Package ‘FKF’

June 14, 2020

Title Fast Kalman Filter

Version 0.1.7

Description This is a fast and flexible implementation of the Kalman filter, 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 RUnit, knitr, rmarkdown, covr, pkgdown

Depends R(>= 2.8)

BugReports https://github.com/waternumbers/FKF/issues

URL https://waternumbers.github.io/FKF/,
https://github.com/waternumbers/FKF

NeedsCompilation yes

RoxygenNote 7.1.0

VignetteBuilder knitr

Author David Luethi [aut],
Philipp Erb [aut],
Simon Otziger [aut],
Paul Smith [cre] (<https://orcid.org/0000-0002-0034-3412>)

Maintainer Paul Smith <paul@waternumbers.co.uk>

Repository CRAN

Date/Publication 2020-06-14 14:50:15 UTC

R topics documented:

`fkf` *Fast Kalman filter*

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

```
fkf(a0, P0, dt, ct, Tt, Zt, HHt, GGt, yt, check.input = TRUE)
```

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**).
`check.input`	A `logical` stating whether the input shall be checked for consistency ("storage.mode", "class", and dimensionality, see **Details**). This input is depreciated and will be removed in a future version, checks are always made.

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

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

Note that `fkf` takes as input `HHt` and `GGt` which corresponds to $H_t H_t'$ and $G_t G_t'$.

Iteration:

Let `i` be the loop variable. The filter iterations are implemented the following way (in case of no NA's):

Initialization: `if(i == 1){ at[,i] = a0 Pt[,,i] = P0 }`

Updating equations:

```
vt[,i] = yt[,i] -ct[,i] -Zt[,,i] %*% at[,i]
Ft[,,i] = Zt[,,i] %*% Pt[,,i] %*% t(Zt[,,i]) + GGt[,,i]
Kt[,,i] = Pt[,,i] %*% t(Zt[,,i]) %*% solve(Ft[,,i])
att[,i] = at[,i] + Kt[,,i] %*% vt[,i]
Ptt[,i] = Pt[,,i] -Pt[,,i] %*% t(Zt[,,i]) %*% t(Kt[,,i])
```

Prediction equations:

```
at[,i + 1] = dt[,i] + Tt[,,i] %*% att[,i]
Pt[,,i + 1] = Tt[,,i] %*% Ptt[,,i] %*% t(Tt[,,i]) + HHt[,,i]
```

Next iteration:

```
i <-i + 1
```

goto "Updating equations".

NA-values:

NA-values in the observation matrix `yt` are supported. If particular observations `yt[,i]` contain NAs, the NA-values are removed and the measurement equation is adjusted accordingly. When the full vector `yt[,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. $y = (y_t)_{t=1,\ldots,n}, y_t = (y_{t1}, \ldots, y_{td})$). 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 $d \times d \times n$ 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.

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. $a_{t\|t} = E(\alpha_t \| y_t)$.
at	A $m \times (n+1)$-matrix containing the predicted state variables, i.e. $a_t = E(\alpha_t \| y_{t-1})$.
Ptt	A $m \times m \times n$-array containing the variance of att, i.e. $P_{t\|t} = var(\alpha_t \| y_t)$.
Pt	A $m \times m \times (n+1)$-array containing the variances of at, i.e. $P_t = var(\alpha_t \| y_{t-1})$.
vt	A $d \times n$-matrix of the prediction errors given by $v_t = y_t - c_t - Z_t a_t$.
Ft	A $d \times d \times n$-array which contains the variances of vt, i.e. $F_t = var(v_t)$.
Kt	A $m \times d \times n$-array containing the "Kalman gain" (ambiguity, see calculation above).
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".

The first element of both at and Pt is filled with the function arguments a0 and P0, and the last, i.e. the (n + 1)-th, element of at and Pt contains the predictions
$at[, n+1] = E(\alpha_{n+1} | y_n)$ and
$Pt[, , n+1] = var(\alpha_{n+1} | y_n)$.

Usage

```
fkf(a0,P0,dt,ct,Tt,Zt,HHt,GGt,yt,check.input = TRUE)
```

References

Harvey, Andrew C. (1990). *Forecasting, Structural Time Series Models and the Kalman Filter*. Cambridge University Press.

Hamilton, James D. (1994). *Time Series Analysis*. Princeton University Press.

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.

See Also

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

Examples

```
## <-------------------------------------------------------------------------->
## Example 1: ARMA(2, 1) model estimation.
## <-------------------------------------------------------------------------->
## This example shows how to fit an ARMA(2, 1) model using this Kalman
## filter implementation (see also stats' makeARIMA and KalmanRun).
n <- 1000

## Set the AR parameters
ar1 <- 0.6
ar2 <- 0.2
ma1 <- -0.2
sigma <- sqrt(0.2)

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

## Create a state space representation out 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(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$logLik)
}

theta <- c(ar = c(0, 0), ma1 = 0, sigma = 1)
fit <- optim(theta, objective, yt = rbind(a), hessian = TRUE)
fit

## Confidence intervals
rbind(fit$par - qnorm(0.975) * sqrt(diag(solve(fit$hessian))),
      fit$par + qnorm(0.975) * sqrt(diag(solve(fit$hessian))))

## Filter the series with estimated parameter values
sp <- arma21ss(fit$par["ar1"], fit$par["ar2"], fit$par["ma1"], fit$par["sigma"])
ans <- fkf(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 = rbind(a))

## Compare the prediction with the realization
plot(ans, at.idx = 1, att.idx = NA, CI = NA)
lines(a, lty = "dotted")

## Compare the filtered series with the realization
plot(ans, at.idx = NA, att.idx = 1, CI = NA)
lines(a, lty = "dotted")

## Check whether the residuals are Gaussian
plot(ans, type = "resid.qq")

## Check for linear serial dependence through 'acf'
plot(ans, type = "acf")


## <-------------------------------------------------------------------------->
## Example 2: 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)
```

`plot.fkf` *Plotting fkf objects*

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

```
## 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 (a_t) and filtered ($a_{t|t}$) 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-diagnoal panels.

Value

Invisibly returns an 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 $std.resid_{ij} \; iid \sim N_d(0, I)$,

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

usage

```
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

```
## <-------------------------------------------------------------------------->
## Example 3: Local level model for the treering data
## <-------------------------------------------------------------------------->
## 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 <- 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 = "Treering data")
lines(ts(fkf.obj$att[1, ], start = start(y), frequency = frequency(y)), col = "blue")
legend("top", c("Treering data", "Local level"), col = c("black", "blue"), lty = 1)

## Check the residuals for normality:
plot(fkf.obj, type = "resid.qq")

## Test for autocorrelation:
plot(fkf.obj, type = "acf", na.action = na.pass)
```

Index