Package 'VGAMextra'

October 12, 2022

Version 0.0-5

Date 2021-05-24

Title Additions and Extensions of the 'VGAM' Package

Author Victor Miranda [aut, cre, cph],
Thomas Yee [ctb, ths, cph]

Maintainer Victor Miranda <victor.miranda@aut.ac.nz>

Depends R (>= 3.5.0), methods, stats, stats4, VGAM (>= 1.1-0)

Suggests VGAMdata

Description Extending the functionalities of the 'VGAM' package with additional functions and datasets. At present, 'VGAMextra' comprises new family functions (ffs) to estimate several time series models by maximum likelihood using Fisher scoring, unlike popular packages in CRAN relying on optim(), including ARMA-GARCH-like models, the Order-(p, d, q) ARIMAX model (non-seasonal), the Order-(p) VAR model, error correction models for cointegrated time series, and ARMA-structures with Student-t errors. For independent data, new ffs to estimate the inverse-Weibull, the inverse-gamma, the generalized beta of the second kind and the general multivariate normal distributions are available. In addition, 'VGAMextra' incorporates new VGLM-links for the mean-function, and the quantile-function (as an alternative to ordinary quantile modelling) of several 1-parameter distributions, that are compatible with the class of VGLM/VGAM family functions. Currently, only fixed-effects models are implemented. All functions are subject to change; see the NEWS for further details on the latest changes.

License GPL-2

NeedsCompilation yes

BuildVignettes yes

Repository CRAN

Date/Publication 2021-05-24 04:10:07 UTC

R topics documented:

`VGAMextra-package` *Additions and extensions of the VGAM package.*

Description

VGAMextra supplies additional functions and methods to the package **VGAM**, under three main topics:

** *Time series modelling*. A novel class of VGLMs to model univariate time series, called *vector generalized linear time series models* (VGLTSMs). It is characterized by incorporating *past information* in the VGLM/VGAM loglikelihood.

** *1–parameter distribution mean modelling*. We return full circle by developing new link functions for the mean of 1–parameter distributions. VGAMs, VGLMs and GAMLSSs are restricted to location, scale and shape, however, the VGLM/VGAM framework has infrastructure to accommodate new links as a function of the parameters.

** *Quantile modelling of 1–parameter distributions*. Similarly, we have implemented link functions to model the quantiles of several 1–parameter distributions, for *quantile regression*.

Details

The inference infrastructure of **VGAMextra** relies on the VGLM/VGAM framework. Particularly, estimation is carried out via IRLS using Fisher scoring, whilst additive models and reduced rank regression are also accommodated by all **VGAMextra** family functions.

At present, this package allows the extent of VGLMs/VGAMs to operate popular time series models as special cases, as well as cointegrated time series (bivariate case), and modelling choices for volatility models incorporating explanatories in the variance equation. The central family functions in this respect are `ARXff`, `MAXff`, `ARMAX.GARCHff`, and `VGLM.INGARCHff`.

Regarding modelling the mean/quantile-functions, **VGAMextra** affords links for several 1-parameter distributions, e.g., `expMlink`, `benini1Qlink`, or `inv.chisqMlink`. Collectively, the quantile-links represent an alternative to quantile regression by directly modelling the quantile function for distributions beyond the exponential family (See Example 3 below).

The VGLM/VGAM framework is very large and encompasses a wide range of multivariate response types and models, e.g., it includes univariate and multivariate distributions, categorical data analysis, and extreme values. See `VGAM-package` for a broad description and further information on **VGAM**.

Future work

* Implement VGLM time series family functions to handle error correction models (ECMs) for cointegrated time series. Upgrade this framework beyond the bivariate case, e.g., the the Vector ECM (VECMs).

* Upgrade VGLMs time series family functions to handle multivariate time series, e.g., the VAR model (Coming shortly).

* Incorporate VGLM/VGAM-links to model the mean and quantile functions of distributions with > 1 parameters.

* Develop the class of *multiple* reduced–rank VGLMs towards time series, to handle, e.g., vector error correction models (VECMs), for multiple cointegrated time series.

Warning

VGAMextra is revised, altered, and/or upgraded on a regular basis. Hence, be aware that any feature, e.g., function names, arguments, or methods, may be modified without prior notice. Check the `NEWS` for the latest changes and additions across the different versions.

Author(s)

Victor Miranda, <victor.miranda@aut.ac.nz>.

Maintainer: Victor Miranda, <victor.miranda@aut.ac.nz>.

Contributor: Thomas Yee, <t.yee@auckland.ac.nz>.

References

Pfaff, B. (2011) Analysis of Integrated and Cointegrated Time Series with R. Seattle, Washington, USA: *Springer*.

Chan, N., Li, D., Peng, L. and Zhang, R. (2013) Tail index of an AR(1) model with ARCH(1) errors. *Econometric Theory*, 29(5):920-940.

Yee, T. W. (2015) Vector Generalized Linear and Additive Models: With an Implementation in R. New York, USA: *Springer*.

Miranda, V. and Yee, T. W. *Vector generalized linear time series models.* In preparation.

Miranda, V. and Yee, T. W. *On mean modelling of 1-parameter distributions using vector generalized linear models*. In preparation

Miranda, V. and Yee, T. W. *Two–Parameter Link Functions with Applications to Negative Binomail, Weibull, and Quantile Regression*. To be submitted to the Scandinavian Journal of Statistics.

Miranda, V. and Yee, T.W. *New Link Functions for Distribution-Specific Quantile Regression Based on Vector Generalized Linear and Additive Models*. Journal of Probability and Statistics, Volume 2019, Article ID 3493628.

Yee, T. W. (2008) The `VGAM` Package. *R News*, **8**, 28–39.

See Also

`vglm`, `vgam`, `rrvglm`, `Links`, `CommonVGAMffArguments`, https://www.stat.auckland.ac.nz/~vmir178/.

