Package 'glarma'

March 28, 2025

Type Package

Title Generalized Linear Autoregressive Moving Average Models

Version 1.7-1

Date 2025-03-26

Maintainer William T.M. Dunsmuir <w.dunsmuir@unsw.edu.au>

Depends R (>= 2.3.0)

Imports MASS, methods

Suggests RUnit, knitr, digest, zoo

VignetteBuilder knitr

Description Functions are provided for estimation, testing, diagnostic checking and forecasting of generalized linear autoregressive moving average (GLARMA) models for discrete valued time series with regression variables. These are a class of observation driven non-linear non-Gaussian state space models. The state vector consists of a linear regression component plus an observation driven component consisting of an autoregressive-moving average (ARMA) filter of past predictive residuals. Currently three distributions (Poisson, negative binomial and binomial) can be used for the response series. Three options (Pearson, score-type and unscaled) for the residuals in the observation driven component are available. Estimation is via maximum likelihood (conditional on initializing values for the ARMA process) optimized using Fisher scoring or Newton Raphson iterative methods. Likelihood ratio and Wald tests for the observation driven component allow testing for serial dependence in generalized linear model settings. Graphical diagnostics including model fits, autocorrelation functions and probability integral transform residuals are included in the package. Several standard data sets are included in the package.

NeedsCompilation no

Author William T.M. Dunsmuir [aut, cre],
Cenanning Li [aut],
David J. Scott [aut]

Repository CRAN

Date/Publication 2025-03-28 12:20:02 UTC

Contents

`Asthma` *Daily Presentations of Asthma at Campbelltown Hospital*

Description

The data arose from a single hospital (at Campbelltown) as part of a larger (ongoing) study into the relationship between atmospheric pollution and the number of asthma cases presenting themselves to various emergency departments in local hospitals in the South West region of Sydney, Australia.

Usage

```
data(Asthma)
```

Format

A data frame containing the following columns:

[, 1]	Count	Daily counts of asthma at Campbelltown Hospital.
[, 2]	Intercept	A vector of ones, providing the intercept in the model.
[, 3]	Sunday	Takes value one for Sundays, otherwise zero.
[, 4]	Monday	Takes value one for Mondays, otherwise zero.
[, 5]	CosAnnual	cos((2*pi*t)/365), annual cosine term.
[, 6]	SinAnnual	sin((2*pi*t)/365), annual sine term.
[, 7]	H7	Scaled lagged and smoothed humidity variable.
[, 8]	NO2max	Maximum daily nitrogen dioxide.
[, 9:16]	T1.1990 - T2.1993	Smooth shapes to capture school terms in each year.

Source

Davis, Richard A and Dunsmuir, William TM and Streett, Sarah B (2003) Observation-driven models for Poisson counts. *Biometrika*, **90**, 777–790.

Examples

```
### Example with asthma data
data(Asthma)
y <- Asthma[,1]
X <- as.matrix(Asthma[,2:16])

## Model in Davis, Dunsmuir and Streett (2003)

## MA(7) specification - see Davis, Dunsmuir and Streett (2003)

## Pearson Residuals, Fisher Scoring
glarmamod <- glarma(y, X, thetaLags = 7, type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

likTests(glarmamod)
plot.glarma(glarmamod)

## Not run:
## Example is specified as \dontrun because it takes too long
## for package inclusion on CRAN

## Pearson Residuals, Newton Raphson, Negative Binomial
## Initial value of the shape parameter take to be zero
glarmamod <- glarma(y, X, thetaLags = 7, type = "NegBin", method = "NR",
                    residuals = "Pearson", alphaInit = 0,
                    maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

likTests(glarmamod)
plot.glarma(glarmamod)

## End(Not run)
```

coef.glarma *Extract GLARMA Model Coefficients*

Description

coef is a generic function which extracts GLARMA model coefficients from objects returned by modeling functions. coefficients is an alias for it.

Usage

```
## S3 method for class 'glarma'
coef(object, types = "all", ...)
```

Arguments

object	An object of class "glarma", obtained from a call to glarma.
types	Character; which coefficients to extract, either ARMA), beta, NB or all. The default is all.
...	Further arguments passed to or from other methods.

Details

This is an S3 generic function. coef or coefficients return the requested coefficients from the object of class "glarma". By changing the argument type, either the ARMA coefficients (ARMA), regression coefficients (beta) or all coefficients are returned. In the case of negative binomial counts, the negative binomial coefficient α is also returned if type is all, or if type is NB. The default is all.

Value

ARMA coefficients, beta coefficients, NB coefficients or all of these three types of coefficients are extracted from the glarma model object object.

A named numeric vector or list of named numeric vectors is returned.

See Also

fitted.glarma and residuals.glarma for related methods;

Examples

```
data(Polio)
Y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(Y, X, thetaLags = c(1, 2, 5), type = "Poi",
                    method = "FS", residuals= "Pearson",
                    maxit = 100, grad = 1e-6)

coef(glarmamod, type = "ARMA")
coef(glarmamod, type = "beta")
coef(glarmamod, type = "all")
```

`DriverDeaths` *Single Vehicle Nighttime Driver Deaths in Utah*

Description

This data set gives the number of single vehicle nighttime driver deaths in the state of Utah by month over the period August 1980 to July 1986, along with observations on a number of possible predictors. The aim of the study from which it was taken was to investigate the effect of the lowering of the legal blood alcohol concentration (BAC) while driving, from 0.1 to 0.08 units, and the simultaneous introduction of administrative license revocation. The time period for the observations is centred on the month of the intervention, August 1983.

Usage

```
data(DriverDeaths)
```

Format

A data frame containing the following columns:

[, 1]	Deaths	Number of single vehicle nighttime driver deaths monthly.
[, 2]	Intercept	A vector of ones, providing the intercept in the model.
[, 3]	ReducedBAC	Indicator of before or after lowering of legal blood alcohol level.0 for months prior to August 1983, 1
[, 4]	FriSat	Number of Friday and Saturday nights in the month.
[, 5]	lnOMVDRate	Log of the number of other motor vehicle deaths per 100,000 of population.
[, 6]	Population	Adult population of the State of Utah.

Source

Debra H. Bernat, William T.M. Dunsmuir, and Alexander C. Wagenaar (2004) Effects of lowering the legal BAC to 0.08 on single-vehicle-nighttime fatal traffic crashes in 19 jurisdictions. *Accident Analysis & Prevention*, **36**, 1089–1097.

Examples

```
### Model number of deaths
data(DriverDeaths)
y <- DriverDeaths[, "Deaths"]
X <- as.matrix(DriverDeaths[, 2:5])
Population <- DriverDeaths[, "Population"]

### Offset included
glarmamodOffset <- glarma(y, X, offset = log(Population/100000),
                          phiLags = c(12),
                          type = "Poi", method = "FS",
                          residuals = "Pearson", maxit = 100, grad = 1e-6)
print(summary(glarmamodOffset))
par(mfrow =c(3,2))
plot(glarmamodOffset)


### No offset included
glarmamodNoOffset <- glarma(y, X, phiLags = c(12),
                            type = "Poi", method = "FS",
                            residuals = "Pearson", maxit = 100, grad = 1e-6)
print(summary(glarmamodNoOffset))
par(mfrow=c(3,2))
plot(glarmamodNoOffset)
```

`extractAIC.glarma` *Extract AIC from a GLARMA Model*

Description

`extractAIC` method for class `"glarma"`. Used to extract AIC from a `glarma` object.

Usage

```
## S3 method for class 'glarma'
extractAIC(fit, ...)
```

Arguments

`fit`	An object of class `"glarma"`, obtained from a call to `glarma`.
`...`	Further arguments passed to or from other methods.

Value

AIC extracted from `object`

See Also

`coef.glarma`, `residuals.glarma`, `glarma`.

`extractGlarmaSimModel` *Extract simulation model from a glarma fit*

Description

Extracts required components from a fitted glarma model to create an object of class "glarmaSimModel" to be used as input to simulate a glarma model using the function glarmaSim.

Usage

```
extractGlarmaSimModel(object, ...)
```

Arguments

object	An object of class "glarma" obtained from a call to glarma
...	Further arguments for the call, currently unused.

