

Package ‘mar1s’

June 16, 2009

Type Package

Title Multiplicative AR(1) with Seasonal Processes

Version 2.0-1

Date 2009-06-12

Author Andrey Paramonov

Maintainer Andrey Paramonov <cmr.Pent@gmail.com>

Depends R (>= 2.8.1), cmrutils (>= 1.1-1), fda

Description Multiplicative AR(1) with Seasonal is a stochastic process model built on top of AR(1). The package provides the following procedures for MAR(1)S processes: fit, compose, decompose, advanced simulate and predict.

License GPL (>= 3)

URL <http://aparamon.msk.ru/svn/study/R-packages/mar1s/>

Repository CRAN

Date/Publication 2009-06-16 06:40:23

R topics documented:

mar1s-package	2
compose.ar1	2
compose.mar1s	4
fit.mar1s	6
forest.fire	7
nesterov.index	8
seasonal.ave	9
seasonal.smooth	10
sim.mar1s	11

Index	14
--------------	-----------

 marls-package

Multiplicative AR(1) with Seasonal Processes

Description

Multiplicative AR(1) with Seasonal is a stochastic process model built on top of AR(1). The package provides the following procedures for MAR(1)S processes: fit, compose, decompose, advanced simulate and predict.

Details

Package: marls
 Type: Package
 Version: 2.0-1
 Date: 2008-06-12
 License: GPL (>= 3)

Author(s)

Andrey Paramonov <cmr.Pent@gmail.com>

 compose.ar1

Compose and Decompose AR(1) Process

Description

`compose.ar1` composes AR(1) process realization by given vector(s) of innovations.

`decompose.ar1` extracts AR(1) process residuals from time series.

Usage

```
compose.ar1(arcoef, innov, init = 0, xregcoef = 0, xreg = NULL,
            init.xreg = rep(0, length(xregcoef)))
```

```
decompose.ar1(arcoef, data, init = NA, xregcoef = 0, xreg = NULL,
              init.xreg = rep(NA, length(xregcoef)))
```

Arguments

`arcoef` A number specifying autoregression coefficient.
`innov` A univariate or multivariate time series containing the innovations.
`data` A univariate or multivariate time series containing the process realization(s).

<code>init</code>	A number specifying the value of the process just prior to the start value in <code>innov/data</code> .
<code>xregcoef</code>	A vector specifying coefficients for the external regressors.
<code>xreg</code>	A matrix-like object of the same row count as <code>innov/data</code> , specifying the values of external regressors. The default <code>NULL</code> means zeroes.
<code>init.xreg</code>	A vector specifying the values of external regressors just prior to the start values in <code>xreg</code> . The default <code>NULL</code> means zeroes.

Details

Here *AR(1) process with external regressors* is a linear regression with AR(1) model for the error term:

$$y_t = b_1 x_{t,1} + \dots + b_k x_{t,k} + z_t$$

$$z_t = a z_{t-1} + e_t$$

Use `xreg = NULL` for the regular AR(1) process.

Value

An object of the same type and dimensions as `innov/data` (typically time series).

See Also

[arima](#) for more general *ARMA(p, q)* processes.

