Package: dlm (via r-universe)

October 28, 2026

Title Bayesian and Likelihood Analysis of Dynamic Linear Models

Version 1.1-6.1

Date 2022-11-12

Suggests MASS

Imports stats, utils, methods, grDevices, graphics

Maintainer Giovanni Petris <GPetris@uark.edu>

Description Provides routines for Maximum likelihood, Kalman filtering and smoothing, and Bayesian analysis of Normal linear State Space models, also known as Dynamic Linear Models.

License GPL (>= 2)

NeedsCompilation yes

Author Giovanni Petris [aut, cre], Wally Gilks [ctb] (Author of original C code for ARMS)

Repository https://cran.r-universe.dev

Date/Publication 2024-09-22 03:02:14 UTC

RemoteUrl https://github.com/cran/dlm

RemoteRef HEAD

RemoteSha 64c187ac71436c1be693978ebe4568d863078391

Contents

arms 2
ARtransPars 5
bdiag 6
convex.bounds 7
dlm 8
dlmBSample 9
dlmFilter 10
dlmForecast 12
dlmGibbsDIG 13
dlmLL 15

dlmMLE 16
dlmModARMA 17
dlmModPoly 19
dlmModReg 20
dlmModSeas 21
dlmModTrig 22
dlmRandom 23
dlmSmooth 24
dlmSum 26
dlmSvd2var 27
dropFirst 28
FF 29
mcmc 31
NelPlo 32
residuals.dlmFiltered 33
rwishart 34
USecon 35

Index **37**

`arms` *Function to perform Adaptive Rejection Metropolis Sampling*

Description

Generates a sequence of random variables using ARMS. For multivariate densities, ARMS is used along randomly selected straight lines through the current point.

Usage

```
arms(y.start, myldens, indFunc, n.sample, ...)
```

Arguments

`y.start`	initial point
`myldens`	univariate or multivariate log target density
`indFunc`	indicator function of the convex support of the target density
`n.sample`	desired sample size
`...`	parameters passed to `myldens` and `indFunc`

Details

Strictly speaking, the support of the target density must be a bounded convex set. When this is not the case, the following tricks usually work. If the support is not bounded, restrict it to a bounded set having probability practically one. A workaround, if the support is not convex, is to consider the convex set generated by the support and define `myldens` to return `log(.Machine$double.xmin)` outside the true support (see the last example.)

The next point is generated along a randomly selected line through the current point using arms.

Make sure the value returned by `myldens` is never smaller than `log(.Machine$double.xmin)`, to avoid divisions by zero.

Value

An `n.sample` by `length(y.start)` matrix, whose rows are the sampled points.

Note

The function is based on original C code by W. Gilks for the univariate case.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Gilks, W.R., Best, N.G. and Tan, K.K.C. (1995) Adaptive rejection Metropolis sampling within Gibbs sampling (Corr: 97V46 p541-542 with Neal, R.M.), *Applied Statistics* **44**:455–472.

Examples

```
#### ==> Warning: running the examples may take a few minutes! <== ####

set.seed(4521222)
### Univariate densities
## Unif(-r,r)
y <- arms(runif(1,-1,1), function(x,r) 1, function(x,r) (x>-r)*(x<r), 5000, r=2)
summary(y); hist(y,prob=TRUE,main="Unif(-r,r); r=2")
## Normal(mean,1)
norldens <- function(x,mean) -(x-mean)^2/2
y <- arms(runif(1,3,17), norldens, function(x,mean) ((x-mean)>-7)*((x-mean)<7),
          5000, mean=10)
summary(y); hist(y,prob=TRUE,main="Gaussian(m,1); m=10")
curve(dnorm(x,mean=10),3,17,add=TRUE)
## Exponential(1)
y <- arms(5, function(x) -x, function(x) (x>0)*(x<70), 5000)
summary(y); hist(y,prob=TRUE,main="Exponential(1)")
curve(exp(-x),0,8,add=TRUE)
## Gamma(4.5,1)
y <- arms(runif(1,1e-4,20), function(x) 3.5*log(x)-x,
          function(x) (x>1e-4)*(x<20), 5000)
summary(y); hist(y,prob=TRUE,main="Gamma(4.5,1)")
```

```
curve(dgamma(x,shape=4.5,scale=1),1e-4,20,add=TRUE)
## Gamma(0.5,1) (this one is not log-concave)
y <- arms(runif(1,1e-8,10), function(x) -0.5*log(x)-x,
          function(x) (x>1e-8)*(x<10), 5000)
summary(y); hist(y,prob=TRUE,main="Gamma(0.5,1)")
curve(dgamma(x,shape=0.5,scale=1),1e-8,10,add=TRUE)
## Beta(.2,.2) (this one neither)
y <- arms(runif(1), function(x) (0.2-1)*log(x)+(0.2-1)*log(1-x),
          function(x) (x>1e-5)*(x<1-1e-5), 5000)
summary(y); hist(y,prob=TRUE,main="Beta(0.2,0.2)")
curve(dbeta(x,0.2,0.2),1e-5,1-1e-5,add=TRUE)
## Triangular
y <- arms(runif(1,-1,1), function(x) log(1-abs(x)), function(x) abs(x)<1, 5000)
summary(y); hist(y,prob=TRUE,ylim=c(0,1),main="Triangular")
curve(1-abs(x),-1,1,add=TRUE)
## Multimodal examples (Mixture of normals)
lmixnorm <- function(x,weights,means,sds) {
    log(crossprod(weights, exp(-0.5*((x-means)/sds)^2 - log(sds))))
}
y <- arms(0, lmixnorm, function(x,...) (x>(-100))*(x<100), 5000, weights=c(1,3,2),
          means=c(-10,0,10), sds=c(1.5,3,1.5))
summary(y); hist(y,prob=TRUE,main="Mixture of Normals")
curve(colSums(c(1,3,2)/6*dnorm(matrix(x,3,length(x),byrow=TRUE),c(-10,0,10),c(1.5,3,1.5))),
      par("usr")[1], par("usr")[2], add=TRUE)

### Bivariate densities
## Bivariate standard normal
y <- arms(c(0,2), function(x) -crossprod(x)/2,
          function(x) (min(x)>-5)*(max(x)<5), 500)
plot(y, main="Bivariate standard normal", asp=1)
## Uniform in the unit square
y <- arms(c(0.2,.6), function(x) 1,
          function(x) (min(x)>0)*(max(x)<1), 500)
plot(y, main="Uniform in the unit square", asp=1)
polygon(c(0,1,1,0),c(0,0,1,1))
## Uniform in the circle of radius r
y <- arms(c(0.2,0), function(x,...) 1,
          function(x,r2) sum(x^2)<r2, 500, r2=2^2)
plot(y, main="Uniform in the circle of radius r; r=2", asp=1)
curve(-sqrt(4-x^2), -2, 2, add=TRUE)
curve(sqrt(4-x^2), -2, 2, add=TRUE)
## Uniform on the simplex
simp <- function(x) if ( any(x<0) || (sum(x)>1) ) 0 else 1
y <- arms(c(0.2,0.2), function(x) 1, simp, 500)
plot(y, xlim=c(0,1), ylim=c(0,1), main="Uniform in the simplex", asp=1)
polygon(c(0,1,0), c(0,0,1))
## A bimodal distribution (mixture of normals)
bimodal <- function(x) { log(prod(dnorm(x,mean=3))+prod(dnorm(x,mean=-3))) }
y <- arms(c(-2,2), bimodal, function(x) all(x>(-10))*all(x<(10)), 500)
plot(y, main="Mixture of bivariate Normals", asp=1)

## A bivariate distribution with non-convex support
support <- function(x) {
```

```
    return(as.numeric( -1 < x[2] && x[2] < 1 &&
                       -2 < x[1] &&
                       ( x[1] < 1 || crossprod(x-c(1,0)) < 1 ) ) )
}
Min.log <- log(.Machine$double.xmin) + 10
logf <- function(x) {
    if ( x[1] < 0 ) return(log(1/4))
    else
        if (crossprod(x-c(1,0)) < 1 ) return(log(1/pi))
    return(Min.log)
}
x <- as.matrix(expand.grid(seq(-2.2,2.2,length=40),seq(-1.1,1.1,length=40)))
y <- sapply(1:nrow(x), function(i) support(x[i,]))
plot(x,type='n',asp=1)
points(x[y==1,],pch=1,cex=1,col='green')
z <- arms(c(0,0), logf, support, 1000)
points(z,pch=20,cex=0.5,col='blue')
polygon(c(-2,0,0,-2),c(-1,-1,1,1))
curve(-sqrt(1-(x-1)^2),0,2,add=TRUE)
curve(sqrt(1-(x-1)^2),0,2,add=TRUE)
sum( z[,1] < 0 ) # sampled points in the square
sum( apply(t(z)-c(1,0),2,crossprod) < 1 ) # sampled points in the circle
```