Value

An object of class "glarmaSimModel"

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "David J Scott" <d.scott@auckland.ac.nz>

See Also

glarmaSim

Examples

```
### Extract model from a glarma fit to use in simulation
### Example with asthma data
data(Asthma)
y <- Asthma[,1]
X <- as.matrix(Asthma[,2:16])
### Pearson Residuals, Fisher Scoring
glarmamod <- glarma(y, X, thetaLags = 7, type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
mod <- extractGlarmaSimModel(glarmamod)
str(mod)

glarmamod <- glarma(y, X, thetaLags = 7, type = "NegBin", method = "NR",
                    residuals = "Pearson", alphaInit = 0,
                    maxit = 100, grad = 1e-6)
mod <- extractGlarmaSimModel(glarmamod)
str(mod)

data(DriverDeaths)
```

```
y <- DriverDeaths[, "Deaths"]
X <- as.matrix(DriverDeaths[, 2:5])
Population <- DriverDeaths[, "Population"]

### Offset included
glarmamodOffset <- glarma(y, X, offset = log(Population/100000),
                          phiLags = c(12),
                          type = "Poi", method = "FS",
                          residuals = "Pearson", maxit = 100, grad = 1e-6)
mod <- extractGlarmaSimModel(glarmamodOffset)
str(mod)

data(OxBoatRace)

y1 <- OxBoatRace$Camwin
n1 <- rep(1, length(OxBoatRace$Year))
Y <- cbind(y1, n1 - y1)
X <- cbind(OxBoatRace$Intercept, OxBoatRace$Diff)
colnames(X) <- c("Intercept", "Weight Diff")

oxcamglm <- glm(Y ~ Diff + I(Diff^2),
                data = OxBoatRace,
                family = binomial(link = "logit"), x = TRUE)
summary(oxcamglm)

X <- oxcamglm$x

glarmamod <- glarma(Y, X, thetaLags = c(1, 2), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
mod <- extractGlarmaSimModel(glarmamod)
str(mod)


data(RobberyConvict)
datalen <- dim(RobberyConvict)[1]
monthmat <- matrix(0, nrow = datalen, ncol = 12)
dimnames(monthmat) <- list(NULL, c("Jan","Feb","Mar","Apr","May","Jun",
                                   "Jul","Aug","Sep","Oct","Nov","Dec"))
monthNames <- months(strptime(RobberyConvict$Date, format = "%m/%d/%Y"),
                     abbreviate=TRUE)

monthNamesUnique <- unique(monthNames)


for (j in 1:12) {
  monthmat[monthNames == monthNamesUnique[j], j] <- 1
}

RobberyConvict <- cbind(rep(1, datalen), RobberyConvict, monthmat)
rm(monthmat)

### Lower court robbery
y1 <- RobberyConvict$LC.Y
```

```
n1 <- RobberyConvict$LC.N

Y <- cbind(y1, n1-y1)

glm.LCRobbery <- glm(Y ~ Step.2001 +
                       I(Feb + Mar + Apr + May + Jun + Jul) +
                       I(Aug + Sep + Oct + Nov + Dec),
                     data = RobberyConvict, family = binomial(link = logit),
                     na.action = na.omit, x = TRUE)

summary(glm.LCRobbery, corr = FALSE)

X <- glm.LCRobbery$x


### Newton Raphson
glarmamod <- glarma(Y, X, phiLags = c(1), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
mod <- extractGlarmaSimModel(glarmamod)
str(mod)
```

`fitted.glarma` *Extract GLARMA Model Fitted Values*

Description

`fitted` method for class `"glarma"`. `fitted.values` is an alias for `fitted`.

Usage

```
## S3 method for class 'glarma'
fitted(object, ...)
```

Arguments

`object`	An object of class `"glarma"`, obtained from a call to `glarma`.
`...`	Further arguments passed to or from other methods.

Details

This is an S3 generic function. `fitted` or `fitted.values` return the required fitted values from an object of class `"glarma"`.

Value

Fitted values `mu` extracted from the object `object`.

See Also

coef.glarma, residuals.glarma, glarma.

---

forecast *Forecasting GLARMA time series*

Description

forecast is a generic function for forecasting time series or time series models. The function invokes particular *methods* which depend on the class of the first argument.

Currently the only method provided by the package is for objects of class "glarma".

Usage

```
forecast(object, ...)
## S3 method for class 'glarma'
forecast(object, n.ahead = 1, newdata = 0,
         newoffset = 0, newm = 1, ...)
```

Arguments

object	An object of class "glarma" obtained from a call to glarma
n.ahead	The number of periods ahead to be forecast.
newdata	The model matrix X comprising the values of the predictors for the times for which the series is to be predicted. Number of rows must be equal to n.ahead
newoffset	A vector containing the values of the offset for the times for which the series is to be predicted. Length must be equal to n.ahead if the model includes an offset
newm	A vector containing the number of trials when forecasting binomial or binary time series. Defaults to the binary case. Length must be equal to n.ahead
...	Further arguments for the call, currently unused.

Details

Only one forecasting method is currently provided, for objects of class "glarma". This produces an object of class "glarmaForecast".

When forecasting one step ahead, the values in the matrix newdata (and offset if there is an offset) in the GLARMA model are used along with the regression coefficients in the model to obtain the predicted value of η, the regression component of the state variable W. The predicted value of the ARMA component of the state variable is then added to this value to give the predicted value of W.

When further predictions are required, since no data is available to calculate the predicted value of the state variable, an observation is generated from the predicted distribution and the methodology for one step ahead is then used on this generated data. This process is repeated until predictions are obtained for the required number of time periods (specified by n.ahead). Note that the value

of n.ahead must equal the row dimension of newdata and if they are specified, of newoffset and newm.

For completeness a randomly generated value of the time series is produced even for one step-ahead prediction.

Note that the forecasted time series returned as the component fitted is then a randomly generated sample path for the predicted time series. If a sample of such paths is produced by repeated calls to forecast then sample predicted distributions can be obtained for the forecast series.

In the case of binary or binomial time series in addition to values of the predictors in the regression component of the state variable and the values of any offset, the numbers of trials for the binomially distributed future observations are required. This information should be provided in the argument newm. If not, the number of trials defaults to 1, which is the case of binary responses.

Value

forecast currently has no default method.

When object is of class "glarma", forecast returns an object of class "glarmaForecast" with components:

eta	the forecast values of the regression component of the state variable
W	the forecast values of the state variable W
mu	the conditional mean μ_t
Y	the simulated series based on the fitted model
n.ahead	the number of steps ahead for which the forecasts were requested in the call to forecast
newdata	the model matrix X comprising the values of the predictors for the times for which the series is to be predicted
newoffset	the vector containing the values of the offset for the times for which the series is to be predicted
newm	the vector giving the number of trials when forecasting binomial or binary time series
model	the "glarma" object from the call to forecast

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "David J Scott" <d.scott@auckland.ac.nz>

Examples

```
require(zoo)
### Model number of deaths
data(DriverDeaths)
y <- DriverDeaths[, "Deaths"]
X <- as.matrix(DriverDeaths[, 2:5])
Population <- DriverDeaths[, "Population"]

### Offset included
```

```
glarmamod <- glarma(y, X, offset = log(Population/100000),
                    phiLags = c(12),
                    thetaLags = c(1),
                    type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
print(summary(glarmamod))

XT1 <- matrix(X[72,], nrow = 1)
offsetT1 <- log(Population/100000)[72]

mu <- forecast(glarmamod, 1, XT1, offsetT1)$mu
print(mu)


### Save some values
allX <- X
allFits <- fitted(glarmamod)
ally <- y

### Look at a succession of forecasts
### Using actual values in forecasts
forecasts <- numeric(72)
for (i in (62:71)){
    y <- DriverDeaths[1:i, "Deaths"]
    X <- as.matrix(DriverDeaths[1:i, 2:5])
    Population <- DriverDeaths[1:i, "Population"]

    ## Offset included
    glarmamod <- glarma(y, X, offset = log(Population/100000),
                        phiLags = c(12),
                        thetaLags = c(1),
                        type = "Poi", method = "FS",
                        residuals = "Pearson", maxit = 100, grad = 1e-6)
    XT1 <- matrix(allX[i + 1, ], nrow = 1)
    offsetT1 <- log(DriverDeaths$Population[i + 1]/100000)
    mu <- forecast(glarmamod, 1, XT1, offsetT1)$mu
    if (i == 62){
        forecasts[1:62] <- fitted(glarmamod)
    }
    forecasts[i+1] <- mu
}
par(mfrow = c(1,1))
forecasts <- ts(forecasts[63:72], start = c(1985, 10), deltat = 1/12)
fitted <- ts(allFits, start = c(1980, 8), deltat = 1/12)
obs <- ts(DriverDeaths$Deaths, start = c(1980, 8), deltat = 1/12)
plot(obs, ylab = "Driver Deaths", lty = 2,
     main = "Single Vehicle Nighttime Driver Deaths in Utah")
points(obs)
lines(fitted, lwd = 2)
lines(forecasts, col = "red")
par(xpd = NA)
graph.param <-
    legend("top",
```

```
           legend = c("observations",expression(estimated~mu[t]),
                      expression(predicted~mu[t])),
           ncol = 3,
           cex = 0.7,
           bty = "n", plot = FALSE)
legend(graph.param$rect$left,
       graph.param$rect$top + graph.param$rect$h,
       legend = c("observations", expression(estimated~mu[t]),
                  expression(predicted~mu[t])),
       col = c("black","black","red"),
       lwd = c(1,2,1), lty = c(2,1,1),
       pch = c(1, NA_integer_, NA_integer_),
       ncol = 3,
       cex = 0.7,
       bty = "n",
       text.font = 4)
par(xpd = FALSE)

### Generate a sample of Y values 2 steps ahead and examine the distribution
data(DriverDeaths)
y <- DriverDeaths[, "Deaths"]
X <- as.matrix(DriverDeaths[, 2:5])
Population <- DriverDeaths[, "Population"]

### Fit the glarma model to the first 70 observations
glarmamod <- glarma(y[1:70], X[1:70, ],
                    offset = log(Population/100000)[1:70],
                    phiLags = c(12),
                    thetaLags = c(1),
                    type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)

nObs <- NROW(X)
n.ahead <- 2
### Specify the X matrix and offset for the times where predictions
### are required
XT1 <- as.matrix(X[(nObs - n.ahead + 1):nObs, ])
offsetT1 <- log(Population/100000)[(nObs - n.ahead + 1):nObs]
nSims <- 500
forecastY <- matrix(ncol = n.ahead, nrow = nSims)
forecastMu <- matrix(ncol = n.ahead, nrow = nSims)

### Generate sample predicted values
for(i in 1:nSims){
    temp <-  forecast(glarmamod, n.ahead, XT1, offsetT1)
    forecastY[i, ] <- temp$Y
    forecastMu[i, ] <- temp$mu
}
### Examine distribution of sample of Y values n.ahead
table(forecastY[, 2])
par(mfrow = c(2,1))
barplot(table(forecastY[, 2]),
        main = "Barplot of Sample Y Values 2 Steps Ahead")
```

```
hist(forecastY[, 2], xlab = "Sample Y values",
     main = paste0("Histogram of Sample Y Values 2 Steps Ahead",
                   "\nwith 0.025 and 0.975 Quantiles"))
abline(v = quantile(forecastY[, 2], c(0.025, 0.975)), col = "red")
```

