

Package ‘kimfilter’

May 7, 2026

Type Package

Title Kim Filter

Version 2.0.0

Date 2026-05-01

Description 'Rcpp' implementation of the multivariate Kim filter, which combines the Kalman and Hamilton filters for state probability inference.

The filter is designed for state space models and can handle missing values and exogenous data in the observation and state equations.

Kim, Chang-Jin and Charles R. Nelson (1999) ``State-Space Models with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications" <<http://econ.korea.ac.kr/~cjkim/doi:10.7551/mitpress/6444.001.0001>><<http://econ.korea.ac.kr/~cjkim/>>.

License GPL (>= 2)

Imports Rcpp (>= 1.0.9)

LinkingTo Rcpp, RcppArmadillo

Suggests data.table (>= 1.14.2), maxLik (>= 1.5-2), ggplot2 (>= 3.3.6), gridExtra (>= 2.3), knitr, rmarkdown, testthat

VignetteBuilder knitr

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Alex Hubbard [aut, cre]

Maintainer Alex Hubbard <hubbard.alex@gmail.com>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2026-05-07 10:10:02 UTC

Contents

kim_filter	2
ss_prob	4
sw_dcf	5

kim_filter	<i>Kim Filter</i>
------------	-------------------

Description

Kim Filter

Usage

```
kim_filter(ssm, yt, Xo = NULL, Xs = NULL, weight = NULL, smooth = FALSE)
```

Arguments

ssm	list describing the state space model, must include names B0 - $N_b \times 1 \times n_state$ array of matrices, initial guess for the unobserved components P0 - $N_b \times N_b \times n_state$ array of matrices, initial guess for the covariance matrix of the unobserved components Dm - $N_b \times 1 \times n_state$ array of matrices, constant matrix for the state equation Am - $N_y \times 1 \times n_state$ array of matrices, constant matrix for the observation equation Fm - $N_b \times p \times n_state$ array of matrices, state transition matrix Hm - $N_y \times N_b \times n_state$ array of matrices, observation matrix Qm - $N_b \times N_b \times n_state$ array of matrices, state error covariance matrix Rm - $N_y \times N_y \times n_state$ array of matrices, state error covariance matrix betaO - $N_y \times N_o \times n_state$ array of matrices, coefficient matrix for the observation exogenous data betaS - $N_b \times N_s \times n_state$ array of matrices, coefficient matrix for the state exogenous data Pm - $n_state \times n_state$ matrix, state transition probability matrix
yt	$N \times T$ matrix of data
Xo	$N_o \times T$ matrix of exogenous observation data
Xs	$N_s \times T$ matrix of exogenous state
weight	column matrix of weights, $T \times 1$
smooth	boolean indication whether to run the backwards smoother