ARtransPars *Function to parametrize a stationary AR process*

Description

The function maps a vector of length p to the vector of autoregressive coefficients of a stationary AR(p) process. It can be used to parametrize a stationary AR(p) process

Usage

```
ARtransPars(raw)
```

Arguments

raw	a vector of length p

Details

The function first maps each element of raw to (0,1) using tanh. The numbers obtained are treated as the first partial autocorrelations of a stationary AR(p) process and the vector of the corresponding autoregressive coefficients is computed and returned.

Value

The vector of autoregressive coefficients of a stationary AR(p) process corresponding to the parameters in raw.

Author(s)

Giovanni Petris, <GPetris@uark.edu>

References

Jones, 1987. Randomly choosing parameters from the stationarity and invertibility region of autoregressive-moving average models. *Applied Statistics*, 36.

Examples

```
(ar <- ARtransPars(rnorm(5)))
all( Mod(polyroot(c(1,-ar))) > 1 ) # TRUE
```

---

bdiag *Build a block diagonal matrix*

Description

The function builds a block diagonal matrix.

Usage

```
bdiag(...)
```

Arguments

...	individual matrices, or a list of matrices.

Value

A matrix obtained by combining the arguments.

Author(s)

Giovanni Petris <GPetris@uark.edu>

Examples

```
bdiag(matrix(1:4,2,2),diag(3))
bdiag(matrix(1:6,3,2),matrix(11:16,2,3))
```

convex.bounds *Find the boundaries of a convex set*

Description

Finds the boundaries of a bounded convex set along a specified straight line, using a bisection approach. It is mainly intended for use within arms.

Usage

```
convex.bounds(x, dir, indFunc, ..., tol=1e-07)
```

Arguments

x	a point within the set
dir	a vector specifying a direction
indFunc	indicator function of the set
...	parameters passed to indFunc
tol	tolerance

Details

Uses a bisection algorithm along a line having parametric representation x + t * dir.

Value

A vector ans of length two. The boundaries of the set are x + ans[1] * dir and x + ans[2] * dir.

Author(s)

Giovanni Petris <GPetris@uark.edu>

Examples

```
## boundaries of a unit circle
convex.bounds(c(0,0), c(1,1), indFunc=function(x) crossprod(x)<1)
```

dlm *dlm objects*

Description

The function dlm is used to create Dynamic Linear Model objects. as.dlm and is.dlm coerce an object to a Dynamic Linear Model object and test whether an object is a Dynamic Linear Model.

Usage

```
dlm(...)
as.dlm(obj)
is.dlm(obj)
```

Arguments

...	list with named elements m0, C0, FF, V, GG, W and, optionally, JFF, JV, JGG, JW, and X. The first six are the usual vector and matrices that define a time-invariant DLM. The remaining elements are used for time-varying DLM. X, if present, should be a matrix. If JFF is not NULL, then it must be a matrix of the same dimension of FF, with the (i, j) element being zero if FF[i,j] is time-invariant, and a positive integer k otherwise. In this case the (i, j) element of FF at time t will be X[t,k]. A similar interpretation holds for JV, JGG, and JW. ... may have additional components, that are not used by dlm. The named components may also be passed to the function as individual arguments.
obj	an arbitrary R object.

Details

The function dlm is used to create Dynamic Linear Model objects. These are lists with the named elements described above and with class of "dlm".

Class "dlm" has a number of methods. In particular, consistent DLM can be added together to produce another DLM.

Value

For dlm, an object of class "dlm".

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer (1997).