---

glarma *Generalized Linear Autoregressive Moving Average Models with Various Distributions*

Description

The function glarma is used to fit generalized linear autoregressive moving average models with various distributions (Poisson, binomial, negative binomial) using either Pearson residuals or score residuals, and for the binomial distribution, identity residuals. It also estimates the parameters of the GLARMA model with various distributions by using Fisher scoring or Newton-Raphson iteration.

For Poisson and negative binomial response distributions the log link is currently used. For binomial responses the logit link is currently used.

Usage

```
glarma(y, X, offset = NULL, type = "Poi", method = "FS", residuals = "Pearson",
       phiLags,  thetaLags, phiInit, thetaInit, beta, alphaInit,
       alpha = 1, maxit = 30, grad = 2.22e-16)

glarmaPoissonPearson(y, X, offset = NULL, delta, phiLags, thetaLags,
                     method =  "FS")

glarmaPoissonScore(y, X, offset = NULL, delta, phiLags, thetaLags,
                   method =  "FS")

glarmaBinomialIdentity(y, X, offset = NULL, delta, phiLags, thetaLags,
                       method =  "FS")

glarmaBinomialPearson(y, X, offset = NULL, delta, phiLags, thetaLags,
                      method =  "FS")

glarmaBinomialScore(y, X, offset = NULL, delta, phiLags, thetaLags,
                    method =  "FS")

glarmaNegBinPearson(y, X, offset = NULL, delta, phiLags, thetaLags,
                    method =  "FS")

glarmaNegBinScore(y, X, offset = NULL, delta, phiLags, thetaLags,
                  method =  "FS")
```

Arguments

y	Numeric vector; the response variable. If the response variable is for the model with the binomial distribution, it should be a n by 2 matrix, one column is the number of successes and another is the number of failures.
X	Matrix; the explanatory variables. A vector of ones should be added to the data matrix as the first column for the beta of the intercept.
offset	Either NULL or a numeric vector of length equal to the number of cases. Used to specify an *a priori* known component to be included in the linear predictor during fitting.
beta	Numeric vector; initial values of the regression coefficients.
phiLags	Numeric vector; AR orders.
phiInit	Numeric vector; initial values for the corresponding AR orders.
thetaLags	Numeric vector; MA orders.
thetaInit	Numeric vector; initial values for the corresponding MA orders.
delta	Numeric vector; initial values of the parameters for the GLARMA estimation procedure. It is a combination of the parameters of beta, the AR terms and the MA terms.
alpha	Numeric; an optional initial shape parameter for glm.nb.
alphaInit	Numeric; an initial shape parameter for glarma for negative binomial counts.
type	Character; the count distribution. Possible values are "Poi" (Poisson), "Bin" (binomial) and "NegBin" (negative binomial). The default is the Poisson distribution.
method	Character; method of iteration to be used. Possible values are "FS" (Fisher scoring), and "NR" (Newton-Raphson). The default is to use Fisher scoring to estimate the parameters of a GLARMA model.
residuals	Character; the type of residuals to be used. Possible values are "Pearson" and "Score", and for the binomial distribution "Identity" is also allowed. The default is to use Pearson residuals.
maxit	Numeric; the maximum number of iterations allowed.
grad	Numeric; the tolerance for recognizing numbers, which are smaller than the specified tolerance, as zero.

Details

Models for glarma are specified symbolically. A typical model has the form y (response), X (terms) where y is the count or factor reponse vector, X is a series of terms which specifies a linear predictor for the response. It should be noted that the first column of X should be a vector of 1s as the intercept in the model. Four initial parameters that need to be estimated are combined into $\delta = (\beta, \phi, \theta, \alpha)$, where α is an optional parameter to accomodate the negative binomial model. Note that in the function glm.nb from the package **MASS**, this parameter is called theta.

For Poisson and negative binomial response distributions the log link is currently used. For binomial responses the logit link is currently used.

The generalized linear autoregressive moving average models are computed as follows.

The linear predictor for the response is

$$\log \mu_t = W_t = X_t^T \beta + \text{offset} + Z_t.$$

The infinite moving average from the linear predictor is

$$Z_t = \sum_{i=1}^{\infty} \gamma_i e_{t-i}.$$

This infinite moving average, is computed using the autoregressive moving average recursions

$$Z_t = \phi_1(Z_{t-1} + e_{t-1}) + ... + \phi_p(Z_{t-p} + e_{t-p}) + \theta_1 e_{t-1} + ... + \theta_q e_{t-q}$$

where p and q are the orders of ϕ and θ respectively and the non-zero lags of the vectors `phi` and `theta` may be specified by the user via the arguments `phiLag` and `thetaLag`.

There are two types of residuals which may be used in each recursion, Pearson residuals or score residuals, and in addition, for the binomial distribution, identity residuals may be used. The infinite moving average, Z_t, depends on the type of residuals used, as do the final parameters obtained from the filter. Standardisation of past observed counts is necessary to avoid instability, therefore the user should choose the appropriate type of residuals depending on the situation.

The method of estimation for parameters implemented in the function aims to maximise the log likelihood by an iterative method commencing from suitably chosen initial values for the parameters. Starting from initial values $\hat{\delta}^{(0)}$ for the vector of parameters updates are obtained using the iterations

$$\hat{\delta}^{(k+1)} = \hat{\delta}^{(k)} + \Omega(\hat{\delta}^{(k)}) \frac{\partial l(\hat{\delta}^{(k)})}{\partial \delta}$$

where $\Omega(\hat{\delta}^{(k)})$ is some suitably chosen matrix.

Iterations continue for $k \geq 1$ until convergence is reached or the number of iterations k reaches a user specified upper limit on maximum iterations in which case they will stop. The convergence criterion used in our implementation is that based on η, the maximum of absolute values of the first derivatives.

When η is less than a user specified value `grad` the iterations stop. There are two methods of optimization of the likelihood, Newton-Raphson and Fisher scoring. The method used is specified by the argument `method`. It should be noticed that if the initial value for parameters are not chosen well, the optimization of the likelihood might fail to converge. Care is needed when fitting mixed ARMA specifications because there is potential for the AR and MA parameters to be non-identifiable if the orders p and q are too large. Lack of identifiability manifests itself in the algorithm to optimize the likelihood failing to converge and/or the hessian being singular—check the warning messages and convergence error codes.

Value

The function `summary` (i.e., `summary.glarma`) can be used to obtain or print a summary of the results.