Examples

```
##### EXAMPLE 1. An AR(1) model with ARCH(1) errors.
# Chan et.al. (2013) proposed a long and technical methodology to
# estimate the tail index of an AR(1) with ARCH(1) errors involving
# its estimation by QMLE. I fit this model straightforwardly by MLE
# using the family function ARXff() for time series, and constraining
# the effect of Y^2_{t - 1} to the conditional variance using
# constraint matrices.


# Generate some data
set.seed(1)
nn       <- ceiling(runif(1, 150, 160))
my.rho   <- rhobitlink(-1.0, inverse = TRUE)  # -0.46212
my.mu    <- 0.0
my.omega <- 1
my.b     <- 0.5
tsdata   <- data.frame(x2 =  sort(runif(n = nn)))
tsdata   <- transform(tsdata, index = 1:nn, TS1   = runif(nn))

for (ii in 2:nn)
  tsdata$TS1[ii] <- my.mu + my.rho * tsdata$TS1[ii-1] +
  sqrt(my.omega + my.b * (tsdata$TS1[ii-1])^2) * rnorm(1)

# Remove the burn-in data:
nnr <- ceiling(nn/5)
tsdata <- tsdata[-(1:nnr), ]
tsdata["index"] <- 1:(nn - nnr)


# The timeplot.
with(tsdata, plot(ts(TS1), lty = "solid", col = "blue", xlab ="Time", ylab = "Series"))
abline(h = mean(tsdata[, "TS1"]), lty = "dotted")

# The constraint matrices, to inhibit the effect of Y^2_{t - 1}
# over sigma^2 only.
const.mat <- list('(Intercept)' = diag(3), 'TS1l1sq' = cbind(c(0, 1, 0)))

# Set up the data using function WN.lags() from VGAMextra to generate
# our 'explanatory'
tsdata <- transform(tsdata, TS1l1sq = WN.lags(y = cbind(tsdata[, "TS1"])^2, lags = 1))

# Fitting the model
fit.Chan.etal <- vglm(TS1 ~ TS1l1sq, ARXff(order = 1,   # AR order
                                           zero = NULL, noChecks = FALSE,
                                           var.arg = TRUE, lvar = "identitylink"),
             crit = "loglikelihood", trace = TRUE,
             constraints = const.mat, data = tsdata)  ## Constraints...
summary(fit.Chan.etal, lrt0 = TRUE, score0 = TRUE, wald0 = TRUE)
constraints(fit.Chan.etal)
```

```
###### EXAMPLE 2. VGLMs handling cointegrated (bivariate) time series.
# In this example, vglm() accommodates an error correction model
# of order (2, 2) to fit two (non-stationary) cointegrated time series.

# Simulating some data.
set.seed(2017081901)
nn <- 280
rho <- 0.75
s2u <- exp(log(1.5))  # Gaussian noise1
s2w <- exp(0)         # Gaussian noise2
my.errors <- rbinorm(nn, mean1 = 0, mean2 = 0, var1 = s2u, var2 = s2w, cov12 = rho)
ut <- my.errors[, 1]
wt <- my.errors[, 2]
yt <- xt <- numeric(0)

xt[1] <- ut[1]     # Initial value: error.u[0]
yt[1] <- wt[1]     # Initial value: error.w[0]
beta <- c(0.0, 2.5, -0.32)  # Coefficients true values.

for (ii in 2:nn) {
  xt[ii] <-  xt[ii - 1] + ut[ii]
  yt[ii] <- beta[1] + beta[2] * xt[ii]  + beta[3] * yt[ii - 1] + wt[ii]
}

# Regression of yt on xt, save residuals. Compute Order--1 differences.
errors.coint <- residuals(lm(yt ~ xt)) # Residuals from the static regression yt ~ xt
difx1 <- diff(ts(xt), lag = 1, differences = 1)  # First difference for xt
dify1 <- diff(ts(yt), lag = 1, differences = 1)  # First difference for yt

# Set up the dataset (coint.data), including Order-2 lagged differences.
coint.data <- data.frame(embed(difx1, 3), embed(dify1, 3))
colnames(coint.data) <- c("difx1", "difxLag1", "difxLag2",
                          "dify1", "difyLag1", "difyLag2")

# Remove unutilized lagged errors accordingly. Here, use from t = 3.
errors.cointLag1 <- errors.coint[-c(1:2, nn)]
coint.data <- transform(coint.data, errors.cointLag1 = errors.cointLag1)


# Plotting the data
plot(ts(yt[-c(1:3, NULL)]), lty = 4, type = "l", col = "red",
     main = "", xlab = "Time", las = 1, ylim = c(-32, 20),
     ylab = expression("Series"~x[t]~"and"~y[t]))
lines(ts(xt[-c(1:3, NULL)]), lty = 4, type = "l", col = "blue")
legend("bottomleft", c(expression("Series"~x[t]),
                       expression("Series"~y[t])),
       col = c("red", "blue"), lty = c(4, 4))


# Fitting an error correction model (2, 2), aka ECM(2, 2)
fit.coint <- vglm(cbind(dify1, difx1) ~ errors.cointLag1 + difxLag1 + difyLag1 +
                                        difxLag2 + difyLag2,
          binormal(zero = c("sd", "rho")),  # 'sigma', 'rho' are intercept--only.
```

```
          trace = FALSE, data = coint.data)
summary(fit.coint)
coef(fit.coint, matrix = TRUE)


##### EXAMPLE 3. Quantile Modelling (QM).
# Here, the quantile function of the Maxwell distribution is modelled
# for percentiles 25%, 50% and 75%. The resulting quantile-curves
# are plotted. The rate parameter is determined by an artificial covariate.

set.seed(123)
# An artificial covariate.
maxdata <- data.frame(x2 = sort(runif(n <- nn <- 120)))
# The 'rate' function.
mymu <- function(x) exp(2 -  6 * sin(2 * x - 0.2) / (x + 0.5)^2)
# Set up the data.
maxdata <- transform(maxdata, y = rmaxwell(n, rate = mymu(x2)))

# 25%, 50% and 75% quantiles are to be modelled.
mytau <- c(0.25, 0.50, 0.75)
mydof <- 4

### Using B-splines with 'mydof'-degrees of freedom on the predictors
fit.QM <- vglm(Q.reg(y, pvector = mytau) ~ bs(x2, df = mydof),
                family = maxwell(link = maxwellQlink(p = mytau), zero = NULL),
                data = maxdata, trace = TRUE)

summary(fit.QM, lscore0 = TRUE)
head(predictors(fit.QM))         # The 'fitted values'


## The 25%, 50%, and 75% quantile curves.
mylwd <- 1.5
with(maxdata, plot(x2, y, col = "orange",
                main = "Example 1; Quantile Modelling",
                ylab = "y", pch = "o", cex = 0.75))
with(maxdata, matlines(x2, predict(fit.QM)[, 1], col = "blue",
                       lty = "dotted", lwd = mylwd))
with(maxdata, matlines(x2, predict(fit.QM)[, 2], col = "chocolate",
                       lty = "dotted", lwd = mylwd))
with(maxdata, matlines(x2, predict(fit.QM)[, 3], col = "brown",
                       lty = "dotted", lwd = mylwd))
legend("topleft", c("percentile25", "percentile50", "percentile75"),
       lty = rep("dotted", 3), lwd = rep(mylwd, 3))


### Double check: The data (in percentage) below the 25%, 50%, and 75% curves
round(length(predict(fit.QM)[, 1][(maxdata$y
                <= predict(fit.QM)[, 1] )]) /nn, 5) * 100  ## Should be 25% approx
round(length(predict(fit.QM)[, 2][(maxdata$y
                <= predict(fit.QM)[, 2] )]) /nn, 5) * 100  ## Should be 50% approx
round(length(predict(fit.QM)[, 3][(maxdata$y
                <= predict(fit.QM)[, 3] )]) /nn, 5) * 100  ## Should be 75% approx
```