See Also

dlmModReg, dlmModPoly, dlmModARMA, dlmModSeas, to create particular objects of class "dlm".

Examples

```
## Linear regression as a DLM
x <- matrix(rnorm(10),nc=2)
mod <- dlmModReg(x)
is.dlm(mod)

## Adding dlm's
dlmModPoly() + dlmModSeas(4) # linear trend plus quarterly seasonal component
```

---

dlmBSample *Draw from the posterior distribution of the state vectors*

Description

The function simulates one draw from the posterior distribution of the state vectors.

Usage

```
dlmBSample(modFilt)
```

Arguments

modFilt	a list, typically the ouptut from dlmFilter, with elements m, U.C, D.C, a, U.R, D.R (see the value returned by dlmFilter), and mod The latter is an object of class "dlm" or a list with elements GG, W and, optionally, JGG, JW, and X

Details

The calculations are based on singular value decomposition.

Value

The function returns a draw from the posterior distribution of the state vectors. If m is a time series then the returned value is a time series with the same tsp, otherwise it is a matrix or vector.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer (1997).

See Also

See also dlmFilter

Examples

```
nileMod <- dlmModPoly(1, dV = 15099.8, dW = 1468.4)
nileFilt <- dlmFilter(Nile, nileMod)
nileSmooth <- dlmSmooth(nileFilt) # estimated "true" level
plot(cbind(Nile, nileSmooth$s[-1]), plot.type = "s",
     col = c("black", "red"), ylab = "Level",
     main = "Nile river", lwd = c(2, 2))
for (i in 1:10) # 10 simulated "true" levels
    lines(dlmBSample(nileFilt[-1]), lty=2)
```

---

dlmFilter *DLM filtering*

Description

The functions applies Kalman filter to compute filtered values of the state vectors, together with their variance/covariance matrices. By default the function returns an object of class "dlmFiltered". Methods for residuals and tsdiag for objects of class "dlmFiltered" exist.

Usage

```
dlmFilter(y, mod, debug = FALSE, simplify = FALSE)
```

Arguments

y	the data. y can be a vector, a matrix, a univariate or multivariate time series.
mod	an object of class dlm, or a list with components m0, C0, FF, V, GG, W, and optionally JFF, JV, JGG, JW, and X, defining the model and the parameters of the prior distribution.
debug	if FALSE, faster C code will be used, otherwise all the computations will be performed in R.
simplify	should the data be included in the output?

Details

The calculations are based on the singular value decomposition (SVD) of the relevant matrices. Variance matrices are returned in terms of their SVD.

Missing values are allowed in y.

Value

A list with the components described below. If `simplify` is `FALSE`, the returned list has class `"dlmFiltered"`.

`y`	The input data, coerced to a matrix. This is present only if `simplify` is `FALSE`.
`mod`	The argument `mod` (possibly simplified).
`m`	Time series (or matrix) of filtered values of the state vectors. The series starts one time unit before the first observation.
`U.C`	See below.
`D.C`	Together with `U.C`, it gives the SVD of the variances of the estimation errors. The variance of $m[t,] - theta[t,]$ is given by `U.C[[t]] %*% diag(D.C[t,]^2) %*% t(U.C[[t]])`.
`a`	Time series (or matrix) of predicted values of the state vectors given the observations up and including the previous time unit.
`U.R`	See below.
`D.R`	Together with `U.R`, it gives the SVD of the variances of the prediction errors. The variance of $a[t,] - theta[t,]$ is given by `U.R[[t]] %*% diag(D.R[t,]^2) %*% t(U.R[[t]])`.
`f`	Time series (or matrix) of one-step-ahead forecast of the observations.

Warning

The observation variance `V` in `mod` must be nonsingular.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Zhang, Y. and Li, X.R., Fixed-interval smoothing algorithm based on singular value decomposition, *Proceedings of the 1996 IEEE International Conference on Control Applications*.
Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).

See Also

See `dlm` for a description of dlm objects, `dlmSvd2var` to obtain a variance matrix from its SVD, `dlmMLE` for maximum likelihood estimation, `dlmSmooth` for Kalman smoothing, and `dlmBSample` for drawing from the posterior distribution of the state vectors.

Examples

```
nileBuild <- function(par) {
  dlmModPoly(1, dV = exp(par[1]), dW = exp(par[2]))
}
nileMLE <- dlmMLE(Nile, rep(0,2), nileBuild); nileMLE$conv
nileMod <- nileBuild(nileMLE$par)
V(nileMod)
W(nileMod)
nileFilt <- dlmFilter(Nile, nileMod)
nileSmooth <- dlmSmooth(nileFilt)
plot(cbind(Nile, nileFilt$m[-1], nileSmooth$s[-1]), plot.type='s',
     col=c("black","red","blue"), ylab="Level", main="Nile river", lwd=c(1,2,2))
```

---

dlmForecast *Prediction and simulation of future observations*

Description

The function evaluates the expected value and variance of future observations and system states. It can also generate a sample from the distribution of future observations and system states.

Usage

```
dlmForecast(mod, nAhead = 1, method = c("plain", "svd"), sampleNew = FALSE)
```

Arguments

mod	an object of class "dlm", or a list with components m0, C0, FF, V, GG, and W, defining the model and the parameters of the prior distribution. mod can also be an object of class "dlmFiltered", such as the output from dlmFilter.
nAhead	number of steps ahead for which a forecast is requested.
method	method="svd" uses singular value decomposition for the calculations. Currently, only method="plain" is implemented.
sampleNew	if sampleNew=n for an integer n, them a sample of size n from the forecast distribution of states and observables will be returned.

Value

A list with components

a	matrix of expected values of future states
R	list of variances of future states
f	matrix of expected values of future observations
Q	list of variances of future observations
newStates	list of matrices containing the simulated future values of the states. Each component of the list corresponds

to one simulation.

`newObs` same as `newStates`, but for the observations.

The last two components are not present if `sampleNew=FALSE`.

Note

The function is currently entirely written in R and is not particularly fast. Currently, only constant models are allowed.