Examples

```
## Simple
e <- ts(c(0, 1, 0, 1, 0), freq = 12)
compose.ar1(0.1, e)
compose.ar1(0.1, e, 1)

x <- ts(c(0, 1, 0, 1, 0), freq = 12)
decompose.ar1(0.1, x)
decompose.ar1(0.1, x, 1)

## Multiseries
compose.ar1(0.1, ts(cbind(0, 1)))
compose.ar1(0.1, ts(cbind(c(0, 1, 0), c(1, 0, 1))))

decompose.ar1(0.1, ts(cbind(0, 1)))
decompose.ar1(0.1, ts(cbind(c(0, 1, 0), c(1, 0, 1))))

## External regressors
xreg1 <- rep(2, 5)
xreg2 <- matrix(rep(c(2, 1), each = 5), 5, 2)
e <- ts(c(0, 1, 0, 1, 0), freq = 12)
```

```

compose.ar1(0.1, e, xregcoef = 0.5, xreg = xreg1)
compose.ar1(0.1, e, xregcoef = 0.5, init = 0, xreg = xreg1, init.xreg = -2)
compose.ar1(0.1, e, xregcoef = c(1, -1), xreg = xreg2)

x <- ts(c(0, 1, 0, 1, 0), freq = 12)
decompose.ar1(0.1, x, xregcoef = 0.5, xreg = xreg1)
decompose.ar1(0.1, x, xregcoef = 0.5, init = 0, xreg = xreg1, init.xreg = -2)
decompose.ar1(0.1, x, xregcoef = c(1, -1), xreg = xreg2)

## Back-test
a <- 0.5
innov <- ts(rnorm(10), frequency = 12)
init <- 1
xrcoef <- seq(-0.1, 0.1, length.out = 3)
xreg <- matrix(1:30, 10, 3)
init.xreg <- 1:3
x <- compose.ar1(a, innov, init, xrcoef, xreg, init.xreg)
r <- decompose.ar1(a, x, init, xrcoef, xreg, init.xreg)
stopifnot(all.equal(innov, r))

```

compose.mar1s

Compose and Decompose MAR(1)S Process

Description

compose.mar1s composes MAR(1)S process realization by given vector of log-innovations.
 decompose.mar1s extracts MAR(1)S process components from time series.

Usage

```

compose.mar1s(object, loginnov, start.time = head(time(loginnov), 1),
              xreg.absdata = NULL, init.absdata = NULL)

decompose.mar1s(object, absdata, start.time = head(time(absdata), 1),
                init.absdata = rep(NA, NCOL(absdata)))

```

Arguments

object	An object of class "mar1s" specifying the model parameters.
loginnov	A univariate time series containing the log-innovations.
absdata	A univariate or multivariate time series containing the process realization.
start.time	The sampling time for the first simulation step.
xreg.absdata	A matrix-like object with row count = n.ahead, specifying the values for the external regressors. If NULL, default values are used.
init.absdata	A vector specifying the initial values of the process. If NULL, default values are used.

Details

Multiplicative AR(1) with Seasonal (MAR(1)S) process is defined as

$$x_t = \exp(s_t + y_t)$$

$$y_{t,1} = b_1 y_{t,2} + \dots + b_k y_{t,k+1} + z_t$$

$$z_t = a z_{t-1} + e_t$$

where

s_t is the log-seasonal component,

y_t is the AR(1) (log-stochastic) component,

e_t is the log-residuals (random component).

Value

absdata	Realization of the process (a univariate or multivariate time series object).
logdata	Logarithm of absdata (a univariate or multivariate time series object).
logstoch	Log-stochastic component (a univariate or multivariate time series object).
logresid	Random component (a univariate time series object).

Note

`decompose.mar1s` and `fit.mar1s` compute the random component in different ways: `decompose.mar1s` uses `filter` while `fit.mar1s` saves the residuals returned by `arima`. The results may be different in:

the first value: `decompose.mar1s` uses specified `init.absdata` while `arima` always assumes zero initial values for the fitted process;

non-finite values: `decompose.mar1s` handles non-finite values more accurately.

See Also

[compose.ar1](#) for the AR(1) with external regressors processes, [fit.mar1s](#) for fitting MAR(1)S process to data, [sim.mar1s](#) for MAR(1)S process simulation and prediction.

Examples

```
data(forest.fire, package = "mar1s")
data(nesterov.index, package = "mar1s")

## Simple
mar1s <- fit.mar1s(window(forest.fire, 1969, 1989))

x <- ts(rnorm(365, sd = mar1s$logresid.sd), start = c(1989, 1))
```

```

plot(compose.mar1s(mar1s, x)$absdata)

decomp <- decompose.mar1s(mar1s, mar1s$decomposed$absdata)
delta <- abs(nan2na(mar1s$decomposed$logresid) -
             nan2na(decomp$logresid))
stopifnot(all(na.exclude(tail(delta, -1)) < 1E-6))

## External regressors
mar1s <- fit.mar1s(window(forest.fire, 1969, 1989),
                  window(nesterov.index[, "mean"], 1969, 1989))

x <- rnorm(365, sd = mar1s$logresid.sd)
xreg <- window(nesterov.index[, "mean"], 1989.001, 1990)
plot(compose.mar1s(mar1s, x, c(1989, 1), xreg)$absdata)

decomp <- decompose.mar1s(mar1s, mar1s$decomposed$absdata)
delta <- abs(mar1s$decomposed$logstoch - decomp$logstoch)
stopifnot(all(na.exclude(delta) < 1E-6))
delta <- abs(nan2na(mar1s$decomposed$logresid) -
             nan2na(decomp$logresid))
stopifnot(all(na.exclude(tail(delta, -1)) < 1E-6))