ap.mx *Air pollution Data, Mexico City.*

Description

Daily air pollution levels in Mexico City, January 2004 – June 2005.

Usage

```
data(ap.mx)
```

Format

This data frame stores time series vectors with the following information:

time Time vector.

PM10 24–hr average concentration of PM_{10}, in micrograms per milliliter.

O3 Daily maximum 8–hour moving average of ozone, in micrograms per milliliter.

temp Daily mean average of temperature, in celsius degrees.

HR Daily mean average (%) of relative humidity.

Details

These are readings of PM_{10}, O_3, temperature and humidity between 1 January 2004 and 30 June 2005 in Mexico City Metropolitan Area. Each observation is the 24–hr mean average (between 00:00 and 23:59 hrs), except for ozone, where the maximum over all the sliding 8–hour–windows, between 00:00 and 23:59 hrs is reported, viz. the daily maximum 8–hour moving average.

Source

National Institute of Ecology. Gathers and disseminates the data generated by the central air quality monitoring network in Mexico City. Website: https://www.gob.mx/inecc/

Examples

```
data(ap.mx)
summary(ap.mx[, -1])
class(ap.mx[, "PM10"])

layout(matrix(c(1, 1, 2,3), 2, 2, byrow = TRUE))
plot.ts(ts(ap.mx$PM10), ylab = expression(PM[10]~"Series"),
        col = "brown", xaxt = "n", las = 1)
xtick <- c(1, 92, 183, 275, 367, 457, 518)
xtext <- c("Jan/04", "April/04", "July/04", "Oct/04", "Jan/05",
           "April/05", "June/05")
axis(side = 1, at = xtick, labels = FALSE)
text(x = xtick, par("usr")[3], labels = xtext,
     pos = 1, xpd = TRUE, col = "black")
```

```
pacf(ap.mx$PM10, main = "", ylim= c(-0.5, 1), lag.max = 60, las = 1)
acf(ap.mx$PM10, main = "", ylim= c(-0.5, 1), lag.max = 60, las = 1)
```

`ARIMAX.errors.ff` *VGLTSMs Family Functions: Generalized integrated regression with order–(p, q) ARMA errors*

Description

A VLTSMff for dynamic regression. Estimates regression models with order–(p, d, q) ARIMA errors by maximum likelihood.

Usage

```
ARIMAX.errors.ff(order = c(1, 1, 1),
                 zero = "var",  # optionally, "mean".
                 order.trend = 0,
                 include.int = TRUE,
                 diffCovs  = TRUE,
                 xLag = 0,
                 include.currentX = TRUE,
                 lvar = "loglink",
                 lmean = "identitylink")
```

Arguments

`order`	The usual (p, d, q) integer vector as in, e.g., `ARIMAXff`. By default, an order–(p, q) ARMA model is fitted on the errors, whlist d is the degree of differencing on the response.
`zero`	What linear predictor is modelled as intercept–only? See `zero` and `CommonVGAMffArguments` for further details.
`order.trend`	Non–negative integer. Allows to incorporate a polynomial trend of order `order.trend` in the forecast mean function.
`include.int`	Logical. Should an intercept (int) be included in the model for y_t? Default is `TRUE`. See below for details.
`diffCovs`	Logical. If `TRUE` (default) the order–`d` difference of the covariates is internally computed and then incorporated in the regression model. Else, only the current values are included.
`xLag`	Integer. If entered, the covariates, say $\boldsymbol{x}_t$ are laggeg (up to order `xLag`) and then embedded in the regression model. See below for further details.
`include.currentX`	Logical. If `TRUE`, the *actual* values, $\boldsymbol{x}_t$, are included in the regression model. Else, this is ignored and only the lagged $\boldsymbol{x}_{t-1}, \ldots, \boldsymbol{x}_{t-xLag}$ will be included.
`lvar, lmean`	Link functions applied to conditional mean and the variance. Same as `uninormal`.

Details

The generalized linear regression model with ARIMA errors is another subclass of VGLTSMs (Miranda and Yee, 2018).