Author(s)

Giovanni Petris <GPetris@uark.edu>

Examples

```
## Comparing theoretical prediction intervals with sample quantiles
set.seed(353)
n <- 20; m <- 1; p <- 5
mod <- dlmModPoly() + dlmModSeas(4, dV=0)
W(mod) <- rwishart(2*p,p) * 1e-1
m0(mod) <- rnorm(p, sd=5)
C0(mod) <- diag(p) * 1e-1
new <- 100
fore <- dlmForecast(mod, nAhead=n, sampleNew=new)
ciTheory <- (outer(sapply(fore$Q, FUN=function(x) sqrt(diag(x))), qnorm(c(0.1,0.9))) +
             as.vector(t(fore$f)))
ciSample <- t(apply(array(unlist(fore$newObs), dim=c(n,m,new))[,1,], 1,
                    FUN=function(x) quantile(x, c(0.1,0.9))))
plot.ts(cbind(ciTheory,fore$f[,1]),plot.type="s", col=c("red","red","green"),ylab="y")
for (j in 1:2) lines(ciSample[,j], col="blue")
legend(2,-40,legend=c("forecast mean", "theoretical bounds", "Monte Carlo bounds"),
       col=c("green","red","blue"), lty=1, bty="n")
```

---

dlmGibbsDIG *Gibbs sampling for d-inverse-gamma model*

Description

The function implements a Gibbs sampler for a univariate DLM having one or more unknown variances in its specification.

Usage

```
dlmGibbsDIG(y, mod, a.y, b.y, a.theta, b.theta, shape.y, rate.y,
            shape.theta, rate.theta, n.sample = 1,
            thin = 0, ind, save.states = TRUE,
            progressBar = interactive())
```

Arguments

y	data vector or univariate time series
mod	a dlm for univariate observations
a.y	prior mean of observation precision
b.y	prior variance of observation precision
a.theta	prior mean of system precisions (recycled, if needed)
b.theta	prior variance of system precisions (recycled, if needed)
shape.y	shape parameter of the prior of observation precision
rate.y	rate parameter of the prior of observation precision
shape.theta	shape parameter of the prior of system precisions (recycled, if needed)
rate.theta	rate parameter of the prior of system precisions (recycled, if needed)
n.sample	requested number of Gibbs iterations
thin	discard thin iterations for every saved iteration
ind	indicator of the system variances that need to be estimated
save.states	should the simulated states be included in the output?
progressBar	should a text progress bar be displayed during execution?

Details

The *d-inverse-gamma* model is a constant univariate DLM with unknown observation variance, diagonal system variance with unknown diagonal entries. Some of these entries may be known, in which case they are typically zero. Independent inverse gamma priors are assumed for the unknown variances. These can be specified be mean and variance or, alternatively, by shape and rate. Recycling is applied for the prior parameters of unknown system variances. The argument ind can be used to specify the index of the unknown system variances, in case some of the diagonal elements of W are known. The unobservable states are generated in the Gibbs sampler and are returned if save.states = TRUE. For more details on the model and usage examples, see the package vignette.

Value

The function returns a list of simulated values.

dV	simulated values of the observation variance.
dW	simulated values of the unknown diagonal elements of the system variance.
theta	simulated values of the state vectors.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).

Examples

```
## See the package vignette for an example
```

---

dlmLL *Log likelihood evaluation for a state space model*

Description

Function that computes the log likelihood of a state space model.

Usage

```
dlmLL(y, mod, debug=FALSE)
```

Arguments

y	a vector, matrix, or time series of data.
mod	an object of class "dlm", or a list with components m0, C0, FF, V, GG, W defining the model and the parameters of the prior distribution.
debug	if debug=TRUE, the function uses R code, otherwise it uses faster C code.

Details

The calculations are based on singular value decomposition. Missing values are allowed in y.

Value

The function returns the negative of the loglikelihood.

Warning

The observation variance V in mod must be nonsingular.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Durbin and Koopman, Time series analysis by state space methods, Oxford University Press, 2001.

See Also

dlmMLE, dlmFilter for the definition of the equations of the model.

Examples

```
##---- See the examples for dlmMLE ----
```

dlmMLE *Parameter estimation by maximum likelihood*

Description

The function returns the MLE of unknown parameters in the specification of a state space model.

Usage

```
dlmMLE(y, parm, build, method = "L-BFGS-B", ..., debug = FALSE)
```

Arguments

y	a vector, matrix, or time series of data.
parm	vector of initial values - for the optimization routine - of the unknown parameters.
build	a function that takes a vector of the same length as parm and returns an object of class dlm, or a list that may be interpreted as such.
method	passed to optim.
...	additional arguments passed to optim and build.
debug	if debug=TRUE, the likelihood calculations are done entirely in R, otherwise C functions are used.

Details

The evaluation of the loglikelihood is done by dlmLL. For the optimization, optim is called. It is possible for the model to depend on additional parameters, other than those in parm, passed to build via the ... argument.

Value

The function dlmMLE returns the value returned by optim.

Warning

The build argument must return a dlm with nonsingular observation variance V.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).

See Also

dlmLL, dlm.

Examples

```
data(NelPlo)
### multivariate local level -- seemingly unrelated time series
buildSu <- function(x) {
  Vsd <- exp(x[1:2])
  Vcorr <- tanh(x[3])
  V <- Vsd %o% Vsd
  V[1,2] <- V[2,1] <- V[1,2] * Vcorr
  Wsd <- exp(x[4:5])
  Wcorr <- tanh(x[6])
  W <- Wsd %o% Wsd
  W[1,2] <- W[2,1] <- W[1,2] * Wcorr
  return(list(
              m0 = rep(0,2),
              C0 = 1e7 * diag(2),
              FF = diag(2),
              GG = diag(2),
              V = V,
              W = W))
}

suMLE <- dlmMLE(NelPlo, rep(0,6), buildSu); suMLE
buildSu(suMLE$par)[c("V","W")]
StructTS(NelPlo[,1], type="level") ## compare with W[1,1] and V[1,1]
StructTS(NelPlo[,2], type="level") ## compare with W[2,2] and V[2,2]

## multivariate local level model with homogeneity restriction
buildHo <- function(x) {
  Vsd <- exp(x[1:2])
  Vcorr <- tanh(x[3])
  V <- Vsd %o% Vsd
  V[1,2] <- V[2,1] <- V[1,2] * Vcorr
  return(list(
              m0 = rep(0,2),
              C0 = 1e7 * diag(2),
              FF = diag(2),
              GG = diag(2),
              V = V,
              W = x[4]^2 * V))
}

hoMLE <- dlmMLE(NelPlo, rep(0,4), buildHo); hoMLE
buildHo(hoMLE$par)[c("V","W")]
```

