# Package 'coconots'

August 22, 2025

**Type** Package

**Title** Convolution-Closed Models for Count Time Series

**Version** 2.0.2

**Date** 2025-08-22

**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 modelled via Poisson and Generalized Poisson innovations. Regression effects can be incorporated through 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> and, Tsay (1992) <doi:10.2307/2347612>.

**LazyData** true

**LinkingTo** Rcpp

**RoxygenNote** 7.3.2

**Depends** R (>= 4.0.2)

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

**Suggests** 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** 2025-08-22 15:30:21 UTC

# Contents



---

`cocoBoot` *Bootstrap Based Model Assessment Procedure*

---

## Description

Model checking procedure emphasising reproducibility in fitted models, as proposed by Tsay (1992).

## Usage

```
cocoBoot(
  coco,
  numb.lags = 21,
  rep.Bootstrap = 1000,
  conf.alpha = 0.05,
  julia = FALSE,
  julia_seed = NULL
)
```

## Arguments

| | |
|---|---|
| `coco` | An object of class coco |
| `numb.lags` | Number of lags for which to compute sample autocorrelations (default: 21). |
| `rep.Bootstrap` | Number of bootstrap replicates to use (default: 1000) |
| `conf.alpha` | $100(1 - \texttt{conf.alpha})\%$ probability interval for the acceptance envelopes (default: 0.05) |
| `julia` | if TRUE, the bootstrap is run with julia (default: FALSE) |
| `julia_seed` | Seed for the julia implementation. Only used if julia equals TRUE |

## Details

Bootstrap-generated acceptance envelopes for the autocorrelation function provides an overall evaluation by comparing it with the sample autocorrelation function in a joint plot.

## Value

an object of class cocoBoot. It contains the bootstrapped 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)

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

---

# 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, conf.alpha = 0.05, julia = FALSE)
```

## Arguments

| | |
|---|---|
| coco | An object of class coco |
| J | Number of bins for the histogram (default: 10) |
| conf.alpha | Significance level for the confidence intervals (default: 0.05) |
| julia | if TRUE, the PIT is computed with julia (default: FALSE) |

## Details

The adequacy of a distributional assumption for a model is assessed 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 probability integral transform values, p-value of the chi-square goodness of fit test 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, R. C., McCabe, B.P.M. and Tremayne, A.R. (2016). Model validation and diagnostics. *In Handbook of Discrete Valued Time Series*. Edited by Davis, R.A., Holan, S.H., Lund, R. and Ravishanker, N.. 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)
fit <- cocoReg(order = 1, type = "Poisson", data = data)

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

---

cocoReg *Fitting First- and Second Order (G)PAR Models*

---

## Description

The function fits first- and second-order (Generalized) Poisson integer autoregressive (G)PAR time series models for count data as discussed in Jung and Tremayne (2011). Autoregressive dependence on past counts is modeled using a special random operator that preserves integer values and, through closure under convolution, ensures that the marginal distribution remains within the same family as the innovations.

These models can be viewed as stationary finite-order Markov chains, where the innovation distribution can be either Poisson or Generalized Poisson, the latter accounting for overdispersion. Estimation is performed via maximum likelihood, with an option to impose linear constraints. Without constraints, parameters may fall outside the theoretically feasible space, but optimization may be faster.

Method of moments estimators are used to initialize numerical optimization, though custom starting values can be provided. If julia is installed, users can opt to run the optimization in julia for potentially faster computation and improved numerical stability via automatic differentiation. See below for details 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,
  link_function = "log"
)
```

## 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 |
| data | time series data to be used in the analysis |
| xreg | optional matrix of explanatory variables (without constant term) 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 (default: 2)

`julia`
: if TRUE, the model is estimated with julia. This can improve computational 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 (default: FALSE).

`julia_installed`
: if TRUE, the model R output will contain a julia compatible output element.