The generic accessor functions coef (i.e., coef.glarma), logLik (i.e., logLik.glarma), fitted (i.e., fitted.glarma), residuals (i.e., residuals.glarma), nobs (i.e., nobs.glarma), model.frame (i.e., model.frame.glarma) and extractAIC (i.e., extractAIC.glarma) can be used to extract various useful features of the value returned by glarma.

glarma returns an object of class "glarma" with components:

delta	a vector of coefficients for beta, AR and MA.
logLik	the loglikelihood of the specific distribution.
logLikDeriv	the derivative of the loglikelhood of the specified distribution.
logLikDeriv2	the second derivative of the loglikelihood of the specified distribution.
eta	the estimated linear predictor.
mu	the GLARMA estimated mean.
fitted.values	the GLARMA fitted values.
residuals	the residuals of the type specified.
cov	the estimated covariance matrix of the maximum likelihood estimators.
phiLags	vector of AR orders.
thetaLags	vector of MA orders.
r	the number of columns in the model matrix.
pq	the number of phiLags plus the number of thetaLags.
null.deviance	the deviance from the initial GLM fit.
df.null	the degrees of freedom from the initial GLM fit.
y	the y vector used in the GLARMA model.
X	the model matrix.
offset	the offset, NULL if there is no offset.
type	the distribution of the counts.
method	the method of iteration used.
residType	the type of the residuals returned.
call	the matched call.
iter	the number of iterations.
errCode	the error code; 0 indicating successful convergence of the iteration method, 1 indicating failure.
WError	error code for finiteness of W; 0 indicating all values of W are finite, 1 indicating at least one infinite value.
min	the minimum of the absolute value of the gradient.
aic	A version of Akaike's An Information Criterion, minus twice the maximized log-likelihood plus twice the number of parameters, computed by the aic component of the family. For binomial and Poisson families the dispersion is fixed at one and the number of parameters is the number of coefficients.

Author(s)

The original GLARMA routine for Poisson responses was developed in collaboration with Richard A. Davis and Ying Wang. The binomial response version was developed with the assistance of Haolan Lu. The extension to negative binomial response was carried out by Bo Wang. Daniel Drescher contributed to the initial structure of the software used as the basis of the package.

The main author of the package is "William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>. Package development was carried out by Cenanning Li supervised by David J. Scott.

References

Dunsmuir, William T. M. and Scott, David J. (2015) The **glarma** Package for Observation-Driven Time Series Regression of Counts. *Journal of Statistical Software*, **67(7)**, 1–36. doi:10.18637/jss.v067.i07

See Also

Additional examples may be found in Asthma, OxBoatRace, RobberyConvict, and DriverDeaths.

Examples

```
### Example from Davis, Dunsmuir Wang (1999)
## MA(1,2,5), Pearson Residuals, Fisher Scoring
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

## Score Type (GAS) Residuals, Fisher Scoring
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Score", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

## Score Type (GAS)  Residuals, Newton Raphson
## Note: Newton Raphson fails to converge from GLM initial estimates.
## Setting up the initial estimates by ourselves
init.delta <- glarmamod$delta
beta <- init.delta[1:6]
thetaInit <- init.delta[7:9]

glarmamod <- glarma(y, X, beta = beta, thetaLags = c(1, 2, 5),
                    thetaInit = thetaInit, type ="Poi", method = "NR",
                    residuals = "Score", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

## AR(1,5), Pearson Residuals, Fisher Scoring
glarmamod <- glarma(y, X, phiLags = c(1, 5), type = "Poi", method = "FS",
```

```
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)
```

---

glarmaSim *Simulate a glarma process.*

Description

Simulate a single instance of a glarma process specified by an object of class `glarmaSim`.

Usage

```
glarmaSim(object)
```

Arguments

`object`	An object of class "glarmaSim" either constructed to be of that class or derived from a fitted glarma model using the function `extractGlarmaSimModel`.

Value

An object of class `"glarmaSimulation"`.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "David J Scott" <d.scott@auckland.ac.nz>

See Also

`glarmaSim`

Examples

```
### Polio data
data("Polio")
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
PolioModel <- extractGlarmaSimModel(glarmamod)
str(PolioModel)
par(mfrow = c(3,1))
sim <- glarmaSim(PolioModel)
ts.plot(sim$W)
ts.plot(sim$mu)
ts.plot(sim$Y)
```

```
### Example with Oxford-Cambridge Boat Race
data(OxBoatRace)

y1 <- OxBoatRace$Camwin
n1 <- rep(1, length(OxBoatRace$Year))
Y <- cbind(y1, n1 - y1)
X <- cbind(OxBoatRace$Intercept, OxBoatRace$Diff)
colnames(X) <- c("Intercept", "Weight Diff")

oxcamglm <- glm(Y ~ Diff + I(Diff^2),
                data = OxBoatRace,
                family = binomial(link = "logit"), x = TRUE)
summary(oxcamglm)

X <- oxcamglm$x

glarmamod <- glarma(Y, X, thetaLags = c(1, 2), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
str(glarmamod)
BoatRaceModel <- glarmaSimModel(X, beta = coef.glarma(glarmamod, type = "beta"),
                                phiLags = glarmamod$phiLags,
                                phi = rep(0.1, length(glarmamod$phiLags)),
                                thetaLags = glarmamod$thetaLags,
                                theta = c(0.34,0.56),
                                type = glarmamod$type, m = n1,
                                residType = glarmamod$residType)
str(BoatRaceModel)
BoatRaceModel <- extractGlarmaSimModel(glarmamod)
str(BoatRaceModel)
par(mfrow = c(3,1))
sim <- glarmaSim(BoatRaceModel)
ts.plot(sim$W)
ts.plot(sim$mu)
ts.plot(sim$Y)


### Example with asthma data, negative binomial
data(Asthma)
y <- Asthma[, 1]
X <- as.matrix(Asthma[, 2:16])

## Pearson Residuals, Newton Raphson, Negative Binomial
## Initial value of the shape parameter take to be zero
glarmamod <- glarma(y, X, thetaLags = 7, type = "NegBin", method = "NR",
                    residuals = "Pearson", alphaInit = 0,
                    maxit = 100, grad = 1e-6)
summary(glarmamod)
str(glarmamod)
AsthmaModel <- glarmaSimModel(X, beta = coef.glarma(glarmamod, type = "beta"),
                              phiLags = glarmamod$phiLags,
                              phi = rep(0.1, length(glarmamod$phiLags)),
                              thetaLags = glarmamod$thetaLags,
```

```
                              theta = c(0.044),
                              type = glarmamod$type, alpha = 37.19,
                              residType = glarmamod$residType)
str(AsthmaModel)
AsthmaModel <- extractGlarmaSimModel(glarmamod)
str(AsthmaModel)
par(mfrow = c(3,1))
sim <- glarmaSim(AsthmaModel)
ts.plot(sim$W)
ts.plot(sim$mu)
ts.plot(sim$Y)
```

`glarmaSimModel` *Create a glarma simulation model*

Description

Defines a glarma model for input to the function `glarmaSim`.

Usage

```
glarmaSimModel(X, beta, offset = NULL, type = "Poi", mBin = NULL,
               alpha = NULL, residType = "Pearson",
               phiLags = NULL, phi = NULL,
               thetaLags = NULL, theta = NULL)
```

Arguments

`X`	Matrix; the explanatory variables. A vector of ones should be added to the data matrix as the first column for the `beta` of the intercept.
`beta`	Numeric vector; values of the regression coefficients.
`offset`	Either `NULL` or a numeric vector of length equal to the number of cases.
`type`	Character; the count distribution. Possible values are `"Poi"` (Poisson), `"Bin"` (binomial) and `"NegBin"` (negative binomial). The default is the Poisson distribution.
`mBin`	Numeric vector of length equal to the forecast horizon; only for the binomial case, the number of trials for each time point.
`alpha`	Numeric; for the negative binomial case, the shape parameter for `glm.nb`
`residType`	Character; the type of residuals to be used. Possible values are `"Pearson"` and `"Score"`, and for the binomial distribution `"Identity"` is also allowed. The default is to use Pearson residuals.
`phiLags`	Numeric vector; AR orders.
`phi`	Numeric vector; values for the corresponding AR orders.
`thetaLags`	Numeric vector; MA orders.
`theta`	Numeric vector; values for the corresponding MA orders.

Value

Creates a list object of type "glarmaSimModel" with the same list elements as in the function call.

Author(s)

"David J. Scott" <d.scott@auckland.ac.nz> and "William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>

See Also

See glarmaSim.

Examples

```
### Test glarmaSimModel
data("Polio")
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
summary(glarmamod)
PolioModel <- glarmaSimModel(X, beta = coef.glarma(glarmamod, type = "beta"),
                             phiLags = glarmamod$phiLags,
                             phi = rep(0.1, length(glarmamod$phiLags)),
                             thetaLags = glarmamod$thetaLags,
                             theta = rep(0.1, length(glarmamod$thetaLags)),
                             type = glarmamod$type,
                             residType = glarmamod$residType)
str(PolioModel)
coef.glarma(glarmamod, type = "ARMA")
PolioModel <- extractGlarmaSimModel(glarmamod)
str(PolioModel)
```

---

initial *Initial Parameter Generator for GLARMA from GLM*

Description

Function used to generate initial values of parameters for the GLARMA model from glm or glm.nb.