```

fit.mar1s

Fit Multiplicative AR(1) with Seasonal Process to Data

Description

Fits Multiplicative AR(1) with Seasonal process model to time series.

Usage

```
fit.mar1s(x, xreg = NULL, seasonal.fun = seasonal.smooth, ...)
```

Arguments

<code>x</code>	A univariate time series.
<code>xreg</code>	A univariate or multivariate time series of external regressors, or NULL.
<code>seasonal.fun</code>	A function which takes a univariate time series as its first argument and returns the estimated seasonal component.
<code>...</code>	Additional arguments passed to <code>seasonal.fun</code> .

Value

An object of class "mar1s" with the following components:

<code>logseasonal</code>	Estimated log-seasonal figure (a univariate or multivariate time series object).
<code>logstoch.ar1</code>	AR(1) with external regressors model fitted for the log-stochastic component (an object of class "Arima").
<code>logresid.sd</code>	Standard deviation of the residuals.
<code>decomposed</code>	An object of class "mar1s.ts" containing decomposed time series (see compose.mar1s).

See Also

`compose.marls` for MAR(1)S process formal definition and composition/decomposition functions, `seasonal.ave`, `seasonal.smooth` for seasonal component extraction functions, `sim.marls` for MAR(1)S process simulation and prediction.

Examples

```
data(forest.fire, package = "marls")
data(nesterov.index, package = "marls")

## Simple
marls <- fit.marls(forest.fire)
plot(marls$logseasonal)
confint(marls$logstoch.ar1)
marls$logresid.sd
resid <- nan2na(marls$decomposed$logresid)
qqnorm(resid)
qqline(resid)

## External regressors
marls <- fit.marls(forest.fire, nesterov.index[, "mean"])
plot(cbind(marls$logseasonal, marls$logseasonal.r))
confint(marls$logstoch.ar1)
resid <- nan2na(marls$decomposed$logresid)
qqnorm(resid)
qqline(resid)
```

forest.fire

Forest fire in Irkutsk region, USSR: historical data

Description

Number of forest fire seats in Irkutsk region, USSR. Daily from April 01 to October 31, 1969–1991 (total 4708 observations).

Usage

```
data(forest.fire)
```

Format

A univariate time series.

Source

AviaLesoOkhrana (<http://www.nffc.aviales.ru/engl/main.sht>).

Examples

```
data(forest.fire)
colnames(forest.fire)
plot(forest.fire)
```

nesterov.index *Nesterov index in Irkutsk region, USSR: historical data*

Description

Values of Nesterov index in 6 districts of Irkutsk region, USSR + region averaged. Daily from April 01 to October 31, 1969–1991 (total 4708 observations).

Usage

```
data(nesterov.index)
```

Format

A multivariate time series.

Details

The Nesterov index is the official Russian fire-danger rating specified by standard GOST R 22.1.09–99. It is calculated using the ignition index, the temperature, the dew-point temperature and the number of days since last significant ($\geq 3mm$) precipitation.

The dataset consists of daily values 1969–1991.

Source

Russian Institute of Hydrometeorological Information–World Data Center (<http://meteo.ru/english/>).

Examples

```
data(nesterov.index)
colnames(nesterov.index)
plot(nesterov.index)
```

seasonal.ave	<i>Averaged Seasonal Component of Time Series</i>
--------------	---

Description

Extracts seasonal component of time series by averaging observations on the same position in the cycle.

