Package 'coconots'

March 29, 2023

Type Package

Title Convolution-Closed Models for Count Time Series

Version 1.1.1

Date 2023-3-9

Description Useful tools for fitting, validating, and forecasting of practical convolution-closed time series models for low counts are provided. Marginal distributions of the data can be modeled via Poisson and Generalized Poisson innovations. Regression effects can be modelled via time varying innovation rates. The models are described in Jung and Tremayne (2011) <doi:10.1111/j.1467-9892.2010.00697.x> and the model assessment tools are presented in Czado et al. (2009) <doi:10.1111/j.1541-0420.2009.01191.x>, Gneiting and Raftery (2007) <doi:10.1198/016214506000001437> and, Tsay (1992) <doi:10.2307/2347612>.

Imports Rcpp, forecast, numDeriv, HMMpa, stats, ggplot2, utils, matrixStats, JuliaConnectoR

LazyData true

LinkingTo Rcpp, StanHeaders (>= 2.21.0), RcppParallel (>= 5.0.1)

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

License MIT + file LICENSE

NeedsCompilation yes

Author Manuel Huth [aut, cre], Robert C. Jung [aut], Andy Tremayne [aut]

Maintainer Manuel Huth <manuel.huth@yahoo.com>

Repository CRAN

Date/Publication 2023-03-29 08:40:02 UTC

R topics documented:

`coconots-package` *Concolution-closed Models for Time Series*

Description

Functions to analyse time series consisting of low counts are provided. The focus in the current version is on practical models that can capture first and higher-order dependence based on the work of Joe (1996). Both equidispersed and overdispersed marginal distributions of data can be modelled. Regression effects can be included. Fast and efficient procedures for likelihood based inference and probabilistic forecasting are provided as well as useful tools for model validation and diagnostics.

Details

The package allows simulation of convolution-closed count time series models with the `cocoSim` function. Model fitting is performed with the `cocoReg` routine. By passing a cocoReg-type object, `cocoForecast` computes the one-step ahead forecasting distribution. `cocoBoot`, `cocoPit`, `cocoScore`, and `cocoResid` provide routines for model assessment. The main usage of the package is illustrated within the cocoReg function chapter. For more details and examples of the functions see the respective sections within this vignette.

By default, our functions make use of an RCPP implementation. However, users with a running Julia installation can choose to call Julia in the background to run their functions by specifiying it in the R function input. This option is particularly useful for the regression (`cocoReg`), where a complex likelihood function must be numerically evaluated to obtain parameter estimates. By leveraging Julia's automatic differentiation capabilities, our functions can take advantage of numerical gradients, leading to increased numerical stability and faster convergence.

As we find both, the Julia and RCPP implementations produce qualitatively similar results in all our tests, we have decided to use the RCPP implementation as the default option to make our package accessible to non-Julia users.

Author(s)

Maintainer: Manuel Huth <manuel.huth@yahoo.com>

References

Czado, C., Gneiting, T. and Held, L. (2009) Predictive model assessment for count data. *Biometrics* **65**, 1254–61.

Gneiting, T. and Raftery, A. E. (2007) Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102:359-378.

R.C. Jung, A.R. Tremayne (2006) Coherent forecasting in integer time series models. *International Journal of Forecasting* **22**, 223–238

Jung, R. C. and Tremayne, A. R. (2011) Convolution-closed models for count time series with applications. *Journal of Time Series Analysis*, **32**, 3, 268–280.

Jung, Robert C., Brendan P. M. McCabe, and Andrew R. Tremayne. (2016). Model validation and diagnostics. *In Handbook of Discrete Valued Time Series*. Edited by Richard A. Davis, Scott H. Holan, Robert Lund and Nalini Ravishanker. Boca Raton: Chapman and Hall, pp. 189–218.

Joe, H. (1996) Time series models with univariate margins in the convolution-closed infinitely divisible class. *Journal of Applied Probability*, 664–677.

Tsay, R. S. (1992) Model checking via parametric bootstraps in time series analysis. *Applied Statistics* **41**, 1–15.

Westgren, A. (1916) Die Veraenderungsgeschwindigkeit der lokalen Teilchenkonzentration in kollioden Systemen (Erste Mitteilung). *Arkiv foer Matematik, Astronomi och Fysik*, **11**, 1–24.

`cocoBoot` *Bootstrap Based Model Assessment Procedure*

Description

Model checking procedure emphasizing reproducibility in fitted models to provide an overall evaluation of fit as proposed by Tsay (1992).