For a univariate time series, say y_t, and a p–dimensional vector of covariates $\boldsymbol{x}_t$ covariates, the model described by this VGLTSM family function is

$$y_t = \boldsymbol{\beta}^T \boldsymbol{x}_t + u_t,$$

$$u_t = \theta_1 u_{t-1} + \cdots + \theta_p u_{t-p} + z_t + \phi_1 z_{t-1} + \cdots + \phi_1 z_{t-q}.$$

The first entry in x_t equals 1, allowing an intercept, for every t. Set `include.int = FALSE` to set this to zero, dimissing the intercept.

Also, if `diffCovs = TRUE`, then the differences up to order `d` of the set $\boldsymbol{x}_t$ are embedded in the model for y_t. If `xLag`> 0, the lagged values up to order `xLag` of the covariates are also included.

The random disturbances z_t are by default handled as $N(0, \sigma_z^2)$. Then, denoting Φ_t as the history of the process (x_{t+1}, u_t) up to time t, yields

$$E(y_t|\Phi_{t-1}) = \boldsymbol{\beta}^T \boldsymbol{x}_t + \theta_1 u_{t-1} + \cdots + \theta_p u_{t-p} + \phi_1 z_{t-1} + \cdots + \phi_1 z_{t-q}.$$

Denoting $\mu_t = E(y_t|\Phi_{t-1})$, the default linear predictor for this VGLTSM family function is

$$\boldsymbol{\eta} = (\mu_t, \log \sigma_z^2)^T.$$

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Note

If `d = 0` in `order`, then `ARIMAX.errors.ff` will perform as `ARIMAXff`.

Author(s)

Victor Miranda

See Also

`ARIMAXff`, `CommonVGAMffArguments`, `uninormal`, `vglm`.

Examples

```
### Estimate a regression model with ARMA(1, 1) errors.
## Covariates are included up to lag 1.
set.seed(20171123)
nn <- 250
x2 <- rnorm(nn)                                    # One covariate
sigma2 <- exp(1.15); theta1 <- 0.5; phi1 <- 0.27   # True coefficients
beta0  <- 1.25; beta1 <- 0.25; beta2 <- 0.5
```

```
y <- numeric(nn)
u <- numeric(nn)
z <- numeric(nn)

u[1] <- rnorm(1)
z[1] <- rnorm(1, 0, sqrt(sigma2))

for(ii in 2:nn) {
  z[ii] <- rnorm(1, 0, sqrt(sigma2))
  u[ii] <- theta1 * u[ii - 1] + phi1 * z[ii - 1] + z[ii]
  y[ii] <- beta0 + beta1 * x2[ii] + beta2 * x2[ii - 1] + u[ii]
}

# Remove warm-up values.
x2 <- x2[-c(1:100)]
y  <- y[-c(1:100)]


plot(ts(y), lty = 2, col = "blue", type = "b")
abline(h = 0, lty = 2)


## Fit the model.
ARIMAX.reg.fit <- vglm(y ~ x2, ARIMAX.errors.ff(order = c(1, 0, 1), xLag = 1),
             data = data.frame(y = y, x2 = x2), trace = TRUE)
coef(ARIMAX.reg.fit, matrix = TRUE)
summary(ARIMAX.reg.fit, HD = FALSE)


# Compare to arima()
# arima() can't handle lagged values of 'x2' by default, but these
# may entered at argument 'xreg'.
arima(y, order = c(1, 0, 1), xreg = cbind(x2, c(0, x2[-150])))
```

ARIMAXff *VGLTSMs Family functions: The Order–(p, d, q) Autoregressive Integrated Moving Average Model (ARIMA(p, d, q)) with covariates*

Description

Maximum likelihood estimation of the drift, standard deviation or variance of the random noise, and coefficients of an autoregressive integrated moving average process of order-(p, d, q) with covariates by MLE using Fisher scoring. No seasonal terms handled yet. No seasonal components handled yet.

Usage

```
ARIMAXff(order   = c(1, 1, 0),
         zero    = c("ARcoeff", "MAcoeff"),
         diffCovs = TRUE,
         xLag    = 0,
         include.current = FALSE,
         type.EIM = c("exact", "approximate")[1],
         var.arg  = TRUE,
         nodrift  = FALSE,
         noChecks = FALSE,
         ldrift   = "identitylink",
         lsd      = "loglink",
         lvar     = "loglink",
         lARcoeff = "identitylink",
         lMAcoeff = "identitylink",
         idrift   = NULL,
         isd      = NULL,
         ivar     = NULL,
         iARcoeff = NULL,
         iMAcoeff = NULL)
```

Arguments

`order`	Integer vector with three components, (`p, d, q`): The AR order (`p`), the degree of differencing (`d`), and the MA order (`q`).
`zero`	Integer or character–strings vector. Name(s) or position(s) of the parameters/linear predictors to be modeled as intercept-only. Details at `zero`.
`diffCovs`	Logical. The default is `diffCovs = TRUE`, which means that the order–`d` differences of the covariates (if entered) are internally computed and then incorporated in the conditional–mean model. Otherwise, only the current `actual` values of the covariates are included.
`xLag`	Integer, non–negative. If `xLag` > 0, the covariates at *current* time, $\boldsymbol{x}_t$, plus the lagged covariates up to order `'xLag'` are embedded into the design matrix (as covariates too). Leave `xLag = 0` and only the *current* value, $\boldsymbol{x}_t$, will be considered. See more details below.
`include.current`	Logical. Same as `ARIMAX.errors.ff`.
`type.EIM`	The type of expected information matrix (EIM) of the ARMA process to be utilized in Fisher scoring. `type.EIM = "exact"` (default) enables the *exact* IM (Porat, et.al. 1986), otherwise the `approximate` version is utilized.
`var.arg`	Logical. If `FALSE` (default), then the standard deviation of the random noise is estimated. Else, the variance estimate is returned.
`nodrift`	Logical. `nodrift = TRUE` supresses estimation of the intercept (the *drift* in the ARMA case), which is set to zero internally.
`noChecks`	Logical. If `FALSE` (default), this family function internally checks *stationarity* (AR case) and *invertibility* (MA case) of the the estimated model. A `warning` is correspondingly displayed.

`ldrift, lsd, lvar, lARcoeff, lMAcoeff`
: Link functions applied to the intercept, the random noise standard deviation (or optionally, the variance), and the coefficients in the ARMA–type conditional–mean model.