Usage

```
seasonal.ave(x, ave.FUN = mean, ...)
```

Arguments

x	A univariate time series.
ave.FUN	Averaging function.
...	Additional arguments passed to ave.FUN.

Value

A time series object with times from 0 to 1 and the same frequency as x.

See Also

[ave](#), [seasonal.smooth](#) for alternative seasonal extraction method.

Examples

```
set.seed(19860306)

## Artificial example
x <- ts(sin(2*pi*(3:97)/10) + 0.5*rnorm(length(3:97)),
        start = c(0, 3), frequency = 10)

plot.default(time(x)%%1, x, xlab = "Phase")
lines(seasonal.ave(x), col = "blue")
lines(seasonal.ave(x, median), col = "green")
legend("bottomleft",
       legend = c("Mean averaging",
                  "Median averaging"),
       col = c("blue", "green"),
       lty = "solid")

## Realistic example
data(nesterov.index, package = "marls")
x <- log(nesterov.index[, "mean"])
x[x < -10] <- -Inf

plot.default(time(x)%%1, x, xlab = "Phase", pch = ".")
```

```

lines(seasonal.ave(x), col = "blue")
lines(seasonal.ave(x, median), col = "green")
legend("topleft",
      legend = c("Mean averaging",
                 "Median averaging"),
      col = c("blue", "green"),
      lty = "solid")

```

seasonal.smooth *Smooth Seasonal Component of Time Series*

Description

Extracts seasonal component of time series by fitting the data with a linear combination of smooth periodic functions.

Usage

```
seasonal.smooth(x, basis = create.fourier.basis(nbasis = 3), lambda = 0, ...)
```

Arguments

<code>x</code>	A univariate time series.
<code>basis</code>	A functional basis object (see basisfd). By default, use a linear combination of <code>const</code> , <code>sin((2*pi)*t)</code> and <code>cos((2*pi)*t)</code> .
<code>lambda</code>	A nonnegative number specifying the amount of smoothing. By default, apply no additional smoothing.
<code>...</code>	Not currently used.

Details

Although it is possible to specify arbitrary functional basis object, the function will only work properly if the basis is periodical on the unit interval. It is recommended to use a Fourier basis with default period ([create.fourier.basis](#)).

Positive values of `lambda` imply a restriction on roughness of the result. The more the value, the more smooth result is; see [smooth.basis](#) for more detailed description.

Value

A time series object with times from 0 to 1 and the same frequency as `x`. The smoothing functional data object is stored in attribute `fd`.

See Also

[smooth.basisPar](#), [fd](#) for functional data objects, [seasonal.ave](#) for alternative seasonal extraction method.

Examples

```

set.seed(19860306)

## Artificial example
x <- ts(sin(2*pi*(3:97)/10) + 0.5*rnorm(length(3:97)),
        start = c(0, 3), frequency = 10)

fourier3 <- seasonal.smooth(x)
fourier9 <- seasonal.smooth(x, create.fourier.basis(nbasis = 9))
fourier9s<- seasonal.smooth(x, create.fourier.basis(nbasis = 9), 1E-6)

plot.default(time(x)%%1, x, xlab = "Phase")
points(fourier3, pch = 20, col = "blue")
lines(attr(fourier3, "fd"), col = "blue")
points(fourier9, pch = 20, col = "green")
lines(attr(fourier9, "fd"), col = "green")
points(fourier9s, pch = 20, col = "red")
lines(attr(fourier9s, "fd"), col = "red")
legend("bottomleft",
       legend = c("Fourier-3 basis",
                  "Fourier-9 basis",
                  "Fourier-9 basis, smooth"),
       col = c("blue", "green", "red"),
       lty = "solid")

## Realistic example
data(nesterov.index, package = "mar1s")
x <- log(nesterov.index[, "mean"])
x[x < -10] <- -Inf

fourier3 <- seasonal.smooth(x)
fourier9 <- seasonal.smooth(x, create.fourier.basis(nbasis = 9))
fourier9s<- seasonal.smooth(x, create.fourier.basis(nbasis = 9), 2E-5)

plot.default(time(x)%%1, x, xlab = "Phase", pch = ".")
lines(attr(fourier3, "fd"), col = "blue")
lines(attr(fourier9, "fd"), col = "green")
lines(attr(fourier9s, "fd"), col = "red")
legend("topleft",
       legend = c("Fourier-3 basis",
                  "Fourier-9 basis",
                  "Fourier-9 basis, smooth"),
       col = c("blue", "green", "red"),
       lty = "solid")

```

Description

`sim.marls` simulates from MAR(1)S process.