Usage

```
cocoBoot(
  coco,
  numb.lags = 21,
  rep.Bootstrap = 400,
  confidence = 0.95,
  julia = FALSE,
  julia_seed = NULL
)
```

Arguments

coco	An object of class coco
numb.lags	Number of lags for which to compute autocorrelations
rep.Bootstrap	Number of bootstrap replicates to use
confidence	Confidence level for the intervals
julia	if TRUE, the bootstrap is run with Julia.
julia_seed	Seed for the julia implementation. Only used if julia equals TRUE.

Details

Computes bootstrap confidence intervals for the autocorrelations of a fitted model.

Value

an object of class cocoBoot. It contains the bootstraped confidence intervals of the autocorrelations and information on the model specifications.

References

Tsay, R. S. (1992) Model checking via parametric bootstraps in time series analysis. *Applied Statistics* **41**, 1–15.

Examples

```
lambda <- 1
alpha <- 0.4
set.seed(12345)
data <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
fit <- cocoReg(order = 1, type = "Poisson", data = data)

#assessment using bootstrap - R implementation
boot_r <- cocoBoot(fit, rep.Bootstrap=400)
```

---

cocoForecast *One-Step Ahead Forecast Distribution*

Description

Computes the one-step ahead forecast distribution for the models included in the coconots package.

Usage

```
cocoForecast(
  coco,
  max = NULL,
  epsilon = 1e-05,
  xcast = NULL,
  decimals = 4,
  julia = FALSE
)
```

Arguments

coco	An object of class coco
max	The maximum number of the forecast support for the plot. If NULL all values for which the cumulative distribution function is below 1- epsilon are used for the plot.
epsilon	If max is NULL, epsilon determines the range of the support that is used by subsequent automatic plotting using R's plot() function.
xcast	A vector of covariate values for forecasting
decimals	Number of decimal places for the forecast probabilities
julia	if TRUE, the estimate is predicted with Julia.

Details

Returns forecasts for each mass point of the one-step ahead distribution for the fitted model. The exact predictive distributions for the models included here are provided in Jung and Tremayne (2011), maximum likelihood estimates replace the true model parameters. Out-of-sample values for covariates can be provided, if necessary.

Point forecasts are provided by returning the median and the mode of the predictive distribution.

Value

an object of class cocoBoot. It contains the the probability mass, mode, and median of the forecast.

Examples

```
lambda <- 1
alpha <- 0.4
set.seed(12345)
data <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
#julia_installed = TRUE ensures that the fit object
#is compatible with the julia cocoForecast implementation
fit <- cocoReg(order = 1, type = "Poisson", data = data)

#median, mode, and density forecasts - R implementation
forecast_r <- cocoForecast(fit)
```

`cocoPit` *Probability Integral Transform Based Model Assessment Procedure*

Description

Computes the probability integral transform (PIT) and provides the non-randomized PIT histogram for assessing absolute performance of a fitted model as proposed by Czado et al. (2009).

Usage

```
cocoPit(coco, J = 10, alpha = 0.05, julia = FALSE)
```

Arguments

`coco`	An object of class coco
`J`	Number of bins for the histogram (default: 10)
`alpha`	Confidence level for the confidence bands.
`julia`	if TRUE, the PIT is computed with Julia.

Details

The adequacy of a distributional assumption for a model is checked by checking the cumulative non-randomized PIT distribution for uniformity. A useful graphical device is the PIT histogram, which displays this distribution to J equally spaced bins. We supplement the graph by incorporating approximately $100(1-\alpha)\%$ confidence intervals obtained from a standard chi-square goodness-of-fit test of the null hypothesis that the J bins of the histogram are drawn from a uniform distribution. For details, see Jung, McCabe and Tremayne (2016).

Value

an object of class cocoPit. It contains the The probability integral transform values, its p-values and information on the model specifications.

Author(s)

Manuel Huth

References

Czado, C., Gneiting, T. and Held, L. (2009) Predictive model assessment for count data. *Biometrics* **65**, 1254–61.

Jung, Robert C., Brendan P. M. McCabe, and Andrew R. Tremayne. (2016). Model validation and diagnostics. *In Handbook of Discrete Valued Time Series*. Edited by Richard A. Davis, Scott H. Holan, Robert Lund and Nalini Ravishanker. Boca Raton: Chapman and Hall, pp. 189–218.