Usage

```
initial(y, X, offset = NULL, type = "Poi", alpha = 1)
```

Arguments

y	Numeric vector; response variable.
X	Matrix; explanatory variables. A vector of ones should be added to the data matrix as the first column for the β of the intercept.
offset	Either NULL or a numeric vector of length equal to the number of cases. Used to specify an *a priori* known component to be included in the linear predictor during fitting.
type	Character; the distribution of the counts.
alpha	Numeric; an optional initial value for the theta parameter in the negative binomial distribution; the default value is 1.

Details

Generates and returns the initial parameters for the GLARMA model under the specified distribution by fitting a generalized linear model.

Value

beta	A named numeric vector of initial coefficients.
y	If requested, the y vector used.
X	If requested, the model matrix.
alpha	The theta parameter in the negative binomial distribution returned by glm.nb. NULL if any other distribution is used.
type	The distribution of the counts in the GLARMA model.
null.deviance	Null deviance of the GLM with the same regression structure as the GLARMA model.
df.null	Null degrees of freedom of the GLM with the same regression structure as the GLARMA model.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>

Examples

```
### Using the polio data
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])

glmMod <- initial(y, X, type = "Poi", alpha=1)
str(glmMod)
head(glmMod)
```

likTests *Likelihood Ratio Test and Wald Test for* GLARMA *Fit*

Description

Function to carry out the likelihood ratio and Wald tests of serial dependence when the alternative is a GLARMA process. This function takes a "glarma" object and uses its attributes to set up a GLM fit that matches the GLARMA model regression structure. This is done to ensure that the GLM object is the null hypothesis for testing against the "glarma" object.

Usage

```
likTests(object)
likeTests(object)
## S3 method for class 'likTests'
print(x, ...)
```

Arguments

object	An object of class "glarma", obtained from a call to glarma.
x	An object of class "likTests", a result of a call to likTests
...	Further arguments passed to or from other methods.

Details

This function carries out the likelihood ratio and Wald tests for comparing the null model and the alternative model.

likeTests is an alias for likTests.

Value

likTests returns an object of class "likTests". A matrix is shown with the statistics and p-value for each test. The significance stars alongside help to identify any probabilities less than 0.05 or 0.01.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>

Examples

```
### Binomial response
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 2.22e-16)
```

```
likTests(glarmamod)
likeTests(glarmamod)
### Negative binomial response
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "NegBin", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 2.22e-16)
glarmamod
summary(glarmamod)
likeTests(glarmamod)

### Negative Binomial Response with offset
glarmamod.offset <- glarma(y, X, thetaLags = c(1,2,5), type = "NegBin",
                           method = "FS", offset=log(X[,1]),
                           residuals = "Pearson", maxit = 100, grad = 2.22e-16)
glarmamod.offset
summary(glarmamod.offset)
likeTests(glarmamod.offset)
```

`logLik.glarma` *Extract Log-Likelihood from GLARMA Models*

Description

`logLik` is a generic function which extracts the GLARMA model log-likelihood from objects returned by modeling functions.

Usage

```
## S3 method for class 'glarma'
logLik(object, deriv, ...)
```

Arguments

`object`	An object of class `"glarma"`, a result of a call to `glarma`.
`deriv`	Numeric; either "0", "1" or "2". It is used to choose and extract the log-likehood, its derivative or its second derivative respectively from the `"glarma"` object. The default is "0".
`...`	Further arguments passed to or from other methods.

Details

This is an S3 generic function. `logLik` returns the log-likelihood, its derivative, or its second derivative from the object of class `glarma` based on the value of the argument `deriv`. "0" is for the log-likelihood, "1" is for the first derivative of log-likelihood and "2" is for the second derivative of the log-likelihood.

Value

The log-likelihood, the derivative of the log-likelihood or the second derivative of the log-likelihood extracted from the GLARMA model object `object`.

See Also

coef.glarma, residuals.glarma, fitted.glarma, glarma.

Examples

```
data(Polio)
Y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(Y, X, thetaLags = c(1, 2, 5), type = "Poi", method ="FS",
                    residuals = "Pearson", maxit = 100 , grad = 1e-6)

logLik(glarmamod, deriv = 0)
logLik(glarmamod, deriv = 1)
logLik(glarmamod, deriv = 2)
```

---

model.frame.glarma *Extracting the Model Frame of the GLARMA Model*

Description

model.frame (a generic function) and its methods return a data frame with the variables that are used in the glarma model.

Usage

```
## S3 method for class 'glarma'
model.frame(formula, ...)
```

Arguments

formula	An object of class glarma, obtained from a call to glarma.
...	Further arguments passed to or from other methods.

Details

This is an S3 generic function. It extracts the response variable vector and the matrix of the explanatory variables from the object of class "glarma", and combines them as a data frame.

Value

A data frame with the variables used in the fitted glarma model.

Author(s)

Cenanning Li <cli113@aucklanduni.ac.nz>

See Also

coef.glarma, residuals.glarma, fitted.glarma, glarma.

Examples

```
data(Polio)
print(y <- Polio[, 2])
X <- as.matrix(Polio[, 3:8])
str(X)
head(X)


glarmamod <- glarma(y, X, thetaLags = c(1, 2, 5), type = "Poi",
                    method = "FS", residuals = "Pearson",
                    maxit = 100, grad = 1e-6)

str(model.frame(glarmamod))
head(model.frame(glarmamod))
```

`mySolve` *Matrix Inversion of the Hessian of the Log-Likelihood*

Description

Inverts the second derivative matrix of the log-likelihood to obtain the estimated covariance matrix of the parameters.

Usage

```
mySolve(A)
```

Arguments

`A`	Matrix; the negative second derivative of the log-likelihood

Details

`mySolve` attempts to invert its matrix argument. If the matrix supplied is not invertible, `ErrCode` is set to 1.

Value

`Ainv`	inverse of the negative second derivative of the loglikelihood. If the inverse is unable to be obtained, returns the original negative second derivative of the log-likelihood.
`ErrCode`	Numeric; 0 if the inverse can be found, 1 if not.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>

Examples

```
### Using the polio data
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])

## Construct the vectors of phi lags and theta lags
theta.lags <- c(1, 2, 5)
phi.lags <- rep(0, 0)
## Construct the initial delta vector
delta <- c("Intcpt" = 0.2069383, "Trend" = -4.7986615 ,
           "CosAnnual" = -0.1487333, "SinAnnual" = -0.5318768,
           "CosSemiAnnual" = 0.1690998, "SinSemiAnnual" = -0.4321435,
           "theta_1" = 0, "theta_2"= 0, "theta_5"= 0 )

## Calculate the second derivative of the loglikelihood
glarmamod <- glarmaPoissonPearson(y, X, delta = delta, phiLags = phi.lags,
                          thetaLags = theta.lags, method = "FS")

## estimate the covariance matrix of the estimators from the second
## derivative of the loglikelihood
mySolve(-glarmamod$ll.dd)
```

`nobs.glarma` *Extract the Number of Observations from a GLARMA Model Fit*

Description

An accessor function used to extract the number of observations from a `"glarma"` object.

Usage

```
## S3 method for class 'glarma'
nobs(object, ...)
```

Arguments

`object`	An object of class `"glarma"`, obtained from a call to `glarma`.
`...`	Further arguments passed to or from other methods.

Value

The number of observations extracted from the object `object`.

Author(s)

"Cenanning Li" <cli113@aucklanduni.ac.nz>

See Also

coef.glarma, residuals.glarma, fitted.glarma, glarma.

Examples

```
### Example from Davis, Dunsmuir Wang (1999)
## MA(1,2,5), Pearson Residuals, Fisher Scoring
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 2.22e-16)

nobs(glarmamod)
```

---

normRandPIT *Random normal probability integral transformation*

Description

Function to create the normalized conditional (randomized) quantile residuals.

Usage

```
normRandPIT(object)
```

Arguments

object	an object of class "glarma"

Details

The function glarmaPredProb produces the non-randomized probability integral transformation (PIT). It returns estimates of the cumulative predictive probabilities as upper and lower bounds of a collection of intervals. If the model is correct, a histogram drawn using these estimated probabilities should resemble a histogram obtained from a sample from the uniform distribution. This function aims to produce observations which instead resemble a sample from a normal distribution. Such a sample can then be examined by the usual tools for checking normality, such as histograms, Q-Q normal plots and for checking independence, autocorrelation and partial autocorrelation plots, and associated portmanteau statistics.

