \mathbb{R}^{d \times d}
\end{align*}
\]

Note that fkf takes as input \( HH_t \) and \( GG_t \) which corresponds to \( H_t H_t' \) and \( G_t G_t' \).

**Iteration:**

The filter iterations are implemented using the expected values

\[
\begin{align*}
a_t &= E[\alpha_t | y_1, \ldots, y_{t-1}] \\
a_{t|t} &= E[\alpha_t | y_1, \ldots, y_t]
\end{align*}
\]

and variances

\[
\begin{align*}
P_t &= Var[\alpha_t | y_1, \ldots, y_{t-1}] \\
P_{t|t} &= Var[\alpha_t | y_1, \ldots, y_t]
\end{align*}
\]

of the state \( \alpha_t \) in the following way (for the case of no NA’s):

**Initialisation:** Set \( t = 1 \) with \( a_t = a_0 \) and \( P_t = P_0 \)

**Updating equations:**

\[
\begin{align*}
v_t &= y_t - c_t - Z_t a_t \\
F_t &= Z_t P_t Z_t' + G_t G_t' \\
K_t &= P_t Z_t' F_t^{-1} \\
a_{t|t} &= a_t + K_t v_t \\
P_{t|t} &= P_t - P_t Z_t' K_t
\end{align*}
\]

**Prediction equations:**

\[
\begin{align*}
a_{t+1} &= d_t + T_t a_{t|t} \\
P_{t+1} &= T_t P_{t|t} T_t' + H_t H_t'
\end{align*}
\]

Next iteration: Set \( t = t + 1 \) and goto “Updating equations”.

**NA-values:**

NA-values in the observation matrix \( y_t \) are supported. If particular observations \( y_{t[,i]} \) contain NAs, the NA-values are removed and the measurement equation is adjusted accordingly. When the full vector \( y_{t[,i]} \) is missing the Kalman filter reduces to a prediction step.

**Parameters:**

The parameters can either be constant or deterministic time-varying. Assume the number of observations is \( n \) (i.e. \( y = (y_t)_{t=1,\ldots,n}, y_t = (y_{t,1}, \ldots, y_{t,d}) \)). 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.

**Htt** 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.

**Gtt** either a $d \times d \times n$ or a $d \times d \times 1$ array.

**yt** a $d \times n$ matrix.

**BLAS and LAPACK routines used:**

The R function `fkf` basically wraps the C-function `FKF`, which entirely relies on linear algebra subroutines provided by BLAS and LAPACK. The following functions are used:

BLAS: `dcopy`, `dgemm`, `daxpy`.

LAPACK: `dpotri`, `dpotrf`.

`FKF` is called through the `.Call` interface. Internally, `FKF` extracts the dimensions, allocates memory, and initializes the R-objects to be returned. `FKF` subsequently calls `cfkf` which performs the Kalman filtering.

The only critical part is to compute the inverse of $F_t$ and the determinant of $F_t$. If the inverse can not be computed, the filter stops and returns the corresponding message in `status` (see **Value**). If the computation of the determinant fails, the filter will continue, but the log-likelihood (element `logLik`) will be “NA”.

The inverse is computed in two steps: First, the Cholesky factorization of $F_t$ is calculated by `dpotrf`. Second, `dpotri` calculates the inverse based on the output of `dpotrf`.

The determinant of $F_t$ is computed using again the Cholesky decomposition.

The first element of both `at` and `Pt` is filled with the function arguments $a_0$ and $P_0$, and the last, i.e. the $(n + 1)$-th, element of `at` and `Pt` contains the predictions for the next time step.

**Value**

An S3-object of class “fkf”, which is a list with the following elements:

- `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 variances of `at`, i.e. `Pt[,t] = P_t`.
- `vt`  A $d \times n$-matrix of the prediction errors i.e. `vt[,t] = v_t`.
- `Ft`  A $d \times d \times n$-array which contains the variances of `vt`, i.e. `Ft[,t] = F_t`.
- `Kt`  A $m \times d \times n$-array containing the “Kalman gain” i.e. `Kt[,t] = k_t`.
- `logLik`  The log-likelihood.
- `status`  A vector which contains the status of LAPACK’s `dpotri` and `dpotrf`. (0, 0) means successful exit.
- `sys.time`  The time elapsed as an object of class “proc_time”.

**References**

See Also

`plot` to visualize and analyze `fkf`-objects, `KalmanRun` from the stats package, function `dlmFilter` from package `dlm`.