`idrift, isd, ivar, iARcoeff, iMAcoeff`
: Optional initial values for the intercept (drift), noise SD (or variance), and ARMA coeffcients (a vector of length $p + q$). If failure to converge occurs then try different values and monitor convergence by using `trace = TRUE` in the `vglm()` call.

Details

Let $\boldsymbol{x}_t$ be a (probably time–varying) vector of suitable covariates. The ARIMAX model handled by `ARIMAXff` is

$$\nabla^d Y_t = \mu^\star + \boldsymbol{\beta}^T \nabla^d \boldsymbol{x}_t + \theta_1 \nabla^d Y_{t-1} + \cdots + \theta_p \nabla^d Y_{t-p} + \phi_1 \varepsilon_{t-1} + \cdots + \phi_q \varepsilon_{t-q} + \varepsilon,$$

with $\nabla^d(\cdot)$ the operator differencing a series `d` times. If `diffCovs = TRUE`, this function differencing the covariates `d` times too.

Similarly, `ARMAXff` manages

$$\nabla^d Y_t = \mu^\star + \boldsymbol{\beta}^T \boldsymbol{x}_t + \theta_1 Y_{t-1} + \cdots + \theta_p Y_{t-p} + \phi_1 \varepsilon_{t-1} + \cdots + \phi_q \varepsilon_{t-q} + \varepsilon,$$

where

$$\varepsilon_{t|\Phi_{t-1}} \sim N(0, \sigma^2_{\varepsilon_t|\Phi_{t-1}}).$$

Note, $\sigma^2_{\varepsilon|\Phi_{t-1}}$ is *conditional* on Φ_{t-1}, the information of the joint process $(Y_{t-1}, \boldsymbol{x}_t)$, at time t, and hence may be modelled in terms of $\boldsymbol{x}_t$, if required.

`ARIMAXff()` and `ARMAXff()` handle *multiple responses*, thus a matrix can be used as the response. Note, no seasonal terms handled. This feature is to be incorporated shortly.

The default linear predictor is

$$\boldsymbol{\eta} = \left(\mu, \log \sigma^2_{\varepsilon_t|\Phi_{t-1}}, \theta_1, \ldots, \theta_p, \phi_1, \ldots, \phi_q\right)^T.$$

Other links are also handled. See `Links`.

Further choices for the random noise, besides Gaussian, will be implemented over time.

As with `ARXff` and `MAXff`, choices for the EIMs are `"exact"` and `"approximate"`. Covariates may be incorporated in the fit for any linear predictor above. Hence, `ARIMAXff` supports non–stationary processes ($\sigma^2_{\varepsilon_t|\Phi_{t-1}}$) may depend on $\boldsymbol{X}_t$. Also, constraint matrices on the linear predictors may be entered through `cm.ARMA` or using the argument `constraints`, from `vglm`.

Checks on stationarity and invertibility on the esitmated process are performed by default. Set `noChecks = TRUE` to dismiss this step.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Warning

`zero` can be a **numeric** or a **character–strings** vector specifying the position(s) or the name(s) of the parameter(s) modeled as intercept–only. Numeric values can be set as usual (See `CommonVGAMffArguments`). If names are entered, the parameter names in this family function are:

`c("drift.mean", "noiseVar" || "noiseSD", "ARcoeff", "MAcoeff")`.

Manually modify this if required. For simplicity, the second choice is recommended.

Note

No seasonal components handled yet.

If no covariates, $\boldsymbol{x}_t$, are incorporated in the analysis, then `ARIMAXff` fits an ordinary ARIMA model. Ditto with `ARMAXff`.

If `nodrift = TRUE`, then the 'drift' is removed from the vector of parameters and is not estimated.

By default, an ARMA model of order–$c(1, 0)$ with order–1 differences is fitted. When initial values are entered (`isd`, `iARcoeff`, etc.), they are recycled according to the number of responses.

Also, the ARMA coefficients are intercept–only (note, `zero = c("ARcoeff", "MAcoeff")`) This may altered via `zero`, or by constraint matrices (See `constraints`) using `cm.ARMA`.

Checks on stationarity and/or invertibility can be manually via `checkTS.VGAMextra`.

Author(s)

Victor Miranda and T. W. Yee

References

Miranda, V. and Yee, T.W. (2018) Vector Generalized Linear Time Series Models. *In preparation*.

Porat, B., and Friedlander, B. (1986) Computation of the Exact Information Matrix of Gaussian Time Series with Stationary Random Components. *IEEE Transactions on Acoustics, Speech and Signal Processing*. **ASSp-34(1)**, 118–130.

See Also

`ARXff`, `MAXff`, `checkTS.VGAMextra`, `cm.ARMA`, `CommonVGAMffArguments`, `constraints`, `vglm`.