---

dlmModARMA *Create a DLM representation of an ARMA process*

Description

The function creates an object of class dlm representing a specified univariate or multivariate ARMA process

Usage

```
dlmModARMA(ar = NULL, ma = NULL, sigma2 = 1, dV, m0, C0)
```

Arguments

ar	a vector or a list of matrices (in the multivariate case) containing the autoregressive coefficients.
ma	a vector or a list of matrices (in the multivariate case) containing the moving average coefficients.
sigma2	the variance (or variance matrix) of the innovations.
dV	the variance, or the diagonal elements of the variance matrix in the multivariate case, of the observation noise. V is assumed to be diagonal and it defaults to zero.
m0	m_0, the expected value of the pre-sample state vector.
C0	C_0, the variance matrix of the pre-sample state vector.

Details

The returned DLM only gives one of the many possible representations of an ARMA process.

Value

The function returns an object of class dlm representing the ARMA model specified by ar, ma, and sigma2.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
Durbin and Koopman, Time series analysis by state space methods, Oxford University Press, 2001.

See Also

dlmModPoly, dlmModSeas, dlmModReg

Examples

```
## ARMA(2,3)
dlmModARMA(ar = c(.5,.1), ma = c(.4,2,.3), sigma2=1)
## Bivariate ARMA(2,1)
dlmModARMA(ar = list(matrix(1:4,2,2), matrix(101:104,2,2)),
           ma = list(matrix(-4:-1,2,2)), sigma2 = diag(2))
```

`dlmModPoly` *Create an n-th order polynomial DLM*

Description

The function creates an nth order polynomial DLM.

Usage

```
dlmModPoly(order = 2, dV = 1, dW = c(rep(0, order - 1), 1),
           m0 = rep(0, order), C0 = 1e+07 * diag(nrow = order))
```

Arguments

`order`	order of the polynomial model. The default corresponds to a stochastic linear trend.
`dV`	variance of the observation noise.
`dW`	diagonal elements of the variance matrix of the system noise.
`m0`	m_0, the expected value of the pre-sample state vector.
`C0`	C_0, the variance matrix of the pre-sample state vector.

Value

An object of class dlm representing the required n-th order polynomial model.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer, 1997.

See Also

`dlmModARMA`, `dlmModReg`, `dlmModSeas`

Examples

```
## the default
dlmModPoly()
## random walk plus noise
dlmModPoly(1, dV = .3, dW = .01)
```

---

dlmModReg *Create a DLM representation of a regression model*

Description

The function creates a dlm representation of a linear regression model.

Usage

```
dlmModReg(X, addInt = TRUE, dV = 1, dW = rep(0, NCOL(X) + addInt),
          m0 = rep(0, length(dW)),
          C0 = 1e+07 * diag(nrow = length(dW)))
```

Arguments

X	the design matrix
addInt	logical: should an intercept be added?
dV	variance of the observation noise.
dW	diagonal elements of the variance matrix of the system noise.
m0	m_0, the expected value of the pre-sample state vector.
C0	C_0, the variance matrix of the pre-sample state vector.

Details

By setting dW equal to a nonzero vector one obtains a DLM representation of a dynamic regression model. The default value zero of dW corresponds to standard linear regression. Only univariate regression is currently covered.

Value

An object of class dlm representing the specified regression model.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer, 1997.

See Also

dlmModARMA, dlmModPoly, dlmModSeas

Examples

```
x <- matrix(runif(6,4,10), nc = 2); x
dlmModReg(x)
dlmModReg(x, addInt = FALSE)
```

---

dlmModSeas *Create a DLM for seasonal factors*

Description

The function creates a DLM representation of seasonal component.

Usage

```
dlmModSeas(frequency, dV = 1, dW = c(1, rep(0, frequency - 2)),
           m0 = rep(0, frequency - 1),
           C0 = 1e+07 * diag(nrow = frequency - 1))
```

Arguments

frequency	how many seasons?
dV	variance of the observation noise.
dW	diagonal elements of the variance matrix of the system noise.
m0	m_0, the expected value of the pre-sample state vector.
C0	C_0, the variance matrix of the pre-sample state vector.

Value

An object of class dlm representing a seasonal factor for a process with frequency seasons.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
Harvey, Forecasting, structural time series models and the Kalman filter, Cambridge University Press, 1989.

See Also

dlmModARMA, dlmModPoly, dlmModReg, and dlmModTrig for the Fourier representation of a seasonal component.

Examples

```
## seasonal component for quarterly data
dlmModSeas(4, dV = 3.2)
```

---

dlmModTrig *Create Fourier representation of a periodic DLM component*

Description

The function creates a dlm representing a specified periodic component.

Usage

```
dlmModTrig(s, q, om, tau, dV = 1, dW = 0, m0, C0)
```

Arguments

s	the period, if integer.
q	number of harmonics in the DLM.
om	the frequency.
tau	the period, if not an integer.
dV	variance of the observation noise.
dW	a single number expressing the variance of the system noise.
m0	m_0, the expected value of the pre-sample state vector.
C0	C_0, the variance matrix of the pre-sample state vector.

Details

The periodic component is specified by one and only one of s, om, and tau. When s is given, the function assumes that the period is an integer, while a period specified by tau is assumed to be noninteger. Instead of tau, the frequency om can be specified. The argument q specifies the number of harmonics to include in the model. When tau or omega is given, then q is required as well, since in this case the implied Fourier representation has infinitely many harmonics. On the other hand, if s is given, q defaults to all the harmonics in the Fourier representation, that is floor(s/2).

The system variance of the resulting dlm is dW times the identity matrix of the appropriate dimension.

Value

An object of class dlm, representing a periodic component.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer (1997).