Examples

```r
## Example: Local level model for the Nile's annual flow.

## Transition equation:
## alpha[t+1] = alpha[t] + eta[t], eta[t] ~ N(0, HHT)
## Measurement equation:
## y[t] = alpha[t] + eps[t], eps[t] ~ N(0, GGT)

y <- Nile
y[c(3, 10)] <- NA  # NA values can be handled

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

## Estimate parameters:
fit.fkf <- optim(c(HHt = var(y, na.rm = TRUE) * .5, GGT = var(y, na.rm = TRUE) * .5),
                 fn = function(par, ...) -fkf(HHt = matrix(par[1]), GGT = matrix(par[2]), ...)$logLik,
                 yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct,
                 Zt = Zt, Tt = Tt)

## Filter Nile data with estimated parameters:
fkf.obj <- fkf(a0, P0, dt, ct, Tt, Zt, HHt = matrix(fit.fkf$par[1]),
               GGT = matrix(fit.fkf$par[2]), yt = rbind(y))

## Compare with the stats' structural time series implementation:
fit.stats <- StructTS(y, type = "level")
fit.fkf$par
fit.stats$coef

## Plot the flow data together with fitted local levels:
plot(y, main = "Nile flow")
lines(fitted(fit.stats), col = "green")
lines(ts(fkf.obj$att[1, ], start = start(y), frequency = frequency(y)), col = "blue")
legend("top", c("Nile flow data", "Local level (StructTS)", "Local level (fkf)"),
       col = c("black", "green", "blue"), lty = 1)
```


Fast Kalman Smoother

Description

This function can be run after running \texttt{fkf} to produce "smoothed" estimates of the state variable \( \alpha_t \). Unlike the output of the filter, these estimates are conditional on the entire set of \( n \) data points rather than only the past, see details.

Usage

\texttt{fks(FKFobj)}

Arguments

- \texttt{FKFobj} An S3-object of class "fkf", returned by \texttt{fkf}.

Details

The following notation is taken from the \texttt{fkf} function descriptions and is close to the one of Koopman et al. The smoother estimates

\[
\begin{align*}
\alpha_t|n &= E[\alpha_t|y_1, \ldots, y_n] \\
\varrho_t|n &= Var[\alpha_t|y_1, \ldots, y_n]
\end{align*}
\]

based on the outputs of the forward filtering pass performed by \texttt{fkf}.

The formulation of Koopman and Durbin is used which evolves the two values \( r_t \in \mathbb{R}^m \) and \( N_t \in \mathbb{R}^{m \times m} \) to avoid inverting the covariance matrix.

**Iteration:**

If there are no missing values the iteration proceeds as follows:

- **Initialisation:** Set \( t = n \), with \( r_t = 0 \) and \( N_t = 0 \).
- **Evolution equations:**
  \[
  \begin{align*}
  L &= T_t - T_t K_t Z_t \\
  r_{t-1} &= Z'_t F_t^{-1} v_t + L' r_t \\
  N_{t-1} &= Z'_t F_t^{-1} Z_t + L' N_t L
  \end{align*}
  \]
- **Updating equations:**
  \[
  \begin{align*}
  \alpha_t|n &= \alpha_t|t-1 + P_t|t-1 r_{t-1} \\
  \varrho_t|n &= P_t|t-1 - P_t|t-1 N_{t-1} P_t|t-1
  \end{align*}
  \]

Next iteration: Set \( t = t - 1 \) and goto “Evolution equations”.

Value

An S3-object of class "fks" which is a list with the following elements:

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

References


Examples

```r
## Example: Local level model for the Nile's annual flow.
## Transition equation:
## \( \alpha_{t+1} = \alpha_t + \eta_t \), \( \eta_t \sim N(0, HH_t) \)
## Measurement equation:
## \( y_t = \alpha_t + \varepsilon_t \), \( \varepsilon_t \sim N(0, GG_t) \)

y <- Nile
y[c(3, 10)] <- NA # NA values can be handled

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

fit.fkf <- optim(c(HHt = var(y, na.rm = TRUE) * .5,
                  GGt = var(y, na.rm = TRUE) * .5),
                  fn = function(par, ...) -fkf(HHt = matrix(par[1]), GGt = matrix(par[2]), ...)$logLik,
                  yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct,
                  Zt = Zt, Tt = Tt)

fkf.obj <- fkf(a0, P0, dt, ct, Tt, Zt, HHt = matrix(fit.fkf$par[1]),
               GGt = matrix(fit.fkf$par[2]), yt = rbind(y))

fks.obj <- fks(fkf.obj)

plot(y, main = "Nile flow")
lines(ts(fkf.obj$att[1, ], start = start(y), frequency = frequency(y)), col = "blue")
lines(ts(fks.obj$ahatt[1, ], start = start(y), frequency = frequency(y)), col = "red")
legend("top", c("Nile flow data", "Local level (fkf)", "Local level (fks)"),
        col = c("black", "green", "blue", "red"), lty = 1)
```
Description

Plotting method for objects of class `fkf`. This function provides tools for graphical analysis of the Kalman filter output: Visualization of the state vector, QQ-plot of the individual residuals, QQ-plot of the Mahalanobis distance, auto- as well as crosscorrelation function of the residuals.

Usage

```r
## S3 method for class 'fkf'
plot(
  x,  
  type = c("state", "resid.qq", "qqchisq", "acf"),  
  CI = 0.95,  
  at.idx = 1:nrow(x$at),  
  att.idx = 1:nrow(x$att),  
  ...  
)
```

Arguments

- `x` The output of `fkf`
- `type` A string stating what shall be plotted (see Details).
- `CI` The confidence interval in case `type == "state"`. Set `CI` to `NA` if no confidence interval shall be plotted.
- `at.idx` An vector giving the indexes of the predicted state variables which shall be plotted if `type == "state"`.
- `att.idx` An vector giving the indexes of the filtered state variables which shall be plotted if `type == "state"`.
- `...` Arguments passed to either `plot`, `qqnorm`, `qqplot` or `acf`.