Examples

```
set.seed(3)
nn      <- 90
theta   <- c(0.12, 0.17)  # AR coefficients
phi     <- c(-0.15, 0.20)  # MA coefficients.
sdWNN <- exp(1.0)          # SDs
mu    <- c(1.25, 0.85)     # Mean (not drift) of the process.
covX  <- runif(nn + 1)     # A single covariate.
mux3  <- mu[1] + covX
##
## Simulate ARMA processes. Here, the drift for 'tsd3' depends on covX.
##
```

```
tsdata <- data.frame(TS1 = mu[1] + arima.sim(model = list(ar = theta, ma = phi,
                           order = c(2, 1, 2)), n = nn, sd = sdWNN ),
                     TS2 = mu[2] + arima.sim(model = list(ar = theta, ma = phi,
                           order = c(2, 1, 2)), n = nn, sd = exp(2 + covX)),
                     TS3 =  mux3 + arima.sim(model = list(ar = theta, ma = phi,
                           order = c(2, 1, 2)), n = nn, sd = exp(2 + covX) ),
                     x2 = covX)

### EXAMPLE 1. Fitting a simple ARIMA(2, 1, 2) using vglm().
# Note that no covariates involved.
fit.ARIMA1 <-  vglm(TS1 ~ 1, ARIMAXff(order = c(2, 1, 2), var.arg = FALSE,
                       # OPTIONAL INITIAL VALUES
                       # idrift = c(1.5)*(1 - sum(theta)),
                       # ivar = exp(4), isd = exp(2),
                       # iARcoeff = c(0.20, -0.3, 0.1),
                       # iMAcoeff = c(0.25, 0.35, 0.1),
                        type.EIM = "exact"),
                  data = tsdata, trace = TRUE, crit = "log")
Coef(fit.ARIMA1)
summary(fit.ARIMA1)
vcov(fit.ARIMA1, untransform = TRUE)
#-------------------------------------------------------------------------#
# Fitting same model using arima().
#-------------------------------------------------------------------------#
# COMPARE to EXAMPLE1
( fitArima  <- arima(tsdata$TS1, order = c(2, 1, 2)) )


### EXAMPLE 2. Here only the ARMA coefficients and drift are intercept-only.
# The random noise variance is not constant.
fit.ARIMA2 <-  vglm(TS2 ~ x2,  ARIMAXff(order = c(2, 1, 2), var.arg = TRUE,
                       lARcoeff = "rhobitlink", lMAcoeff = "identitylink",
                       type.EIM = c("exact", "approximate")[1],
                       # NOTE THE ZERO ARGUMENT.
                       zero = c("drift.mean", "ARcoeff", "MAcoeff")),
                data = tsdata, trace = TRUE)

coef(fit.ARIMA2, matrix = TRUE)
summary(fit.ARIMA2)
constraints(fit.ARIMA2)



### EXAMPLE 3. Here only ARMA coefficients are intercept-only.
# The random noise variance is not constant.
# Note that the "drift" and the "variance" are "generated" in
# terms of 'x2' above for TS3.

fit.ARIMA3 <- vglm(TS3 ~ x2,  ARIMAXff(order = c(1, 1, 2), var.arg = TRUE,
                       lARcoeff = "identitylink", lMAcoeff = "identitylink",
                       type.EIM = c("exact", "approximate")[1], nodrift = FALSE,
                       zero = c( "ARcoeff", "MAcoeff")), # NOTE THE ZERO ARGUMENT.
                data = tsdata, trace = TRUE)
```

```
coef(fit.ARIMA3, matrix = TRUE)
summary(fit.ARIMA3)
constraints(fit.ARIMA3)
```

`ARMA.studentt.ff` *VGLTSMs Family Functions: Generalized autoregressive moving average model with Student-t errors*

Description

For an ARMA model, estimates a 3–parameter Student-t distribution characterizing the errors plus the ARMA coefficients by MLE usign Fisher scoring. Central Student–t handled currently.

Usage

```
ARMA.studentt.ff(order = c(1, 0),
                 zero = c("scale", "df"),
                 cov.Reg = FALSE,
                 llocation = "identitylink",
                 lscale    = "loglink",
                 ldf       = "logloglink",
                 ilocation = NULL,
                 iscale = NULL,
                 idf = NULL)
```

Arguments

`order`	Two–entries vector, non–negative. The order u and v of the ARMA model.
`zero`	Same as `studentt3`.
`cov.Reg`	Logical. If covariates are entered, Should these be included in the ARMA model as a Regressand? Default is FALSE, then only embedded in the linear predictors.
`llocation, lscale, ldf, ilocation, iscale, idf`	Same as `studentt3`.

Details

The normality assumption for time series analysis is relaxed to handle heavy–tailed data, giving place to the ARMA model with shift-scaled Student-t errors, another subclass of VGLTSMs.

For a univariate time series, say y_t, the model described by this VGLTSM family function is

$$\theta(B)y_t = \phi(B)\varepsilon_t,$$

where ε_t are distributed as a shift-scaled Student–t with ν degrees of freedom, i.e., $\varepsilon_t \sim t(\nu_t, \mu_t, \sigma_t)$. This family functions estimates the location (μ_t), scale (σ_t) and degrees of freedom (ν_t) parameters, plus the ARMA coefficients by MLE.

Currently only centered Student–t distributions are handled. Hence, the non–centrality parameter is set to zero.

The linear/additive predictors are $\boldsymbol{\eta} = (\mu, \log \sigma, \log\log \nu)^T$, where $\log \sigma$ and ν are intercept–only by default.

Value

An object of class "vglmff" (see vglmff-class) to be used by VGLM/VGAM modelling functions, e.g., vglm or vgam.

Note

If order = 0, then AR.studentt.ff fits a usual 3–parameter Student–t, as with studentt3.

If covariates are incorporated in the analysis, these are embedded in the location–parameter model. Modify this through zero. See CommonVGAMffArguments for details on zero.

Author(s)

Victor Miranda

See Also

ARIMAXff, studentt, vglm.

Examples

```
### Estimate the parameters of the errors distribution for an
## AR(1) model. Sample size = 50

set.seed(20180218)
nn <- 250
y  <- numeric(nn)
ncp   <- 0           # Non--centrality parameter
nu    <- 3.5         # Degrees of freedom.
theta <- 0.45        # AR coefficient
res <- numeric(250)  # Vector of residuals.

y[1] <- rt(1, df = nu, ncp = ncp)
for (ii in 2:nn) {
  res[ii] <- rt(1, df = nu, ncp = ncp)
  y[ii] <- theta * y[ii - 1] + res[ii]
}
# Remove warm up values.
y <- y[-c(1:200)]
res <- res[-c(1:200)]

### Fitting an ARMA(1, 0) with Student-t errors.
AR.stut.er.fit <- vglm(y ~ 1, ARMA.studentt.ff(order = c(1, 0)),
                       data = data.frame(y = y), trace = TRUE)

summary(AR.stut.er.fit)
```

```
Coef(AR.stut.er.fit)


plot(ts(y), col = "red", lty = 1, ylim = c(-6, 6), main = "Plot of series Y with Student-t errors")
lines(ts(fitted.values(AR.stut.er.fit)), col = "blue", lty = 2)
abline( h = 0, lty = 2)
```

ARXff *VGLTSMs family functions: Order–p Autoregressive Model with covariates*

Description

Maximum likelihood estimation of the order–p autoregressive model (AR(p)) with covariates. Estimates the drift, standard deviation (or variance) of the random noise (not necessarily constant), and coefficients of the conditional–mean model.