Jung, R. C. and Tremayne, A. R. (2011) Convolution-closed models for count time series with applications. *Journal of Time Series Analysis*, **32**, 3, 268–280.

Examples

```
lambda <- 1
alpha <- 0.4
set.seed(12345)
data <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
#julia_installed = TRUE ensures that the fit object
#is compatible with the julia cocoPit implementation
fit <- cocoReg(order = 1, type = "Poisson", data = data)

#PIT R implementation
pit_r <- cocoPit(fit)
```

---

cocoReg *cocoReg*

Description

The function fits first and second order (Generalized) Poisson Autoregressive (G)PAR models presented in (Jung and Tremayne, 2010). Autoregressive dependence on past counts is modelled by a special random operator that not only preserve integer status, but also, via the property of closure under convolution, ensure that the marginal distribution of the observed counts is from the same family as the innovations. The models can be thought of as stationary Markov chains of finite order, where the distribution of the innovations can either be Poisson or Generalized Poisson, where the latter can account for overdispersed data. Maximum likelihood is used for estimation and the user can choose to include linear constraints or not. If linear constraints are not included, it cannot be guaranteed that the parameters will lie in the theoretically feasible parameter space, but the optimization process might be faster. The function uses method of moments estimators to obtain starting values for the numerical optimization, but the user can also specify their own starting values if desired.

If Julia is installed, the user can choose whether the optimization is run in Julia which might faster yield results and increased numeric stability due to the use of automatic differentiation. See details for more information on the Julia implementation.

Usage

```
cocoReg(
  type,
  order,
  data,
  xreg = NULL,
  constrained.optim = TRUE,
  b.beta = -10,
  start = NULL,
  start.val.adjust = TRUE,
  method_optim = "Nelder-Mead",
  replace.start.val = 1e-05,
  iteration.start.val = 0.6,
```

```
  method.hessian = "Richardson",
  cores = 2,
  julia = FALSE,
  julia_installed = FALSE
)
```

Arguments

type	character string indicating the type of model to be fitted
order	integer vector indicating the order of the model
data	time series data to be used in the analysis
xreg	optional matrix of explanatory variables for use in a regression model
constrained.optim	logical indicating whether optimization should be constrained, currently only available in the R version
b.beta	numeric value indicating the lower bound for the parameters of the explanatory variables for the optimization, currently only available in the R version
start	optional numeric vector of starting values for the optimization
start.val.adjust	logical indicating whether starting values should be adjusted, currently only available in the R version
method_optim	character string indicating the optimization method to be used, currently only available in the R version. In the julia implementation this is by default the LBFGS algorithm
replace.start.val	numeric value indicating the value to replace any invalid starting values, currently only available in the R version
iteration.start.val	numeric value indicating the proportion of the interval to use as the new starting value, currently only available in the R version
method.hessian	character string indicating the method to be used to approximate the Hessian matrix, currently only available in the R version
cores	numeric indicating the number of cores to use, currently only available in the R version
julia	if TRUE, the model is estimated with Julia. This can improve the speed significantly since Julia makes use of derivatives using autodiff. In this case, only type, order, data, xreg, and start are used as other inputs.
julia_installed	if TRUE, the model R output will contain a Julia compatible output element.

Details

Let a time series of counts be $\{X_t\}$ and be $R(\cdot)$ a random operator that differs between model specifications. For more details on the random operator, see Jung and Tremayne (2011) and Joe (1996). The general first-order model is of the form

$$X_t = R(X_{t-1}) + I_t,$$

and the general second-order model of the form

$$X_t = R(X_{t-1}, X_{t-2}) + I_t,$$

where I_t are i.i.d Poisson ($I_t \sim Po(\lambda_t)$) or Generalized Poisson ($I_t \sim GP(\lambda_t, \eta)$) innovations. Through closure under convolution the marginal distributions of $\{X_t\}$ are therefore Poisson or Generalized Poisson distributions, respectively.

If no covariates are used $\lambda_t = \lambda$ and if covariates are used

$$\lambda_t = \exp\left(\beta_0 + \sum_{j=1}^{k} \beta_j \cdot z_{t,j}\right),$$

whereby $z_{t,j}$ is the j-th covariate at time t.

Standard errors are computed by the square root of the diagonal elements of the inverse Hessian.