`predict.marls` is a wrapper around `sim.marls` which estimates confidence intervals for the future values of the MAR(1)S process.

Usage

```
sim.marls(object, n.ahead = 1, n.sim = 1, start.time = 0,
          xreg.absdata = NULL, init.absdata = NULL)
```

```
## S3 method for class 'marls':
predict(object, n.ahead = 1, start.time = 0,
        xreg.absdata = NULL, init.absdata = NULL,
        probs = c(0.05, 0.5, 0.95), n.sim = 1000, ...)
```

Arguments

<code>object</code>	An object of class "marls" specifying the model parameters.
<code>n.ahead</code>	Number of steps ahead at which to simulate/predict.
<code>n.sim</code>	Number of simulations.
<code>start.time</code>	The sampling time for the first simulation step.
<code>xreg.absdata</code>	A matrix-like object with row count = <code>n.ahead</code> , specifying the values for the external regressors. If <code>NULL</code> , default values are used.
<code>init.absdata</code>	A vector specifying the initial values of the process. If <code>NULL</code> , default values are used.
<code>probs</code>	A vector of probabilities.
<code>...</code>	Arguments from previous methods.

Value

For `sim.marls`, a vector of simulated values.

For `predict.marls`, a vector of estimated quantiles.

See Also

[compose.marls](#) for MAR(1)S process formal definition and composition/decomposition functions, [fit.marls](#) for fitting MAR(1)S process to data.

Examples

```
data(forest.fire, package = "marls")
data(nesterov.index, package = "marls")

## Univariate
marls <- fit.marls(forest.fire)

sim.marls(marls)
```

```
sim.marls(marls, n.sim = 6)
sim.marls(marls, n.ahead = 3)

predict(marls)
predict(marls, n.ahead = 10)
predict(marls, init.absdata = 100)

t <- seq(1/12, 11/12, 1/6)
p <- mapply(predict, start.time = t,
            MoreArgs = list(object = marls, probs = c(0.05, 0.95)))
plot(exp(marls$logseasonal), ylim = c(0, max(p)),
     ylab = "Forest fire")
arrows(t, p[1, ], t, p[2, ],
       code = 3, angle = 90, length = 0.05)

## External regressors
marls <- fit.marls(forest.fire, nesterov.index[, "mean"])

sim.marls(marls)
sim.marls(marls, n.sim = 6)

predict(marls)
predict(marls, xreg.absdata = 10000)
predict(marls, init.absdata = c(100, 1000))
```

Index

*Topic **datasets**

forest.fire, 7
nesterov.index, 8

*Topic **models**

compose.ar1, 2
compose.marls, 4
fit.marls, 6
marls-package, 1
sim.marls, 11

*Topic **multivariate**

compose.ar1, 2
compose.marls, 4
fit.marls, 6
marls-package, 1
sim.marls, 11

*Topic **package**

marls-package, 1

*Topic **ts**

compose.ar1, 2
compose.marls, 4
fit.marls, 6
marls-package, 1
seasonal.ave, 8
seasonal.smooth, 10
sim.marls, 11

arima, 3

ave, 9

basisfd, 10

compose.ar1, 2, 5
compose.marls, 4, 6, 12
create.fourier.basis, 10

decompose.ar1 (compose.ar1), 2
decompose.marls (compose.marls), 4

fd, 10

fit.marls, 5, 6, 12

forest.fire, 7

marls (marls-package), 1

marls-package, 1

nesterov.index, 8

predict.marls (sim.marls), 11

seasonal.ave, 6, 8, 10

seasonal.smooth, 6, 9, 10

sim.marls, 5, 6, 11

smooth.basis, 10

smooth.basisPar, 10