Usage

```
ARXff(order    = 1,
      zero     = c(if (nodrift) NULL else "ARdrift", "ARcoeff"),
      xLag     = 0,
      type.EIM = c("exact", "approximate")[1],
      var.arg  = TRUE,
      nodrift  = FALSE,
      noChecks = FALSE,
      ldrift   = "identitylink",
      lsd      = "loglink",
      lvar     = "loglink",
      lARcoeff = "identitylink",
      idrift   = NULL,
      isd      = NULL,
      ivar     = NULL,
      iARcoeff = NULL)
```

Arguments

order	The order (i.e., 'p') of the AR model, which is recycled if needed. See below for further details. By default, an autoregressive model of order-1 is fitted.
zero	Integer or character–strings vector. Name(s) or position(s) of the parameters/linear predictors to be modeled as intercept-only. Details at zero.
xLag	Same as ARIMAXff.

`type.EIM`, `var.arg`, `nodrift`, `noChecks`
: Same as `ARIMAXff`.

`ldrift`, `lsd`, `lvar`, `lARcoeff`
: Link functions applied to the *drift*, the standar deviation (or variance) of the noise, and the AR coefficients. Same as `ARIMAXff`.

Further details on `CommonVGAMffArguments`.

`idrift`, `isd`, `ivar`, `iARcoeff`
: Same as `ARIMAXff`.

Details

This family function describes an autoregressive model of order-p with covariates (ARX(p)). It is a special case of the subclass VGLM–ARIMA (Miranda and Yee, 2018):

$$Y_t|\Phi_{t-1} = \mu_t + \theta_1 Y_{t-1} + \ldots + \theta_p Y_{t-p} + \varepsilon_t,$$

where $\boldsymbol{x}_t$ a (possibly time–varying) covariate vector and $\mu_t = \mu^\star + \boldsymbol{\beta}^T \boldsymbol{x}_t$ is a (time–dependent) scaled–mean, known as *drift*.

At this stage, conditional Gaussian white noise, $\varepsilon_t|\Phi_{t-1}$ is handled, in the form

$$\varepsilon_t|\Phi_{t-1} \sim N(0, \sigma^2_{\varepsilon_t|\Phi_{t-1}}).$$

The distributional assumptions on the observations are then

$$Y_t|\Phi_{t-1} \sim N(\mu_{t|\Phi_{t-1}}, \sigma^2_{\varepsilon_t|\Phi_{t-1}}),$$

involving the conditional mean equation for the ARX(p) model:

$$\mu_{t|\Phi_{t-1}} = \mu_t + \boldsymbol{\beta}^T * \boldsymbol{x}_t \theta_1 Y_{t-1} + \ldots + \theta_p Y_{t-p}.$$

Φ_t denotes the information of the joint process $\left(Y_t, \boldsymbol{x}^T_{t+1}\right)$, at time t.

The loglikelihood is computed by `dARp`, at each Fisher scoring iteration.

The linear predictor is

$$\boldsymbol{\eta} = \left(\mu_t, \log \sigma^2_{\varepsilon_t|\Phi_{t-1}}, \theta_1, \ldots, \theta_p\right)^T.$$

Note, the covariates may also intervene in the conditional variance model $\log \sigma^2_{\varepsilon_t|\Phi_{t-1}}$. Hence, this family function does not restrict the noise to be *strictly* white noise (in the sense of *constant variance*).

The unconditional mean, $E(Y_t) = \mu$, satisfies

$$\mu \to \frac{\mu^\star}{1 - (\theta_1 + \ldots + \theta_p)}$$

when the process is stationary, and no covariates are involved.

This family function currently handles *multiple responses* so that a matrix can be used as the response. Also, for further details on VGLM/VGAM–link functions refer to `Links`.

Further choices for the random noise, besides Gaussian, will be implemented over time.

Value

An object of class "vglmff" (see vglmff-class). The object is used by VGLM/VGAM modelling functions, such as vglm or vgam.

Note

zero can be either an *integer* vector or a vector of **character strings** specifying either the position(s) or name(s) (partially or not) of the parameter(s) modeled as intercept-only. Numeric values can be set as usual (See CommonVGAMffArguments). Character strings can be entered as per parameter names in this family function, given by:

c("drift", "noiseVar" or "noiseSD", "ARcoeff").

Users can modify the zero argument according to their needs.

By default, μ_t and the coefficients $\theta_1, \ldots, \theta_p$ are intercept–only. That is, $\log \sigma^2_{\varepsilon_t|\Phi_{t-1}}$ is modelled in terms of any explanatories entered in the formula.

Users, however, can modify this according to their needs via zero. For example, set the covariates in the drift model, μ_t. In addition, specific constraints for parameters are handled through the function cm.ARMA.

If var.arg = TRUE, this family function estimates $\sigma^2_{\varepsilon_t|\Phi_{t-1}}$. Else, the $\sigma_{\varepsilon_t|\Phi_{t-1}}$ estimate is returned.

For this family function the order is recycled. That is, order will be replicated up to the number of responses given in the vglm call is matched.

Warning

Values of the estimates may not correspond to stationary ARs, leading to low accuracy in the MLE estimates, e.g., values very close to 1.0. *Stationarity* is then examined, via checkTS.VGAMextra, if noChecks = FALSE (default) and **no** constraint matrices are set (See constraints for further details on this). If the estimated model very close to be non-stationary, then a warning will be outlined. Set noChecks = TRUE to completely ignore this.

NOTE: Full details on these 'checks' are shown within the summary() output.

Author(s)

Victor Miranda and T. W. Yee

References

Madsen, H. (2008) Time Series Analysis. Florida, USA: *Chapman & Hall*(Sections 5.3 and 5.5).