See Also

dlmModSeas, dlmModARMA, dlmModPoly, dlmModReg

Examples

```
dlmModTrig(s = 3)
dlmModTrig(tau = 3, q = 1) # same thing
dlmModTrig(s = 4) # for quarterly data
dlmModTrig(s = 4, q = 1)
dlmModTrig(tau = 4, q = 2) # a bad idea!
m1 <- dlmModTrig(tau = 6.3, q = 2); m1
m2 <- dlmModTrig(om = 2 * pi / 6.3, q = 2)
all.equal(unlist(m1), unlist(m2))
```

---

dlmRandom *Random DLM*

Description

Generate a random (constant or time-varying) object of class "dlm", along with states and observations from it.

Usage

```
dlmRandom(m, p, nobs = 0, JFF, JV, JGG, JW)
```

Arguments

m	dimension of the observation vector.
p	dimension of the state vector.
nobs	number of states and observations to simulate from the model.
JFF	should the model have a time-varying FF component?
JV	should the model have a time-varying V component?
JGG	should the model have a time-varying GG component?
JW	should the model have a time-varying W component?

Details

The function generates randomly the system and observation matrices and the variances of a DLM having the specified state and observation dimension. The system matrix GG is guaranteed to have eigenvalues strictly less than one, which implies that a constant DLM is asymptotically stationary. The default behavior is to generate a constant DLM. If JFF is TRUE then a model for nobs observations in which all the elements of FF are time-varying is generated. Similarly with JV, JGG, and JW.

Value

The function returns a list with the following components.

mod	An object of class "dlm".
theta	Matrix of simulated state vectors from the model.
y	Matrix of simulated observations from the model.

If nobs is zero, only the mod component is returned.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Anderson and Moore, Optimal filtering, Prentice-Hall (1979)

See Also

dlm

Examples

```
dlmRandom(1, 3, 5)
```

---

dlmSmooth *DLM smoothing*

Description

The function apply Kalman smoother to compute smoothed values of the state vectors, together with their variance/covariance matrices.

Usage

```
dlmSmooth(y, ...)
## Default S3 method:
dlmSmooth(y, mod, ...)
## S3 method for class 'dlmFiltered'
dlmSmooth(y, ..., debug = FALSE)
```

Arguments

y	an object used to select a method.
...	futher arguments passed to or from other methods.
mod	an object of class "dlm".
debug	if debug=FALSE, faster C code will be used, otherwise all the computations will be performed in R.

Details

The default method returns means and variances of the smoothing distribution for a data vector (or matrix) y and a model mod.

dlmSmooth.dlmFiltered produces the same output based on a dlmFiltered object, typically one produced by a call to dlmFilter.

The calculations are based on the singular value decomposition (SVD) of the relevant matrices. Variance matrices are returned in terms of their SVD.

Value

A list with components

s	Time series (or matrix) of smoothed values of the state vectors. The series starts one time unit before the first observation.
U.S	See below.
D.S	Together with U.S, it gives the SVD of the variances of the smoothing errors.

Warning

The observation variance V in mod must be nonsingular.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Zhang, Y. and Li, X.R., Fixed-interval smoothing algorithm based on singular value decomposition, *Proceedings of the 1996 IEEE International Conference on Control Applications*.
Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).

See Also

See dlm for a description of dlm objects, dlmSvd2var to obtain a variance matrix from its SVD, dlmFilter for Kalman filtering, dlmMLE for maximum likelihood estimation, and dlmBSample for drawing from the posterior distribution of the state vectors.

Examples

```
s <- dlmSmooth(Nile, dlmModPoly(1, dV = 15100, dW = 1470))
plot(Nile, type ='o')
lines(dropFirst(s$s), col = "red")

## Multivariate
set.seed(2)
tmp <- dlmRandom(3, 5, 20)
obs <- tmp$y
m <- tmp$mod
rm(tmp)

f <- dlmFilter(obs, m)
s <- dlmSmooth(f)
all.equal(s, dlmSmooth(obs, m))
```

`dlmSum` *Outer sum of Dynamic Linear Models*

Description

`dlmSum` creates a unique DLM out of two or more independent DLMs. `%+%` is an alias for `dlmSum`.

Usage

```
dlmSum(...)
x %+% y
```

Arguments

`...`	any number of objects of class `dlm`, or a list of such objects.
`x, y`	objects of class `dlm`.

Value

An object of class `dlm`, representing the outer sum of the arguments.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).

Examples

```
m1 <- dlmModPoly(2)
m2 <- dlmModPoly(1)
dlmSum(m1, m2)
m1 %+% m2 # same thing
```

`dlmSvd2var` *Compute a nonnegative definite matrix from its Singular Value Decomposition*

Description

The function computes a nonnegative definite matrix from its Singular Value Decomposition.

Usage

```
dlmSvd2var(u, d)
```

Arguments

`u`	a square matrix, or a list of square matrices for a vectorized usage.
`d`	a vector, or a matrix for a vectorized usage.

Details

The SVD of a nonnegative definite n by n square matrix x can be written as ud^2u', where u is an n by n orthogonal matrix and d is a diagonal matrix. For a single matrix, the function returns just ud^2u'. Note that the argument `d` is a vector containing the diagonal elements of d. For a vectorized usage, `u` is a list of square matrices, and `d` is a matrix. The returned value in this case is a list of matrices, with the element i being `u[[i]] %*% diag(d[i,]^2) %*% t(u[[i]])`.

Value

The function returns a nonnegative definite matrix, reconstructed from its SVD, or a list of such matrices (see details above).

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Horn and Johnson, Matrix analysis, Cambridge University Press (1985)

Examples

```
x <- matrix(rnorm(16),4,4)
x <- crossprod(x)
tmp <- La.svd(x)
all.equal(dlmSvd2var(tmp$u, sqrt(tmp$d)), x)
## Vectorized usage
x <- dlmFilter(Nile, dlmModPoly(1, dV=15099, dW=1469))
x$se <- sqrt(unlist(dlmSvd2var(x$U.C, x$D.C)))
## Level with 50% probability interval
plot(Nile, lty=2)
lines(dropFirst(x$m), col="blue")
lines(dropFirst(x$m - .67*x$se), lty=3, col="blue")
lines(dropFirst(x$m + .67*x$se), lty=3, col="blue")
```

---

dropFirst *Drop the first element of a vector or matrix*

Description

A utility function, `dropFirst` drops the first element of a vector or matrix, retaining the correct time series attributes, in case the argument is a time series object.