Value

list of cubes and matrices output by the Kim filter

Examples

```
## Not run:
#Stock and Watson Markov switching dynamic common factor
library(kimfilter)
library(data.table)
data(sw_dcf)
data = sw_dcf[, colnames(sw_dcf) != "dcoinc", with = FALSE]
vars = colnames(data)[colnames(data) != "date"]
```

```

#Set up the state space model
ssm = list()
ssm[["Fm"]] = rbind(c(0.8760, -0.2171, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0.0364, -0.0008, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, -0.2965, -0.0657, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0, -0.3959, -0.1903, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.2436, 0.1281),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0))
ssm[["Fm"]] = array(ssm[["Fm"]], dim = c(dim(ssm[["Fm"]]), 2))
ssm[["Dm"]] = matrix(c(-1.5700, rep(0, 11)), nrow = nrow(ssm[["Fm"]]), ncol = 1)
ssm[["Dm"]] = array(ssm[["Dm"]], dim = c(dim(ssm[["Dm"]]), 2))
ssm[["Dm"]][1, 2] = 0.2802
ssm[["Qm"]] = diag(c(1, 0, 0, 0, 0.0001, 0, 0.0001, 0, 0.0001, 0, 0.0001, 0))
ssm[["Qm"]] = array(ssm[["Qm"]], dim = c(dim(ssm[["Qm"]]), 2))
ssm[["Hm"]] = rbind(c(0.0058, -0.0033, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
                    c(0.0011, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0),
                    c(0.0051, -0.0033, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0),
                    c(0.0012, -0.0005, 0.0001, 0.0002, 0, 0, 0, 0, 0, 0, 1, 0))
ssm[["Hm"]] = array(ssm[["Hm"]], dim = c(dim(ssm[["Hm"]]), 2))
ssm[["Am"]] = matrix(0, nrow = nrow(ssm[["Hm"]]), ncol = 1)
ssm[["Am"]] = array(ssm[["Am"]], dim = c(dim(ssm[["Am"]]), 2))
ssm[["Rm"]] = matrix(0, nrow = nrow(ssm[["Am"]]), ncol = nrow(ssm[["Am"]]))
ssm[["Rm"]] = array(ssm[["Rm"]], dim = c(dim(ssm[["Rm"]]), 2))
ssm[["B0"]] = matrix(c(rep(-4.60278, 4), 0, 0, 0, 0, 0, 0, 0, 0))
ssm[["B0"]] = array(ssm[["B0"]], dim = c(dim(ssm[["B0"]]), 2))
ssm[["B0"]][1:4, 2] = rep(0.82146, 4)
ssm[["P0"]] = rbind(c(2.1775, 1.5672, 0.9002, 0.4483, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(1.5672, 2.1775, 1.5672, 0.9002, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0.9002, 1.5672, 2.1775, 1.5672, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0.4483, 0.9002, 1.5672, 2.1775, 0, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0.0001, 0, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0.0001, 0, 0, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0.0001, -0.0001, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, -0.0001, 0.0001, 0, 0, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0.0001, -0.0001, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, -0.0001, 0.0001, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0.0001, -0.0001, 0, 0),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0001, -0.0001),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, -0.0001, 0.0001))
ssm[["P0"]] = array(ssm[["P0"]], dim = c(dim(ssm[["P0"]]), 2))
ssm[["Pm"]] = rbind(c(0.8406, 0.0304),
                    c(0.1594, 0.9696))

#Log, difference and standardize the data
data[, c(vars) := lapply(.SD, log), .SDcols = c(vars)]
data[, c(vars) := lapply(.SD, function(x){
  x - shift(x, type = "lag", n = 1)
}), .SDcols = c(vars)]

```

```

data[, c(vars) := lapply(.SD, scale), .SDcols = c(vars)]

#Convert the data to an NxT matrix
yt = t(data[, c(vars), with = FALSE])
kf = kim_filter(ssm, yt, smooth = TRUE)

## End(Not run)

```

ss_prob

Steady State Probabilities

Description

Finds the steady state probabilities from a transition matrix $mat = \begin{bmatrix} p_{11} & p_{21} & \dots & p_{m1} \\ p_{12} & p_{22} & \dots & p_{m2} \\ \dots & \dots & \dots & \dots \\ p_{1m} & p_{2m} & \dots & p_{mm} \end{bmatrix}$ where the columns sum to 1

Usage

```
ss_prob(mat)
```

Arguments

mat square $S \times S$ matrix of probabilities with column sums of 1. S represents the number of states

Value

matrix of dimensions $S \times 1$ with steady state probabilities

Examples

```

## Not run:
Pm = rbind(c(0.8406, 0.0304),
           c(0.1594, 0.9696))
ss_prob(Pm)

## End(Not run)

```

sw_dcf	<i>Stock and Watson Markov Switching Dynamic Common Factor Data Set</i>
--------	---

Description

Stock and Watson Markov Switching Dynamic Common Factor Data Set

Usage

```
data(sw_dcf)
```

Format

data.table with columns DATE, VARIABLE, VALUE, and MATURITY The data is monthly frequency with variables ip (industrial production), gmyxpg (total personal income less transfer payments in 1987 dollars), mtq (total manufacturing and trade sales in 1987 dollars), lpnag (employees on non-agricultural payrolls), and dcoinc (the coincident economic indicator)

Source

Kim, Chang-Jin and Charles R. Nelson (1999) "State-Space Models with Regime Switching: Classical and Gibbs-Sampling Approaches with Applications" <doi:10.7551/mitpress/6444.001.0001><<http://econ.korea.ac.kr/~cjkim/>>.

Index

* **datasets**

sw_dcf, 5

kim_filter, 2

ss_prob, 4

sw_dcf, 5