Details

The argument `type` states what shall be plotted. `type` must partially match one of the following:

- `state` The state variables are plotted. By the arguments `at.idx` and `att.idx`, the user can specify which of the predicted (`at`) and filtered (`att`) state variables will be drawn.
- `resid.qq` Draws a QQ-plot for each residual-series `invt`.
- `qqchisq` A Chi-Squared QQ-plot will be drawn to graphically test for multivariate normality of the residuals based on the Mahalanobis distance.
- `acf` Creates a pairs plot with the autocorrelation function (`acf`) on the diagonal panels and the crosscorrelation function (`ccf`) of the residuals on the off-diagonal panels.
Value

Invisibly returns a list with components:

- **distance**: The Mahalanobis distance of the residuals as a vector of length \( n \).
- **std.resid**: The standardized residuals as an \( d \times n \)-matrix. It should hold that \( \text{std.resid}_{ij} \ iid \sim \mathcal{N}_d(0, I) \),

where \( d \) denotes the dimension of the data and \( n \) the number of observations.

usage

```r
plot(x, type = c("state", "resid.qq", "qqchisq", "acf"), CI = 0.95, at.idx = 1:nrow(x$at), att.idx = 1:nrow(x$att), ...)
```

See Also

fkf

Examples

```r
## <--------------------------------------------------------------------------->
## Example: Local level model for the tree ring data
## <--------------------------------------------------------------------------->
## Transition equation:
## \[ \alpha[t+1] = \alpha[t] + \eta[t], \eta[t] \sim N(0, HH_t) \]
## Measurement equation:
## \[ y[t] = \alpha[t] + \epsilon[t], \epsilon[t] \sim N(0, GG_t) \]

y <- treering
y[c(3, 10)] <- NA  # NA values can be handled

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- array(1,c(1,1,1))
a0 <- y[1]  # Estimation of the first width
P0 <- matrix(100)  # Variance of 'a0'

## Estimate parameters:
fit.fkf <- optim(c(HHt = var(y, na.rm = TRUE) * .5, 
                  GGt = var(y, na.rm = TRUE) * .5),
                 fn = function(par, ...) 
                 -fkf(HHt = array(par[1],c(1,1,1)), GGt = array(par[2],c(1,1,1)), ...)$logLik, 
                 yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct, 
                 Zt = Zt, Tt = Tt)

## Filter tree ring data with estimated parameters:
fkf.obj <- fkf(a0, P0, dt, ct, Tt, Zt, HHt = array(fit.fkf$par[1],c(1,1,1)), 
               GGt = array(fit.fkf$par[2],c(1,1,1)), yt = rbind(y))

## Plot the width together with fitted local levels:
plot(y, main = "Tree ring data")
lines(ts(fkf.obj$att[1, ], start = start(y), frequency = frequency(y)), col = "blue")
```
## Check the residuals for normality:
plot(fkf.obj, type = "resid.qq")

## Test for autocorrelation:
plot(fkf.obj, type = "acf", na.action = na.pass)
## y[t] = alpha[t] + eps[t], eps[t] ~ N(0, G\sigma_t)

y <- treering
y[c(3, 10)] <- NA # NA values can be handled

## Set constant parameters:
dt <- ct <- matrix(0)
Zt <- Tt <- array(1,c(1,1,1))
a0 <- y[1] # Estimation of the first width
P0 <- matrix(100) # Variance of 'a0'

## Estimate parameters:
fit.fkf <- optim(c(HHt = var(y, na.rm = TRUE) * .5, GGt = var(y, na.rm = TRUE) * .5),
                 fn = function(par, ...)
                 -fkf(HHt = array(par[1], c(1,1,1)), GGt = array(par[2], c(1,1,1)), ...)$logLik,
                 yt = rbind(y), a0 = a0, P0 = P0, dt = dt, ct = ct,
                 Zt = Zt, Tt = Tt)

## Filter tree ring data with estimated parameters:
fkf.obj <- fkf(a0, P0, dt, ct, Tt, Zt, HHt = array(fit.fkf$par[1], c(1,1,1)),
               GGt = array(fit.fkf$par[2], c(1,1,1)), yt = rbind(y))

fks.obj <- fks(fkf.obj)
plot(fks.obj)
lines(as.numeric(y), col="blue")
Index

* algebra
  fkf, 2

* hplot
  plot.fkf, 8

* models
  fkf, 2

* multivariate
  fkf, 2

* plot
  plot.fks, 10

acf, 8, 10
ccf, 8

fkf, 2, 6, 8, 9
fks, 6, 10

KalmanRun, 5

plot, 5, 8, 10
plot.fkf, 8
plot.fks, 10

qqnorm, 8, 10
qqplot, 8, 10