`link_function`
: Specifies the link function for the conditional mean of the innovation ($\lambda$). The default is `log`, but other available options include `identity` and `relu`. This parameter is applicable only when covariates are used. Note that using the `identity` link function may result in $\lambda$ becoming negative. To prevent this, ensure all covariates are positive and restrict the parameter $\beta$ to positive values by setting `b.beta` to a small positive value.

## 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

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

and the general second-order model of the form

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

where $W_t$ are i.i.d Poisson ($W_t \sim Po(\lambda_t)$) or Generalized Poisson ($W_t \sim GP(\lambda_t, \eta)$) 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

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

whereby $z_{t,j}$ is the $j$-th covariate at time $t$ and $g$ is a link function. Current supported link functions are the identity $g(x) = x$ and a logarithmic link function $g(x) = \ln x$. To ensure positivity of $\lambda$ if the identity function is used, $\beta_j, z_{t,j} > 0$ must be enforced. Alternatively, computational values of $\lambda \leq 0$ can be set to a small positive value. This option is named 'relu', due to its similarity to a ReLu function commonly used in machine learning.

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

This function is implemented in two 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 time series with applications. *Journal of Time Series Analysis*, **32**, 268–280.

Joe, H. (1996) Time series models with univariate margins in the convolution-closed infinitely divisible class. *Journal of Applied Probability*, **33**, 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 <- 12
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-10)
```

## Arguments

| | |
|---|---|
| coco | An object of class "coco |
| val.num | A non-negative real number that halts the calculation once the cumulative probability reaches 1-val.num |

## 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 computes log, quadratic and ranked probability scores for assessing relative performance of a fitted model.

## Usage

```
cocoScore(coco, max_x = 50, julia = FALSE)
```

## Arguments

| | |
|---|---|
| `coco` | An object of class coco |
| `max_x` | An integer which is used as the maximum count for the computation of the score (default: `50`) |
| `julia` | if TRUE, the scores are computed with julia (default: FALSE). |

## 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**, 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, R. C., McCabe, B.P.M. and Tremayne, A.R. (2016). Model validation and diagnostics. *In Handbook of Discrete Valued Time Series*. Edited by Davis, R.A., Holan, S.H., Lund, R. and Ravishanker, N.. 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**, 268–280.

## 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)

# 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,
  link_function = "log"
)
```

## 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 of control variables (defaul: NULL) |
| init | numeric vector, initial data to use (default: 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 (default: FALSE). |
| julia_seed | Seed for the julia implementation. Only used if julia equals TRUE. |
| link_function | Specifies the link function for the conditional mean of the innovation ($\lambda$). The default is log, but other available options include identity and relu. This parameter is applicable only when covariates are used. Note that using the identity link function may result in $\lambda$ becoming negative. To prevent this, ensure all covariates are positive and restrict the parameter $\beta$ to positive values. |

## 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

```
#First Order Model
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)

# Second Order Model
lambda <- 1
alpha_1 <- 0.3
alpha_2 <- 0.1
alpha_3 <- 0.2
eta <- 0.2
```

```
data_j <- cocoSim(order = 2, type = "GP",
                  par = c(lambda, alpha_1, alpha_2,alpha_3,eta),
                  length = 100)
```

---

cocoSoc *Computes Scores for Various Models Maintaining a Common Sample*

---

## Description

This function computes log, quadrtic and ranked probability scores for Poisson and Generalized Poisson models.

## Usage

```
cocoSoc(
  data,
  models = "all",
  print.progress = TRUE,
  max_x_score = 50,
  julia = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | A numeric vector containing the data to be used for modeling |
| `models` | A character string specifying which models to use. Default is `"all"`, which uses both Poisson and GP models. |
| `print.progress` | A logical value indicating whether to print progress messages (Default: `TRUE`). |
| `max_x_score` | An integer which is used as the maximum count for the computation of the score (defaul: `50`) |
| `julia` | if TRUE, `cocoSoc` is run with julia (default: FALSE) |
| `...` | Additional arguments to be passed to the `cocoReg` function. |

## Details

Supports model selection by computing score over a range of models while maintaining a common sample and a common specification.

## Value

A list of class `"cocoSoc"` containing:

**fits** A list of fitted model objects.

**scores_list** A list of score objects for each model.

**scores_df** A data frame containing the logarithmic, quadratic, and ranked probability scores for each model.

### Author(s)

Manuel Huth

---

`cuts` *Time Series of Monthly Counts of Claimants Collecting Wage Loss Benefit*

---

### 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.

### Usage

