Package 'garma'

May 16, 2020

Type Package

Title GARMA provides support for estimating, and fitting Gegenbauer Seasonal/Cyclical time series models.

Version 0.2.0

Maintainer Richard Hunt<maint@huntemail.id.au>

Description This package provides methods for estimating long memory-seasonal/cyclical Gegenbauer univariate time series proceses.
Refer to the vignette for details of fitting these processes.

License GPL-3

Encoding UTF-8

LazyData true

Imports Matrix,
assertthat,
zoo,
forecast,
lubridate,
FKF,
signal,
pracma,
nloptr,
Rsolnp,
ggplot2,
Rdpack (>= 0.7)

Suggests forecast,
BB,
GA,
pso,
dfoptim,
testthat

RoxygenNote 7.0.1

R topics documented:

`forecast.garma_model` *The forecast function predicts future values of a "garma_model" object, and is exactly the same as the "predict" function with slightly different parameter values.*

Description

The forecast function predicts future values of a "garma_model" object, and is exactly the same as the "predict" function with slightly different parameter values.

Usage

```
## S3 method for class 'garma_model'
forecast(mdl, h = 1)
```

Arguments

`mdl`	(garma_model) The garma_model from which to forecast the values.
`h`	(int) The number of time periods to predict ahead. Default: 1

Value

- a "ts" object containing the requested forecasts.

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
forecast(mdl, h=12)
```

garma *garma: A package for estimating and foreasting Gegenbauer time series models.*

Description

The GARMA package provides the main function "garma" as well as print, summary, predict, forecast and plot/ggplot options.

The garma function is the main function for the garma package. Depending on the parameters it will calculate the parameter estimates for the GARMA process, and if available the standard errors (se's) for those parameters.

Usage

```
garma(
  x,
  order = list(0, 0, 0),
  k = 1,
  include.mean = (order[2] == 0),
  method = "Whittle",
  allow_neg_d = TRUE,
  maxeval = 10000,
  opt_method = "solnp",
  m_trunc = 50
)
```

Arguments

x	(num) This should be a numeric vector representing the process to estimate. A minimum length of 96 is required.
order	(list) This should be a list (similar to the stats::arima order parameter) which will give the order of the process to fit. The format should be list(p,d,q) where p, d, and q are all positive integers. p represents the degree of the autoregressive process to fit, q represents the order of the moving average process to fit and d is the (integer) differencing to apply prior to any fitting.
k	(int) This is the number of (multiplicative) Gegenbauer terms to fit. Only 0 or 1 are allowed in this version.
include.mean	(bool) A boolean value indicating whether a mean should be fit. Note if you have any differencing, then it generally does not make sense to fit a mean term. Because of this, the default here is to fit a mean time when d (in the "order" parmaeter) is zero and otherwise not.
method	(character) This defines the estimation method for the routine. The valid values are 'CSS', 'Whittle', 'QML' and 'WLL'. The default (Whittle) will generally return very accurate estimates quite quickly, provided the asumption of a Gaussian distribution is even approximately correct, and is probably the method of choice for most users. For the theory behind this, refer Giraitis et. al. (2001)

'CSS' is a conditional 'sum-of-squares' technique and can be quite slow. Reference: Chung (1996). 'QML' is a Quasi-Maximum-Likelihood technique, and can also be quite slow. Reference Dissanayake (2016). (k>1 is not supported for QML) 'WLL' is a new technique which appears to work well even if the ϵ_t are highly skewed and/or have heavy tails (skewed and/or lepto-kurtic). However the asymptotic theory for the WLL method is not complete and so standard errors are not available for most parameters.

`allow_neg_d` (bool) A boolean value indicating if a negative value is allowed for the fractional differencing component of the Gegenbauer term is allowed. This can be set to FALSE to force the routine to find a positive value.

`maxeval` (int) the maximum function eveluations to be allowed during each optimisation.

`opt_method` (character) This names the optimisation method used to find the parameter estimates. The default is to use 'solnp' from package Rsolnp, which has shown to have good performance in a wide range of circumstances. For some data or some models, however, other methods may work well. Supported algorithms include:

- cobyla algorithm in package nloptr
- directL algorithm in package nloptr
- BBoptim from package BB
- psoptim from package pso
- hjkb from dfoptim package
- nmkb from dfoptim package
- solnp from Rsolnp package
- best - this option evaluates all the above options in turn and picks the one which finds the lowest value of the objective. This can be quite time consuming to run, particularly for the 'CSS' method.

Note further that if you specify a k>1, then inequality constraints are required, and this will further limit the list of supported routines.

`m_trunc` Used for the QML estimation method. This defines the AR-truncation point when evaluating the likelihood function. Refer to Dissanayake et. al. (2016) for details.

Details

The GARMA model is specified as

$$\phi(B)\prod_{i=1}^{k}(1-2u_iB+B^2)^{d_i}(X_t-\mu)=\theta(B)\epsilon_t$$

where

- $\phi(B)$ represents the short-memory Autoregressive component of order p,
- $\theta(B)$ represents the short-memory Moving Average component of order q,
- $(1-2u_iB+B^2)^{d_i}$ represents the long-memory Gegenbauer component (there may in general be k of these),

- X_t represents the observed process,
- ϵ_t represents the random component of the model - these are assumed to be uncorrelated but identically distributed variates. Generally the routines in this package will work best if these have an approximate Gaussian distribution.
- B represents the Backshift operator, defined by $BX_t = X_{t-1}$.

when k=0, then this is just a short memory model as fit by the stats "arima" function.

Value

An S3 object of class "garma_model".

References:
C Chung. A generalized fractionally integrated autoregressive moving-average process. Journal of Time S
G Dissanayake, S Peiris, and T Proietti. State space modelling of Gegenbauer processes with long memory.
L Giraitis, J Hidalgo, and P Robinson. Gaussian estimation of parametric spectral density with unknown p

Author(s)

Richard Hunt

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
print(garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F))
# Compare with the built-in arima function
print(arima(ap,order=c(9,1,0),include.mean=F))
```