For each of the intervals produced by glarmaPredProb, a random uniform observation is generated, which is then converted to a normal observation by applying the inverse standard normal distribution function (that is qnorm). The vector of these values is returned by the function in the list element rt. In addition non-random observations which should appear similar to a sample from a normal distribution are obtained by applying qnorm to the mid-points of the predictive distribution intervals. The vector of these values is returned by the function in the list element rtMid.

Value

A list consisting of two elements:

`rt`	the normalized conditional (randomized) quantile residuals
`rtMid`	the midpoints of the predictive probability intervals

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "David J Scott" <d.scott@auckland.ac.nz>

References

Berkowitz, J. (2001) Testing density forecasts, with applications to risk management. *Journal of Business & Economic Statistics*, **19**, 465–474.

Dunn, Peter K. and Smyth, Gordon K. (1996) Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, **5**, 236–244.

See Also

See also as `glarmaPredProb`.

Examples

```
data(DriverDeaths)
y <- DriverDeaths[, "Deaths"]
X <- as.matrix(DriverDeaths[, 2:5])
Population <- DriverDeaths[, "Population"]

### Offset included
glarmamodOffset <- glarma(y, X, offset = log(Population/100000),
                          phiLags = c(12),
                          type = "Poi", method = "FS",
                          residuals = "Pearson", maxit = 100, grad = 1e-6)
rt <- normRandPIT(glarmamodOffset)$rt
par(mfrow = c(2,2))
hist(rt, main = "Histogram of Randomized Residuals",
     xlab = expression(r[t]))
box()
qqnorm(rt, main = "Q-Q Plot of  Randomized Residuals" )
abline(0, 1, lty = 2)
acf(rt, main = "ACF of Randomized Residuals")
pacf(rt, main = "PACF of Randomized Residuals")
```

OxBoatRace *Oxford-Cambridge Boat Race*

Description

Results of the boat race between Oxford and Cambridge from 1829–2011.

Usage

```
data(OxBoatRace)
```

Format

A data frame containing the following columns:

[, 1]	Year	Year in which the race occurred. Some years are missing when the race was not run.
[, 2]	Intercept	A vector of ones, providing the intercept in the model.
[, 3]	Camwin	A binary response, zero for an Oxford win, one for a Cambridge win.
[, 4]	WinnerWeight	Weight of winning team's crew.
[, 5]	LoserWeight	Weight of losing team's crew.
[, 6]	Diff	Difference between winning team's weight and losing team's weight.

Source

Klingenberg, Bernhard (2008) Regression models for binary time series with gaps. *Computational Statistics & Data Analysis*, **52**, 4076–4090.

Examples

```
### Example with Oxford-Cambridge Boat Race
data(OxBoatRace)

y1 <- OxBoatRace$Camwin
n1 <- rep(1, length(OxBoatRace$Year))
Y <- cbind(y1, n1 - y1)
X <- cbind(OxBoatRace$Intercept, OxBoatRace$Diff)
colnames(X) <- c("Intercept", "Weight Diff")

oxcamglm <- glm(Y ~ Diff + I(Diff^2),
                data = OxBoatRace,
                family = binomial(link = "logit"), x = TRUE)
summary(oxcamglm)

X <- oxcamglm$x

glarmamod <- glarma(Y, X, thetaLags = c(1, 2), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
```

```
summary(glarmamod)
likTests(glarmamod)

## Plot Probability of Cambridge win versus Cambridge Weight advantage:
beta <- coef(glarmamod, "beta")
par(mfrow = c(1, 1))
plot(OxBoatRace$Diff, 1 / (1 + exp(-(beta[1] + beta[2] * OxBoatRace$Diff +
                                     beta[3] * OxBoatRace$Diff^2))),
     ylab = "Prob", xlab = "Weight Diff")
title("Probability of Cambridge win \n versus Cambridge weight advantage")

## Residuals and fit plots
par(mfrow=c(3, 2))
plot.glarma(glarmamod)
```

---

paramGen *Parameter Generators*

Description

Functions which use the arguments of a glarma call to generate the initial delta, theta and phi vectors.

Usage

```
deltaGen(y, X, offset = NULL, phiInit, thetaInit, type, alpha,
         beta, alphaInit)
thetaGen(thetaLags, thetaInit)
phiGen(phiLags, phiInit)
```

Arguments

y	Numeric vector; response variable.
X	Matrix; the explanatory variables. A vector of ones should be added to the data matrix as the first column for the beta of the intercept.
offset	Either NULL or a numeric vector of length equal to the number of cases. Used to specify an *a priori* known component to be included in the linear predictor during fitting.
phiInit	Numeric vector; initial values for the corresponding AR orders.
thetaInit	Numeric vector; initial values for the corresponding orders.
type	Character; the count distribution. The default is the Poisson distribution.
beta	Numeric vector; initial values of the parameters of variables. It is for the user to construct the specific delta by themselves.
alpha	Numeric; an optional initial theta parameter for glm.nb.
alphaInit	Numeric; an initial theta parameter for glarma for negative binomial counts.
thetaLags	Numeric vector; MA orders
phiLags	Numeric vector; AR orders

Details

The thetaGen and phiGen functions take the arguments, thetaLags, phiLags, thetaInit and phiInit, in a glarma call to generate and return the initial theta and phi vectors with orders corresponding to their names. Then the deltaGen function uses the values returned by thetaGen, phiGen and other arguments in the glarma call to generate and return the initial delta vector with correct names.

Value

thetaGen returns a list containing thetaLags and thetaInit. thetaInit is the initial theta vector with its corresponding MA orders as its names.

phiGen returns a list containing phiLags and phiInit. phiInit is the initial phi vector with its corresponding MA orders as its names.

deltaGen returns a named vector giving the values of beta, phiInit, thetaInit plus alpha in the negative binomial case.

Author(s)

"Cenanning Li" <cli113@aucklanduni.ac.nz> and "William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au>

Examples

```
### Using the polio data
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])

## generate the theta vector
theta.lags <- c(1, 2, 5)
theta.init <- c(0.0, 0.0, 0.0)

theta <- thetaGen(theta.lags, theta.init)
print(thetaLags <- theta[[1]])
print(theta.init <- theta[[2]])

## generate the vector of phi
phi.lags <- rep(0, 0)
phi.init <- rep(0, 0)
phi <- phiGen(phi.lags, phi.init)
print(phiLags <- phi[[1]])
print(phi.init <- phi[[2]])

## generate the delta vector
delta <- deltaGen(y = y, X = X, phiInit = phi.init,
                  thetaInit = theta.init, type = "Poi",
                  alpha = 1)
delta
```

PIT *Non-randomized Probability Integral Transformation*

Description

Functions to produce the non-randomized probability integral transform (PIT) to check the adequacy of the distributional assumption of the GLARMA model.

Usage

```
glarmaPredProb(object)
glarmaPIT(object, bins = 10)
```

Arguments

object	An object of class "glarma", obtained from a call to glarma.
bins	Numeric; the number of bins used in the PIT.

Details

These functions are used for the assessment of predictive distributions in discrete data. They obtain the predictive probabilities and the probability integral transformation for a fitted GLARMA model.

Value

glarmaPredProb returns a list with values:

upper	the predictive cumulative probabilities used as the upper bound for computing the non-randomized PIT.
lower	the predictive cumulative probabilities used as the lower bound for computing the non-randomized PIT.

glarmaPIT returns a list with values:

upper	the predictive cumulative probabilities used as the upper bound for computing the non-randomized PIT.
lower	the predictive cumulative probabilities used as the lower bound for computing the non-randomized PIT.
conditionalPIT	the conditional probability integral transformation given the observed counts.
PIT	the probability integral transformation.

Author(s)

"David J. Scott" <d.scott@auckland.ac.nz> and "Cenanning Li" <cli113@aucklanduni.ac.nz>

References

Czado, Claudia and Gneiting, Tilmann and Held, Leonhard (2009) Predictive model assessment for count data. *Biometrics*, **65**, 1254–1261.

Jung, Robert.C and Tremayne, A.R (2011) Useful models for time series of counts or simply wrong ones? *Advances in Statistical Analysis*, **95**, 59–91.

Examples

```
### Example from Davis, Dunsmuir Wang (1999)
## MA(1,2,5), Pearson Residuals, Fisher Scoring
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1,2,5), type = "Poi", method = "FS",
                    residuals = "Pearson", maxit = 100, grad = 2.22e-16)
glarmaPredProb(glarmamod)
glarmaPIT(glarmamod)
```

---

plot.glarma *Plot Diagnostics for a* glarma *Object*

Description

Ten plots (selectable by `which`) are currently available: a time series plot with observed values of the dependent variable, fixed effects fit, and GLARMA fit; an ACF plot of residuals; a plot of residuals against time; a normal Q-Q plot; the PIT histogram; a uniform Q-Q plot for the PIT; a histogram of the normal randomized residuals; a Q-Q plot of the normal randomized residuals; a plot of the autocorrelation of the normal randomized residuals; and a plot of the partial autocorrelation of the normal randomized residuals. By default, six plots are provided, numbers 1, 3, 5, 7, 8 and 9 from this list of plots.

