Package ‘kalmanfilter’

March 8, 2024

Type Package

Title Kalman Filter

Version 2.1.1

Date 2024-03-01

Description 'Rcpp' implementation of the multivariate Kalman filter for state space models that can handle missing values and exogenous data in the observation and state equations. There is also a function to handle time varying parameters.
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

RoxygenNote 7.2.3

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

NeedsCompilation yes

Author Alex Hubbard [aut, cre]

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

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2024-03-08 05:00:06 UTC

R topics documented:

contains 2
gen_inv 2

kalman_filter 3
kalman_filter_cpp 5
Rginv 6
sw_dcf 7
treasuries 7

Index **8**

`contains` *Check if list contains a name*

Description

Check if list contains a name

Usage

```
contains(s, L)
```

Arguments

`s`	a string name
`L`	a list object

Value

boolean

`gen_inv` *Generalized matrix inverse*

Description

Generalized matrix inverse

Usage

```
gen_inv(m)
```

Arguments

`m`	matrix

Value

matrix inverse of m

`kalman_filter` *Kalman Filter*

Description

Kalman Filter

Usage

```
kalman_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 x 1 matrix (or array of length yt), initial guess for the unobserved components P0 - N_b x N_b matrix (or array of length yt), initial guess for the covariance matrix of the unobserved components Dm - N_b x 1 matrix (or array of length yt), constant matrix for the state equation Am - N_y x 1 matrix (or array of length yt), constant matrix for the observation equation Fm - N_b X p matrix (or array of length yt), state transition matrix Hm - N_y x N_b matrix (or array of length yt), observation matrix Qm - N_b x N_b matrix (or array of length yt), state error covariance matrix Rm - N_y x N_y matrix (or array of length yt), state error covariance matrix betaO - N_y x N_o matrix (or array of length yt), coefficient matrix for the observation exogenous data betaS - N_b x N_s matrix (or array of length yt), coefficient matrix for the state exogenous data
`yt`	N x T matrix of data
`Xo`	N_o x T matrix of exogenous observation data
`Xs`	N_s x T matrix of exogenous state
`weight`	column matrix of weights, T x 1
`smooth`	boolean indication whether to run the backwards smoother

Value

list of cubes and matrices output by the Kalman filter

Examples

```
## Not run:
#Stock and Watson Markov switching dynamic common factor
library(kalmanfilter)
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, 1, 0, 0, 0),
                  c(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, 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, 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, 0.0001, -0.0001),
                    c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.0001, 0.0001))
ssm[["P0"]] = array(ssm[["P0"]], dim = c(dim(ssm[["P0"]]), 2))

#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 = kalman_filter(ssm, yt, smooth = TRUE)

## End(Not run)
```

`kalman_filter_cpp` *Kalman Filter*

Description

Kalman Filter

Usage

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

Arguments

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

Value

list of matrices and cubes output by the Kalman filter

Examples

```
#Nelson-Siegel dynamic factor yield curve
library(kalmanfilter)
library(data.table)
data(treasuries)
tau = unique(treasuries$maturity)

#Set up the state space model
ssm = list()
ssm[["Fm"]] = rbind(c(0.9720, -0.0209, -0.0061),
```

```
                    c(0.1009 , 0.8189, -0.1446),
                    c(-0.1226, 0.0192, 0.8808))
ssm[["Dm"]] = matrix(c(0.1234, -0.2285, 0.2020), nrow = nrow(ssm[["Fm"]]), ncol = 1)
ssm[["Qm"]] = rbind(c(0.1017, 0.0937, 0.0303),
                    c(0.0937, 0.2267, 0.0351),
                    c(0.0303, 0.0351, 0.7964))
ssm[["Hm"]] = cbind(rep(1, 11),
                    -(1 - exp(-tau*0.0423))/(tau*0.0423),
                    (1 - exp(-tau*0.0423))/(tau*0.0423) - exp(-tau*0.0423))
ssm[["Am"]] = matrix(0, nrow = length(tau), ncol = 1)
ssm[["Rm"]] = diag(c(0.0087, 0, 0.0145, 0.0233, 0.0176, 0.0073,
                     0, 0.0016, 0.0035, 0.0207, 0.0210))
ssm[["B0"]] = matrix(c(5.9030, -0.7090, 0.8690), nrow = nrow(ssm[["Fm"]]), ncol = 1)
ssm[["P0"]] = diag(rep(0.0001, nrow(ssm[["Fm"]])))

#Convert to an NxT matrix
yt = dcast(treasuries, "date ~ maturity", value.var = "value")
yt = t(yt[, 2:ncol(yt)])
kf = kalman_filter(ssm, yt, smooth = TRUE)
```

`Rginv` *R's implementation of the Moore-Penrose pseudo matrix inverse*

Description

R's implementation of the Moore-Penrose pseudo matrix inverse

Usage

```
Rginv(m)
```

Arguments

`m`	matrix

Value

matrix inverse of m

sw_dcf *Stock and Watson Dynamic Common Factor Data Set*

Description

Stock and Watson 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/~c

treasuries *Treasuries*

Description

Treasuries

Usage

```
data(treasuries)
```

Format

data.table with columns DATE, VARIABLE, VALUE, and MATURITY The data is quarterly frequency with variables DGS1MO, DGS3MO, DGS6MO, DGS1, DGS2, DGS3, DGS5, DGS7, DGS10, DGS20, and DGS30

Source

FRED

Index

∗ **datasets**
    sw_dcf, 7
    treasuries, 7

contains, 2

gen_inv, 2

kalman_filter, 3
kalman_filter_cpp, 5

Rginv, 6

sw_dcf, 7

treasuries, 7