```
cuts
```

### Format

A time series (`ts`) object containing monthly data from January 1985 to December 1994.

### Source

Freeland, R. K.; McCabe, B.P.M. (2004) Analysis of count data by means of the Poisson autoregressive model. *Journal of Time Series Analysis*, **25**, 701–722.

---

`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. The dataset contains 267 observations. The data have been provided by Christian Weiss.

### Usage

```
downloads
```

### Format

A time series (`ts`) object containing daily data from June 2006 to February 2007.

### Source

Weiss, C.H. (2008) Thinning operations for modelling time series of counts – a survey. *Advances in Statistical Analysis*, **92**, 319–341.

## goldparticle *Time Series of Gold Particle Counts in a Well-Defined Colloidal Solution*

### Description

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

### Usage

```
goldparticle
```

### Format

A time series (`ts`) object containing 370 observations at equidistant time points.

### Source

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

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

### Examples

```
plot(goldparticle)
```

## 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.

predict.coco *K-Step Ahead Forecast Distributions*

## Description

Computes the k-step ahead forecast (distributions) using the models in the coconots package.

## Usage

```
## S3 method for class 'coco'
predict(
  object,
  k = 1,
  number_simulations = 1000,
  alpha = 0.05,
  simulate_one_step_ahead = FALSE,
  max = NULL,
  epsilon = 1e-08,
  xcast = NULL,
  decimals = 4,
  julia = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| object | An object that has been fitted previously, of class coco. |
| k | The number of steps ahead for which the forecast should be computed (Default: 1). |
| number_simulations | The number of simulation runs to compute (Default: 1000). |
| alpha | Significance level used to construct the prediction intervals (Default: 0.05). |
| simulate_one_step_ahead | If FALSE, the one-step ahead prediction is obtained using the analytical predictive distribution. If TRUE, bootstrapping is used. |
| 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 | An optional matrix of covariate values for the forecasting. If NULL, the function assumes no covariates. |
| decimals | Number of decimal places for the forecast probabilities |
| julia | if TRUE, the estimate is predicted with julia (Default: FALSE). |
| ... | Optional arguments. |

## Details

Returns forecasts for each mass point of the k-step ahead distribution for the fitted model. The exact predictive distributions for one-step ahead predictions for the models included here are provided in Jung and Tremayne (2011), maximum likelihood estimates replace the true model parameters. For k>1 forecast distributions are estimated using a parametric bootstrap. See Jung and Tremanye (2006). Out-of-sample values for covariates can be provided, if necessary.

for k > 1

## Value

A list of frequency tables. Each table represents a k-step ahead forecast frequency distribution based on the simulation runs.

## References

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

## Examples

```
length <- 500
pars <- c(1, 0.4)
set.seed(12345)
data <- cocoSim(order = 1, type = "Poisson", par = pars, length = length)
fit <- cocoReg(order = 1, type = "Poisson", data = data)
forecast <- predict(fit, k=1, simulate_one_step_ahead = FALSE)
plot(forecast[[1]]) #plot one-step ahead forecast distribution
```

---

# setJuliaSeed *Set Seed for julia's Random Number Generator*

---

## Description

Sets the seed for julia's random number generator to ensure reproducibility.

## Usage

```
setJuliaSeed(julia_seed)
```

## Arguments

| | |
|---|---|
| julia_seed | An integer seed value to be passed to julia's random number generator. |

## Details

This function initializes the necessary julia functions and sets the random seed for julia. If the provided seed is NULL, the function does nothing.

## Author(s)

Manuel Huth

# Index