This function is implemented in 2 versions. The default runs on RCPP. An alternative version uses a Julia implementation which can be chosen by setting the argument julia to TRUE. In order to use this feature, a running Julia installation is required on the system. The RCPP implementation uses the derivative-free Nelder-Mead optimizer to obtain parameter estimates. The Julia implementation makes use of Julia's automatic differentiation in order to obtain gradients such that it can use the LBFGS algorithm for optimization. This enhances the numeric stability of the optimization and yields an internal validation if both methods yield qualitatively same parameter estimates. Furthermore, the Julia implementation can increase the computational speed significantly, especially for large models.

The model assessment tools `cocoBoot`, `cocoPit`, and `cocoScore` will use a Julia implementation as well, if the `cocoReg` was run with Julia. Additionally, one can make the RCPP output of `cocoReg` compatible with the Julia model assessments by setting julia_installed to true. In this case, the user can choose between the RCPP and the Julia implementation for model assessment.

Value

an object of class coco. It contains the parameter estimates, standard errors, the log-likelihood, and information on the model specifications. If Julia is used for parameter estimation or the Julia installation parameter is set to TRUE, the results contain an additional Julia element that is called from the model Julia assessment tools if they are run with the Julia implementation.

Author(s)

Manuel Huth

References

Jung, R. C. and Tremayne, A. R. (2011) Convolution-closed models for count timeseries with applications. *Journal of Time Series Analysis*, **32**, 3, 268–280.

Joe, H. (1996) Time series models with univariate margins in the convolution-closed infinitely divisible class. *Journal of Applied Probability*, 664–677.

Examples

```
## GP2 model without covariates
length <- 1000
par <- c(0.5,0.2,0.05,0.3,0.3)
data <- cocoSim(order = 2, type = "GP", par = par, length = length)
fit <- cocoReg(order = 2, type = "GP", data = data)

##Poisson1 model with covariates
length <- 1000
period <- 50
sin <- sin(2*pi/period*(1:length))
cos <- cos(2*pi/period*(1:length))
cov <- cbind(sin, cos)
par <- c(0.2, 0.2, -0.2)
data <- cocoSim(order = 1, type = "Poisson", par = par, xreg = cov, length = length)
fit <- cocoReg(order = 1, type = "Poisson", data = data, xreg = cov)
```

---

cocoResid *Residual Based Model Assessment Procedure*

Description

Calculates the (Pearson) residuals of a fitted model for model evaluation purposes.

Usage

```
cocoResid(coco, val.num = 1e-11)
```

Arguments

coco	An object of class "coco
val.num	A non-negative real number which is used to stop the calculation of

Details

The Pearson residuals are computed as the scaled deviation of the observed count from its conditional expectation given the relevant past history, including covariates, if applicable. If a fitted model is correctly specified, the Pearson residuals should exhibit mean zero, variance one, and no significant serial correlation.

Value

a list that includes the (Pearson) residuals, conditional expectations, conditional variances, and information on the model specifications.

Author(s)

Manuel Huth

cocoScore *Scoring Rule Based Model Assessment Procedure*

Description

The function calculates the log, quadratic and ranked probability scores for assessing relative performance of a fitted model as proposed by Czado et al. (2009).

Usage

```
cocoScore(coco, val.num = 1e-10, julia = FALSE)
```

Arguments

coco	An object of class coco
val.num	A non-negative real number which is used to stop the calculation of the score in case of GP models. The default value is 1e-10
julia	if TRUE, the scores are computed with Julia.

Details

Scoring rules assign a numerical score based on the predictive distribution and the observed data to measure the quality of probabilistic predictions. They are provided here as a model selection tool and are computed as averages over the relevant set of (in-sample) predictions. Scoring rules are, generally, negatively oriented penalties that one seeks to minimize. The literature has developed a large number of scoring rules and, unless there is a unique and clearly defined underlying decision problem, there is no automatic choice of a (proper) scoring rule to be used in any given situation. Therefore, the use of a variety of scoring rules may be appropriate to take advantage of specific emphases and strengths. Three proper scoring rules (for a definition of the concept of propriety see Gneiting and Raftery, 2007) which Jung, McCabe and Tremayne (2016) found to be particularly useful are implemented. For more information see the references listed below.

Value

a list containing the log score, quadratic score and ranked probability score.

Author(s)

Manuel Huth

References

Czado, C. and Gneitling, T. and Held, L. (2009) Predictive Model Assessment for Count Data. *Biometrics*, **65**, 4, 1254–1261.