Usage

```
## S3 method for class 'glarma'
plot(x, which = c(1L,3L,5L,7L,8L,9L), fits = 1L:3L,
     ask = prod(par("mfcol")) < length(which) && dev.interactive(),
     lwdObs = 1, lwdFixed = 1, lwdGLARMA = 1,
     colObs = "black", colFixed = "blue", colGLARMA = "red",
     ltyObs = 2, ltyFixed = 1, ltyGLARMA = 1,
     pchObs = 1, legend = TRUE, residPlotType = "h", bins = 10,
     line = TRUE, colLine = "red", colHist = "royal blue",
     lwdLine = 2, colPIT1 = "red", colPIT2 = "black",
     ltyPIT1 = 1, ltyPIT2 = 2, typePIT = "l",
     ltyQQ = 2, colQQ = "black", titles, ...)
```

Arguments

x	An object of class "glarma", obtained from a call to glarma.
which	Numeric; if a subset of the plots is required, specify a subset of the numbers 1:10. 1 is the time series plot with observed values of the dependent variable, fixed effects fit, and GLARMA fit. 2 is the ACF plot of residuals. 3 is a plot of residuals against time. 4 is the normal Q-Q plot. 5 is the PIT histogram. 6 is the uniform Q-Q plot for the PIT. 7 is the histogram of the normal randomized residuals. 8 is the Q-Q plot of the normal randomized residuals. 9 is the autocorrelation of the normal randomized residuals. 10 is the partial autocorrelation of the normal randomized residuals. By default, plots 1, 3, 5, 7, 8 and 9 are provided.
fits	Numeric; if a subset of fits on the time series plot is required, specify a subset of the numbers 1:3. 1 is the observed values of the dependent variable, 2 is the fixed effects fit, and 3 is GLARMA fit. By default, all fits are provided.
ask	Logical; if TRUE, the user is asked before each plot, see par(ask = .).
lwdObs	Numeric; the line widths for lines of the observed values of the dependent variable appearing in the time series plot.
lwdFixed	Numeric; the line widths for lines of the fixed effects fit appearing in the time series plot.
lwdGLARMA	Numeric; the line widths for lines of GLARMA fit appearing in the time series plot.
ltyObs	An integer or character string; the line types for the line of the observed data of the dependent variable appearing in the time series plot, see par(lty = .).
ltyFixed	An integer or character string; the line types for the line of the fixed effects fit appearing in the time series plot, see par(lty = .).
ltyGLARMA	An integer or character string; the line types for the line of GLARMA fit appearing in the time series plot, see par(lty = .).
pchObs	Numeric; the point type for the point of the observed data of the dependent variable appearing in the time series plot.
colObs	Numeric or character; the colour of lines or points of the observed data of the dependent variable appearing in the time series plot.
colFixed	Numeric or character; the colour of lines of the fixed effects fit appearing in the time series plot.
colGLARMA	Numeric or character; the colour of lines of GLARMA fit appearing in the time series plot.
legend	Logical; if TRUE, the legend for the fits in the time series plot would be shown. By default, it would be shown.
residPlotType	A 1-character string giving the type of plot desired. The following values are possible, for details, see plot: "p" for points, "l" for lines, "b" for both points and lines, "c" for empty points joined by lines, "o" for overplotted points and lines, "s" and "S" for stair steps and "h" for histogram-like vertical lines. Finally, "n" does not produce any points or lines.
bins	Numeric; the number of bins shown in the PIT histogram and of the number of breaks in the histogram of the normal randomized residuals. By default, it is 10.

line	Logical; if TRUE, the line for displaying the standard uniform distribution will be shown for the purpose of comparison. The default is TRUE.
colLine	Numeric or character; the colour of the line for comparison in the PIT histogram.
lwdLine	Numeric; the line widths for the comparison line in the PIT histogram.
colHist	Numeric or character; the colour of the histogram for the PIT, and of the histogram of the normal randomized residuals.
colPIT1	Numeric or character; the colour of the sample uniform Q-Q plot in the PIT.
colPIT2	Numeric or character; the colour of the theoretical uniform Q-Q plot in the PIT.
ltyPIT1	An integer or character string; the line types for the sample uniform Q-Q plot in the PIT, see par(lty = .).
ltyPIT2	An integer or character string; the line types for the theoretical uniform Q-Q plot in the PIT, see par(lty = .).
typePIT	A 1-character string; the type of plot for the sample uniform Q-Q plot in the PIT.
ltyQQ	An integer or character string; the line type for the normal Q-Q plot of the normal randomized residuals, see par(lty = .).
colQQ	Numeric or character; the colour of the line in the normal Q-Q plot of the normal randomized residuals.
titles	A list of the same length as which. For any elements which are NULL, useful titles will be created for the corresponding plot.
...	Further arguments passed to plot.default and plot.ts.

Details

plot.glarma is an S3 generic function for objects of class glarma.

The plots in this method display the fixed effects fit, GLARMA fit and various types of residuals for the GLARMA fit under the Poisson distribution, the binomial distribution or the negative binomial distribution, plus a number of plots of the randomized residuals (see normRandPIT for details of the randomized residuals). In all, ten plots can be produced. The observed values of the dependent variable shown in the time series plot are mainly used to compare with the two fits.

The fixed effects fit is calculated from η, the multiplication of the data matrix X and β coefficients in GLARMA model. In contrast, the GLARMA fit is calculated from W, the product of the data matrix X and δ in the GLARMA model, which is the combination of both the β and ARMA coefficients, and is also called the state variable of the series.

There are some major differences for computing the fixed effects fit from η and the GLARMA fit from W under different distributions.

Under the Poisson distribution and negative binomial distribution,

$$\mathsf{fit}_{\mathsf{fixed}} = \exp \eta$$

and

$$\mathsf{fit}_{\mathsf{glarma}} = \exp W.$$

Under the binomial distribution,

$$\mathsf{fit}_{\mathsf{fixed}} = \frac{1}{(1 + e^{-\eta})}$$

and

$$\mathsf{fit}_{\mathsf{glarma}} = \frac{1}{(1+e^{-W})}.$$

The residuals are calculated from the observed data and GLARMA fit. The exact computation for the residuals depends on the type of residuals used. The details are given in glarma. The ACF plot, the residuals against time and the normal Q-Q plot are all based on these residuals. Further details about those three plots are passed to acf and qqnorm.

There are four plots based on the randomized residuals calculated using normRandPIT. These are a histogram, a Q-Q plot, an autocorrelation plot and a partial autocorrelation plot.

The number of plots to be shown in the window depends on the value of the graphical parameter mfrow (or mfcol). If the displayed window is set to be large enough to show all ten plots, they will be shown at one time. Otherwise, the required number of plots will appear each time in the displayed window, and the user will need to enter return to see subsequent plots. By default, six plots are produced.

For the time series plot in the function, the fit displayed is specified by the argument fits. The legend will be shown if legend is TRUE. It will appear under the title of the time series plot. Also the legend and the title will alter automatically according to the fits shown in the plot.

Author(s)

"Cenanning Li" <cli113@aucklanduni.ac.nz>

See Also

plot.ts, qqnorm, acf, plot.default, normRandPIT.

Examples

```
### A example from Davis, Dunsmuir Wang (1999)
## MA(1,2,5), Pearson Residuals, Fisher Scoring
data(Polio)
y <- Polio[, 2]
X <- as.matrix(Polio[, 3:8])
glarmamod <- glarma(y, X, thetaLags = c(1, 2, 5), type = "Poi",method = "FS",
                    residuals = "Pearson", maxit = 100 , grad = 1e-6)

## The default plots are shown
plot(glarmamod)

## The plots used only to compared GLARMA fit and the observed data
plot(glarmamod, which = 1L, fits = c(1, 3))
```

plotPIT *PIT Plots for a* glarma *Object*

Description

Two plots for the non-randomized PIT are currently available for checking the distributional assumption of the fitted GLARMA model: the PIT histogram, and the uniform Q-Q plot for PIT.