Usage

```
dropFirst(x)
```

Arguments

x	a vector or matrix.

Value

The function returns `x[-1]` or `x[-1,]`, if the argument is a matrix. For an argument of class `ts` the class is preserved, together with the correct `tsp` attribute.

Author(s)

Giovanni Petris <GPetris@uark.edu>

Examples

```
(pres <- dropFirst(presidents))
start(presidents)
start(pres)
```

FF *Components of a dlm object*

Description

Functions to get or set specific components of an object of class dlm

Usage

```
## S3 method for class 'dlm'
FF(x)
## S3 replacement method for class 'dlm'
FF(x) <- value
## S3 method for class 'dlm'
V(x)
## S3 replacement method for class 'dlm'
V(x) <- value
## S3 method for class 'dlm'
GG(x)
## S3 replacement method for class 'dlm'
GG(x) <- value
## S3 method for class 'dlm'
W(x)
## S3 replacement method for class 'dlm'
W(x) <- value
## S3 method for class 'dlm'
m0(x)
## S3 replacement method for class 'dlm'
m0(x) <- value
## S3 method for class 'dlm'
C0(x)
## S3 replacement method for class 'dlm'
C0(x) <- value
## S3 method for class 'dlm'
JFF(x)
## S3 replacement method for class 'dlm'
JFF(x) <- value
## S3 method for class 'dlm'
JV(x)
## S3 replacement method for class 'dlm'
JV(x) <- value
## S3 method for class 'dlm'
JGG(x)
## S3 replacement method for class 'dlm'
JGG(x) <- value
## S3 method for class 'dlm'
JW(x)
```

```
## S3 replacement method for class 'dlm'
JW(x) <- value
## S3 method for class 'dlm'
X(x)
## S3 replacement method for class 'dlm'
X(x) <- value
```

Arguments

x	an object of class dlm.
value	a numeric matrix (or vector for m0).

Details

Missing or infinite values are not allowed in value. The dimension of value must match the dimension of the current value of the specific component in x

Value

For the assignment forms, the updated dlm object.

For the other forms, the specific component of x.

Author(s)

Giovanni Petris <GPetris@uark.edu>

See Also

dlm

Examples

```
set.seed(222)
mod <- dlmRandom(5, 6)
all.equal( FF(mod), mod$FF )
all.equal( V(mod), mod$V )
all.equal( GG(mod), mod$GG )
all.equal( W(mod), mod$W )
all.equal( m0(mod), mod$m0 )
all.equal( C0(mod), mod$C0)
m0(mod)
m0(mod) <- rnorm(6)
C0(mod)
C0(mod) <- rwishart(10, 6)
### A time-varying model
mod <- dlmModReg(matrix(rnorm(10), 5, 2))
JFF(mod)
X(mod)
```

mcmc *Utility functions for MCMC output analysis*

Description

Returns the mean, the standard deviation of the mean, and a sequence of partial means of the input vector or matrix.

Usage

```
mcmcMean(x, sd = TRUE)
mcmcMeans(x, sd = TRUE)
mcmcSD(x)
ergMean(x, m = 1)
```

Arguments

x	vector or matrix containing the output of a Markov chain Monte Carlo simulation.
sd	logical: should an estimate of the Monte Carlo standard deviation be reported?
m	ergodic means are computed for `i` in `m:NROW(x)`

Details

The argument `x` is typically the output from a simulation. If a matrix, rows are considered consecutive simulations of a target vector. In this case means, standard deviations, and ergodic means are returned for each column. The standard deviation of the mean is estimated using Sokal's method (see the reference). `mcmcMeans` is an alias for `mcmcMean`.

Value

`mcmcMean` returns the sample mean of a vector containing the output of an MCMC sampler, together with an estimated standard error. If the input is a matrix, means and standard errors are computed for each column.
`mcmcSD` returns an estimate of the standard deviation of the mean for the output of an MCMC sampler.
`ergMean` returns a vector of running ergodic means.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

P. Green (2001). A Primer on Markov Chain Monte Carlo. In *Complex Stochastic Systems*, (Barndorff-Nielsen, Cox and Kl\"uppelberg, eds.). Chapman and Hall/CRC.

Examples

```
x <- matrix(rexp(1000), nc=4)
dimnames(x) <- list(NULL, LETTERS[1:NCOL(x)])
mcmcSD(x)
mcmcMean(x)
em <- ergMean(x, m = 51)
plot(ts(em, start=51), xlab="Iteration", main="Ergodic means")
```

NelPlo *Nelson-Plosser macroeconomic time series*

Description

A subset of Nelson-Plosser data.

Usage

```
data(NelPlo)
```

Format

The format is: mts [1:43, 1:2] -4.39 3.12 1.08 -1.50 3.91 ... - attr(*, "tsp")= num [1:3] 1946 1988 1 - attr(*, "class")= chr [1:2] "mts" "ts" - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:2] "ip" "stock.prices"

Details

The series are `100*diff(log())` of industrial production and stock prices (S&P500) from 1946 to 1988.

Source

The complete data set is available in package `tseries`.

Examples

```
data(NelPlo)
plot(NelPlo)
```

`residuals.dlmFiltered` *One-step forecast errors*

Description

The function computes one-step forecast errors for a filtered dynamic linear model.

Usage

```
## S3 method for class 'dlmFiltered'
residuals(object, ..., type = c("standardized", "raw"), sd = TRUE)
```

Arguments

`object`	an object of class `"dlmFiltered"`, such as the output from `dlmFilter`
`...`	unused additional arguments.
`type`	should standardized or raw forecast errors be produced?
`sd`	when `sd = TRUE`, standard deviations are returned as well.

Value

A vector or matrix (in the multivariate case) of one-step forecast errors, standardized if `type = "standardized"`. Time series attributes of the original observation vector (matrix) are retained by the one-step forecast errors.

If `sd = TRUE` then the returned value is a list with the one-step forecast errors in component `res` and the corresponding standard deviations in component `sd`.

Note

The `object` argument must include a component `y` containing the data. This component will not be present if `object` was obtained by calling `dlmFilter` with `simplify = TRUE`.