---

ggplot.garma_model *The ggplot function generates a ggplot of actuals and predicted values for a "garma_model" object.*

Description

The ggplot function generates a ggplot of actuals and predicted values for a "garma_model" object.

Usage

```
## S3 method for class 'garma_model'
ggplot(mdl, h = 24)
```

Arguments

mdl	(garma_model) The garma_model from which to ggplot the values.
h	(int) The number of time periods to predict ahead. Default: 24
...	other arguments to be passed to the "plot" function.

Value

A ggplot2 "ggplot" object. Note that the standard ggplot2 "+" notation can be used to enhance the default output.

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
ggplot(mdl)
```

`plot.garma_model` *The plot function generates a plot of actuals and predicted values for a "garma_model" object.*

Description

The plot function generates a plot of actuals and predicted values for a "garma_model" object.

Usage

```
## S3 method for class 'garma_model'
plot(mdl, h = 24, ...)
```

Arguments

`mdl`	(garma_model) The garma_model from which to plot the values.
`h`	(int) The number of time periods to predict ahead. Default: 24
`...`	other arguments to be passed to the "plot" function.

Value

An R "plot" object.

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
plot(mdl)
```

predict.garma_model *The predict function predicts future values of a "garma_model" object.*

Description

The predict function predicts future values of a "garma_model" object.

Usage

```
## S3 method for class 'garma_model'
predict(mdl, n.ahead = 1)
```

Arguments

mdl	(garma_model) The garma_model from which to predict the values.
n.ahead	(int) The number of time periods to predict ahead. Default: 1

Value

A "ts" object containing the requested forecasts.

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
predict(mdl, n.ahead=12)
```

print.garma_model *The print function prints a summary of a "garma_model" object.*

Description

The print function prints a summary of a "garma_model" object.

Usage

```
## S3 method for class 'garma_model'
print(mdl, verbose = FALSE)
```

Arguments

mdl	(garma_model) The garma_model from which to print the values.
verbose	(bool) whether to print out the verbose version of the model or not. Default: FALSE

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
print(mdl)
```

`summary.garma_model` *The summary function provides a summary of a "garma_model" object.*

Description

The summary function provides a summary of a "garma_model" object.

Usage

```
## S3 method for class 'garma_model'
summary(mdl, verbose = TRUE)
```

Arguments

`mdl`	(garma_model) The garma_model from which to print the values.
`verbose`	(bool) whether to print out the verbose version of the model or not. Default: TRUE

Examples

```
data(AirPassengers)
ap  <- as.numeric(diff(AirPassengers,12))
mdl <- garma(ap,order=c(9,1,0),k=0,method='CSS',include.mean=F)
summary(mdl)
```

Index