Usage

```
histPIT(object, bins = 10, line = TRUE, colLine = "red",
        colHist = "royal blue", lwdLine = 2, main = NULL, ...)
qqPIT(object, bins = 10, col1 = "red", col2 = "black",
      lty1 = 1, lty2 = 2, type = "l", main = NULL, ...)
```

Arguments

object	An object of class "glarma", obtained from a call to glarma.
bins	Numeric; the number of bins shown in the PIT histogram or the PIT Q-Q plot. By default, it is 10.
line	Logical; if TRUE, the line for displaying the standard uniform distribution will be shown for the purpose of comparison. The default is TRUE.
colLine	Numeric or character; the colour of the line for comparison in PIT histogram.
lwdLine	Numeric; the line widths for the comparison line in PIT histogram.
colHist	Numeric or character; the colour of the histogram for PIT.
col1	Numeric or character; the colour of the sample uniform Q-Q plot in PIT.
col2	Numeric or character; the colour of the theoretical uniform Q-Q plot in PIT.
lty1	An integer or character string; the line types for the sample uniform Q-Q plot in PIT, see par(lty = .).
lty2	An integer or character string; the line types for the theoretical uniform Q-Q plot in PIT, see par(lty = .).
type	A 1-character string; the type of plot for the sample uniform Q-Q plot in PIT.
main	A character string giving a title. For each plot the default provides a useful title.
...	Further arguments passed to plot.default and plot.ts.

Details

The histogram and the Q-Q plot are used to compare the fitted profile with **U**(0, 1). If they match relatively well, it means the distributional assumption is satisfied.

Author(s)

"David J. Scott" <d.scott@auckland.ac.nz> and "Cenanning Li" <cli113@aucklanduni.ac.nz>

References

Czado, Claudia and Gneiting, Tilmann and Held, Leonhard (2009) Predictive model assessment for count data. *Biometrics*, **65**, 1254–1261.

Jung, Robert.C and Tremayne, A.R (2011) Useful models for time series of counts or simply wrong ones? *AStA Advances in Statistical Analysis*, **95**, 59–91.

Examples

```
## For examples see example(plot.glarma)
```

`Polio` *Cases of Poliomyelitis in the U.S.*

Description

This data set gives the monthly number of cases of poliomyelitis in the U.S. for the years 1970–1983 as reported by the Center for Disease Control. The polio data frame has 168 rows and 8 columns.

Usage

```
data(Polio)
```

Format

A data frame containing the following columns:

[, 1]	Cases	monthly number of cases of poliomyelitis.
[, 2]	Intcpt	a vector of ones, providing the intercept in the model.
[, 3]	Trend	a linear trend.
[, 4]	CosAnnual	cosine harmonics at periods of 12.
[, 5]	SinAnnual	sine harmonics at periods of 12.
[, 6]	CosSemiAnnual	cosine harmonics at periods of 6.
[, 7]	SinSemiAnnual	sine harmonics at periods of 6.

Source

Zeger, S.L (1988) A regression model for time series of counts. *Biometrika*, **75**, 621–629.

residuals.glarma *Extract GLARMA Model Residuals*

Description

`residuals` is a generic function which extracts model residuals from objects returned by the modeling function `glarma`. `resid` is an alias for `residuals`.

Usage

```
## S3 method for class 'glarma'
residuals(object, ...)
```

Arguments

`object`	An object of class `"glarma"`, a result of a call to `glarma`.
`...`	Further arguments passed to or from other methods.

Value

Residuals extracted from the object `object`.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "Cenanning Li" <cli113@aucklanduni.ac.nz>

See Also

`coefficients.glarma`, `fitted.glarma`, `glarma`.

RobberyConvict *Court Convictions for Armed Robbery in New South Wales*

Description

Monthly counts of charges laid and convictions made in Local Courts and Higher Court in armed robbery in New South Wales from 1995–2007.

Usage

```
data(RobberyConvict)
```

Format

A data frame containing the following columns:

[, 1]	Date	Date in month/year format.
[, 2]	Incpt	A vector of ones, providing the intercept in the model.
[, 3]	Trend	Scaled time trend.
[, 4]	Step.2001	Unit step change from 2001 onwards.
[, 5]	Trend.2001	Change in trend term from 2001 onwards.
[, 6]	HC.N	Monthly number of cases for robbery (Higher Court).
[, 7]	HC.Y	Monthly number of convictions for robbery (Higher court).
[, 8]	HC.P	Proportion of convictions to charges for robbery (Higher court).
[, 9]	LC.N	Monthly number of cases for robbery (Lower court).
[, 10]	LC.Y	Monthly number of convictions for robbery (Lower court).
[, 11]	LC.P	Proportion of convictions to charges for robbery (Lower court).

Source

Dunsmuir, William TM, Tran, Cuong, and Weatherburn, Don (2008) *Assessing the Impact of Mandatory DNA Testing of Prison Inmates in NSW on Clearance, Charge and Conviction Rates for Selected Crime Categories.*

Examples

```
### Example with Robbery Convictions
data(RobberyConvict)
datalen <- dim(RobberyConvict)[1]
monthmat <- matrix(0, nrow = datalen, ncol = 12)
dimnames(monthmat) <- list(NULL, c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                                   "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"))
months <- unique(months(strptime(RobberyConvict$Date, format = "%m/%d/%Y"),
                        abbreviate=TRUE))


for (j in 1:12) {
  monthmat[months(strptime(RobberyConvict$Date,  "%m/%d/%Y"),
                  abbreviate = TRUE) == months[j], j] <- 1
}

RobberyConvict <- cbind(rep(1, datalen), RobberyConvict, monthmat)
rm(monthmat)

## LOWER COURT ROBBERY
y1 <- RobberyConvict$LC.Y
n1 <- RobberyConvict$LC.N

Y <- cbind(y1, n1-y1)

glm.LCRobbery <- glm(Y ~ Step.2001 +
                        I(Feb + Mar + Apr + May + Jun + Jul) +
                        I(Aug + Sep + Oct + Nov + Dec),
```

```
                    data = RobberyConvict, family = binomial(link = logit),
                    na.action = na.omit, x = TRUE)

summary(glm.LCRobbery, corr = FALSE)

X <- glm.LCRobbery$x


## Newton Raphson
glarmamod <- glarma(Y, X, phiLags = c(1), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)

## LRT, Wald tests.
likTests(glarmamod)

## Residuals and Fit Plots
plot.glarma(glarmamod)


## HIGHER COURT ROBBERY
y1 <- RobberyConvict$HC.Y
n1 <- RobberyConvict$HC.N

Y <- cbind(y1, n1-y1)

glm.HCRobbery <- glm(Y ~ Trend + Trend.2001 +
                       I(Feb + Mar + Apr + May + Jun) + Dec,
                     data = RobberyConvict, family = binomial(link = logit),
                     na.action = na.omit, x = TRUE)

summary(glm.HCRobbery,corr = FALSE)

X <- glm.HCRobbery$x


## Newton Raphson
glarmamod <- glarma(Y, X, phiLags = c(1, 2, 3), type = "Bin", method = "NR",
                    residuals = "Pearson", maxit = 100, grad = 1e-6)
glarmamod
summary(glarmamod)


## LRT, Wald tests.
likTests(glarmamod)

## Residuals and Fit Plots
plot.glarma(glarmamod)
```

---

summary.glarma *Summarize GLARMA Fit*

Description

summary method for class glarma and functions to generate the estimates for this summary method.

Usage

```
## S3 method for class 'glarma'
summary(object, tests = TRUE, ...)
## S3 method for class 'summary.glarma'
print(x, digits = max(3L, getOption("digits") - 3L), ...)
glarmaModelEstimates(object)
```

Arguments

object	An object of class "glarma", obtained from a call to glarma.
x	An object of class "summary.glarma", obtained from a call to summary.glarma.
digits	Numeric; minimum number of significant digits to be used for most numbers.
tests	Logical; if TRUE, the likelihood-ratio test and the Wald test are shown in the summary. The default is TRUE.
...	Further arguments passed to or from other methods.

Value

summary.glarma returns an object of class "summary.glarma", a list with components

call	the component from object
null.deviance	null deviance of the GLM with the same regression structure as the GLARMA model.
df.null	null degrees of freedom of the GLM with the same regression structure as the GLARMA model.
phi.lags	the component from object.
theta.lags	the component from object.
pq	the component from object.
iter	the component from object.
deviance	the deviance of the fitted model.
df.residual	the degrees of freedom of the fitted model.
deviance.resid	the component from object.
aic	the component from object.
methods	vector specifying the count distribution of the GLARMA model, the iteration method and the type of residual used.
tests	whether tests were asked for.
likTests	if tests is TRUE, the result of a call to likTests, NULL otherwise.
coefficients1	the matrix of beta coefficients, standard errors, z-ratio and p-values.
coefficients2	the matrix of ARMA coefficients, standard errors, z-ratio and p-values.
coefficients3	when the count distribution is negative binomial, a matrix with 1 row, giving the negative binomial parameter, its standard error, z-ratio and p-value.

Author(s)

"William T.M. Dunsmuir" <w.dunsmuir@unsw.edu.au> and "Cenanning Li" <cli113@aucklanduni.ac.nz>

See Also

glarma, summary.

Examples

```
## For examples see example(glarma)
```

Index