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Giovanni Petris (2010), An R Package for Dynamic Linear Models. Journal of Statistical Software, 36(12), 1-16. https://www.jstatsoft.org/v36/i12/.
Petris, Petrone, and Campagnoli, Dynamic Linear Models with R, Springer (2009).
West and Harrison, Bayesian forecasting and dynamic models (2nd ed.), Springer (1997).

See Also

`dlmFilter`

Examples

```
## diagnostic plots
nileMod <- dlmModPoly(1, dV = 15100, dW = 1468)
nileFilt <- dlmFilter(Nile, nileMod)
res <- residuals(nileFilt, sd=FALSE)
qqnorm(res)
tsdiag(nileFilt)
```

`rwishart` *Random Wishart matrix*

Description

Generate a draw from a Wishart distribution.

Usage

```
rwishart(df, p = nrow(SqrtSigma), Sigma, SqrtSigma = diag(p))
```

Arguments

`df`	degrees of freedom. It has to be integer.
`p`	dimension of the matrix to simulate.
`Sigma`	the matrix parameter Sigma of the Wishart distribution.
`SqrtSigma`	a *square root* of the matrix parameter Sigma of the Wishart distribution. Sigma must be equal to `crossprod(SqrtSigma)`.

Details

The Wishart is a distribution on the set of nonnegative definite symmetric matrices. Its density is

$$p(W) = \frac{c|W|^{(n-p-1)/2}}{|\Sigma|^{n/2}} \exp\left\{-\frac{1}{2}\mathrm{tr}(\Sigma^{-1}W)\right\}$$

where n is the degrees of freedom parameter `df` and c is a normalizing constant. The mean of the Wishart distribution is $n\Sigma$ and the variance of an entry is

$$\mathrm{Var}(W_{ij}) = n(\Sigma_{ij}^2 + \Sigma_{ii}\Sigma_{jj})$$

The matrix parameter, which should be a positive definite symmetric matrix, can be specified via either the argument Sigma or SqrtSigma. If Sigma is specified, then SqrtSigma is ignored. No checks are made for symmetry and positive definiteness of Sigma.

Value

The function returns one draw from the Wishart distribution with `df` degrees of freedom and matrix parameter `Sigma` or `crossprod(SqrtSigma)`

Warning

The function only works for an integer number of degrees of freedom.

Note

From a suggestion by B.Venables, posted on S-news

Author(s)

Giovanni Petris <GPetris@uark.edu>

References

Press (1982). Applied multivariate analysis.

Examples

```
rwishart(25, p = 3)
a <- matrix(rnorm(9), 3)
rwishart(30, SqrtSigma = a)
b <- crossprod(a)
rwishart(30, Sigma = b)
```

USecon *US macroeconomic time series*

Description

US macroeconomic data.

Usage

```
data(USecon)
```

Format

The format is: mts [1:40, 1:2] 0.1364 0.0778 -0.3117 -0.5478 -1.2636 ... - attr(*, "dimnames")=List of 2 ..$: NULL ..$: chr [1:2] "M1" "GNP" - attr(*, "tsp")= num [1:3] 1978 1988 4 - attr(*, "class")= chr [1:2] "mts" "ts"

Details

The series are `100*diff(log())` of seasonally adjusted real U.S. money 'M1' and GNP from 1978 to 1987.

Source

The complete data set is available in package `tseries`.

Examples

```
data(USecon)
plot(USecon)
```

Index

∗ **array**
dlmSvd2var, 27
∗ **datagen**
dlmRandom, 23
∗ **datasets**
NelPlo, 32
USecon, 35
∗ **distribution**
arms, 2
rwishart, 34
∗ **misc**
arms, 2
ARtransPars, 5
bdiag, 6
convex.bounds, 7
dlm, 8
dlmBSample, 9
dlmFilter, 10
dlmForecast, 12
dlmGibbsDIG, 13
dlmLL, 15
dlmMLE, 16
dlmModARMA, 17
dlmModPoly, 19
dlmModReg, 20
dlmModSeas, 21
dlmModTrig, 22
dlmRandom, 23
dlmSmooth, 24
dlmSum, 26
dlmSvd2var, 27
dropFirst, 28
mcmc, 31
residuals.dlmFiltered, 33
∗ **multivariate**
arms, 2
∗ **smooth**
dlmSmooth, 24
∗ **ts**
dlmFilter, 10
dlmSmooth, 24
dlmSum, 26
dropFirst, 28
FF, 29
%+% *(*dlmSum*)*, 26

arms, 2, *7*
ARtransPars, 5
as.dlm *(*dlm*)*, 8

bdiag, 6

C0 *(*FF*)*, 29
C0<- *(*FF*)*, 29
convex.bounds, 7

dlm, 8, *11*, *17*, *24*, *25*, *30*
dlmBSample, 9, *11*, *25*
dlmFilter, 10, *10*, *15*, *25*, *33*
dlmForecast, 12
dlmGibbsDIG, 13
dlmLL, 15, *17*
dlmMLE, *11*, *15*, 16, *25*
dlmModARMA, *9*, 17, *19*, *21–23*
dlmModPoly, *9*, *18*, 19, *21–23*
dlmModReg, *9*, *18*, *19*, 20, *22*, *23*
dlmModSeas, *9*, *18*, *19*, 21, *21*, *23*
dlmModTrig, 22, *22*
dlmRandom, 23
dlmSmooth, *11*, 24
dlmSum, 26
dlmSvd2var, *11*, *25*, 27
dropFirst, 28

ergMean *(*mcmc*)*, 31

FF, 29
FF<- *(*FF*)*, 29

GG *(*FF*)*, 29

GG<- (FF), 29

is.dlm (dlm), 8

JFF (FF), 29
JFF<- (FF), 29
JGG (FF), 29
JGG<- (FF), 29
JV (FF), 29
JV<- (FF), 29
JW (FF), 29
JW<- (FF), 29

m0 (FF), 29
m0<- (FF), 29
mcmc, 31
mcmcMean (mcmc), 31
mcmcMeans (mcmc), 31
mcmcSD (mcmc), 31

NelPlo, 32

residuals.dlmFiltered, 33
rwishart, 34

USecon, 35

V (FF), 29
V<- (FF), 29

W (FF), 29
W<- (FF), 29

X (FF), 29
X<- (FF), 29