Gneiting, T. and Raftery, A. E. (2007) Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102:359-378.

Jung, Robert C., Brendan P. M. McCabe, and Andrew R. Tremayne. (2016). Model validation and diagnostics. *In Handbook of Discrete Valued Time Series*. Edited by Richard A. Davis, Scott H. Holan, Robert Lund and Nalini Ravishanker. Boca Raton: Chapman and Hall, pp. 189–218.

Jung, R. C. and Tremayne, A. R. (2011) Convolution-closed models for count timeseries with applications. *Journal of Time Series Analysis*, **32**, 3, 268–280.

Examples

```
lambda <- 1
alpha <- 0.4
set.seed(12345)
data <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
#julia_installed = TRUE ensures that the fit object
#is compatible with the julia cocoScore implementation
fit <- cocoReg(order = 1, type = "Poisson", data = data)

#assessment using scoring rules - R implementation
score_r <- cocoScore(fit)
```

`cocoSim` *Simulation of Count Time Series*

Description

The function generates a time series of low counts from the (G)PAR model class for a specified innovation distribution, sample size, lag order, and parameter values.

Usage

```
cocoSim(
  type,
  order,
  par,
  length,
  xreg = NULL,
  init = NULL,
  julia = FALSE,
  julia_seed = NULL
)
```

Arguments

`type`	character, either "Poisson" or "GP" indicating the type of the innovation distribution
`order`	integer, either 1 or 2 indicating the order of the model
`par`	numeric vector, the parameters of the model, the number of elements in the vector depends on the type and order specified.

length	integer, the number of observations in the generated time series
xreg	data.frame, data frame of control variables
init	numeric vector, initial data to use, default is NULL. See details for more information on the usage.
julia	If TRUE, the Julia implementation is used. In this case, init is ignored but it might be faster.
julia_seed	Seed for the Julia implementation. Only used if Julia equals TRUE.

Details

The function checks for valid input of the type, order, parameters, and initial data before generating the time series.

The init parameter allows users to set a custom burn-in period for the simulation. By default, when simulating with covariates, no burn-in period is specified since there is no clear choice on the covariates. However, the init argument gives users the flexibility to select an appropriate burn-in period for the covariate case. One way to do this is to simulate a time series using `cocoSim` with appropriate covariates and pass the resulting time series to the init argument of a new `cocoSim` run so that the first time series is used as the burn-in period. If init is not specified for the covariate case, a warning will be returned to prompt the user to specify a custom burn-in period. This helps ensure that the simulation accurately captures the dynamics of the system being modeled.

Value

a vector of the simulated time series.

Author(s)

Manuel Huth

Examples

```
lambda <- 1
alpha <- 0.4
set.seed(12345)

# Simulate using the RCPP implementation
data_rcpp <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
# Simulate using the Julia implementation
data_julia <- cocoSim(order = 1, type = "Poisson", par = c(lambda, alpha), length = 100)
```

cuts — *Time Series of Monthly Counts of Claimants Collecting Wage Loss Benefit for Injuries in the Workplace*

Description

Monthly counts of claimants collecting wage loss benefit for injuries in the workplace at one specific service delivery location of the Workers Compensation Board of British Columbia, Canada in the period January 1985 to December 1994. Only injuries due to cuts and lacerations are considered. The data have been provided by Brendan McCabe.

downloads — *Time Series of Daily Downloads of a TeX-Editor*

Description

The data represent the number of daily downloads of a TeX-editor between June 2006 and February 2007 and has a sample size of 267. The data have been provided by Christian Weiss.

goldparticle — *Time Series of Gold particles Counts in a well-efined Colloidal Solution*

Description

A sample of 370 counts of gold particles in a well-defined colloidal solution at equidistant points in time, originally published in Westgren (1916) and used in Jung and Tremanye (2006).

Source

R.C. Jung, A.R. Tremayne (2006) Coherent forecasting in integer time series models. *International Journal of Forecasting* **22**, 223–238

Westgren, A. (1916) Die Veraenderungsgeschwindigkeit der lokalen Teilchenkonzentration in kollioden Systemen (Erste Mitteilung). *Arkiv foer Matematik, Astronomi och Fysik*, **11**, 1–24.

`installJuliaPackages` *installJuliaPackages*

Description

checks for needed Julia packages and installs them if not installed.

Usage

```
installJuliaPackages()
```

Value

no return value, called to install Julia packages in Julia.

Index