Porat, B., and Friedlander, B. (1986) Computation of the Exact Information Matrix of Gaussian Time Series with Stationary Random Components. *IEEE Transactions on Acoustics, Speech and Signal Processing*. **ASSp-34(1)**, 118–130.

See Also

ARIMAXff, ARMAXff, MAXff, checkTS.VGAMextra, CommonVGAMffArguments, Links, vglm,

Examples

```
set.seed(1)
nn     <- 150
tsdata <- data.frame(x2 =  runif(nn))             # A single covariate.
theta1 <- 0.45; theta2 <- 0.31; theta3 <- 0.10     # Coefficients
drift  <- c(1.3, -1.1)                            # Two responses.
sdAR   <- c(sqrt(4.5), sqrt(6.0))                 # Two responses.

# Generate AR sequences of order 2 and 3, under Gaussian noise.
# Note, the drift for 'TS2' depends on x2 !
tsdata  <-  data.frame(tsdata, TS1 = arima.sim(nn,
              model = list(ar = c(theta1, theta1^2)),  rand.gen = rnorm,
              mean = drift[1], sd = sdAR[1]),
                                TS2 = arima.sim(nn,
              model = list(ar = c(theta1, theta2, theta3)), rand.gen = rnorm,
              mean = drift[2] + tsdata$x2 , sd = sdAR[2]))

# EXAMPLE 1. A simple AR(2), maximizing the exact log-likelihood
# Note that parameter constraints are involved for TS1, but not
# considered in this fit. "rhobitlink" is used as link for AR coeffs.

fit.Ex1 <- vglm(TS1 ~ 1, ARXff(order = 2, type.EIM = "exact",
                      #iARcoeff = c(0.3, 0.3, 0.3), # OPTIONAL INITIAL VALUES
                      # idrift = 1, ivar = 1.5, isd = sqrt(1.5),
                      lARcoeff = "rhobitlink"),
              data = tsdata,  trace = TRUE, crit = "loglikelihood")
Coef(fit.Ex1)
summary(fit.Ex1)
vcov(fit.Ex1, untransform = TRUE)       # Conformable with this fit.
AIC(fit.Ex1)
#-------------------------------------------------------------------------#
# Fitting same model using arima().
#-------------------------------------------------------------------------#
(fitArima <- arima(tsdata$TS1, order = c(2, 0, 0)))
# Compare with 'fit.AR'. True are theta1 = 0.45; theta1^2 = 0.2025
Coef(fit.Ex1)[c(3, 4, 2)]    # Coefficients estimated in 'fit.AR'


# EXAMPLE 2. An AR(3) over TS2, with one covariate affecting the drift only.
# This analysis makes sense as the TS2's drift is a function ox 'x2',
# i.e., 'x2' affects the 'drift' parameter only. The noise variance
# (var.arg = TRUE) is estimated, as intercept-only. See the 'zero' argument.


#-------------------------------------------------------------------------#
# This model CANNOT be fitted using arima()
#-------------------------------------------------------------------------#
fit.Ex2 <- vglm(TS2 ~ x2,  ARXff(order = 3, zero = c("noiseVar", "ARcoeff"),
                             var.arg = TRUE),
                  ## constraints = cm.ARMA(Model = ~ 1, lags.cm = 3, Resp = 1),
              data = tsdata,  trace = TRUE, crit = "log")
```

```
# True are theta1 <- 0.45; theta2 <- 0.31; theta3 <- 0.10
coef(fit.Ex2, matrix = TRUE)
summary(fit.Ex2)
vcov(fit.Ex2)
BIC(fit.Ex2)
constraints(fit.Ex2)


# EXAMPLE 3. Fitting an ARX(3) on two responses TS1, TS2; intercept-only model with
#  constraints over the drifts. Here,
# a) No checks on invertibility performed given the use of cm.ARMA().
# b) Only the drifts are modeled in terms of 'x2'. Then,  'zero' is
# set correspondingly.
#------------------------------------------------------------------------#
# arima() does not handle this model.
#------------------------------------------------------------------------#
fit.Ex3 <- vglm(cbind(TS1, TS2) ~ x2, ARXff(order = c(3, 3),
                            zero = c("noiseVar", "ARcoeff"), var.arg = TRUE),
             constraints = cm.ARMA(Model = ~ 1 + x2, lags.cm = c(3, 3), Resp = 2),
             trace = TRUE, data = tsdata, crit = "log")

coef(fit.Ex3, matrix = TRUE)
summary(fit.Ex3)
vcov(fit.Ex3)
constraints(fit.Ex3)
```

---

benini1Qlink *Link functions for the quantiles of several 1–parameter continuous distributions*

Description

Computes the benini1Qlink transformation, its inverse and the first two derivatives.

Usage

```
benini1Qlink(theta, p = stop("Argument 'p' must be entered."),
             y0 = stop("Argument 'y0' must be specified."),
             bvalue = NULL, inverse = FALSE,
             deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. See below for further details.
p	Numeric. A single value between 0.0 and 1.0. It is the p–quantile to be modeled by this link function.

`y0` Same as benini1.

`bvalue, inverse, deriv, short, tag`
See Links.

Details

This is a link function to model any p–quantile of the 1–parameter Benini distribution. It is called the `benini1Qlink` transformation defined as

$$\log y_0 + \sqrt{\frac{-\log(1-p)}{s}}$$

where $y_0 > 0$ is a scale parameter and s is a positive shape parameter, as in benini1.

Numerical values of s or p out of range may result in `Inf`, `-Inf`, `NA` or `NaN`.

In particular, arguments `inverse` and `deriv` are disregarded if `theta` is character.

Value

For `deriv = 0`, the `benini1Qlink` transformation of `theta`, when inverse = FALSE. If `inverse = TRUE`, then the inverse transformation given by `-log(1 - p) / (theta - log y0)^2` is returned.

For `deriv = 1`, this function returns the derivative d `eta` / d `theta`, if `inverse = FALSE`. Else, the reciprocal d `theta` / d `eta` as a function of `theta`.

If deriv = 2, then the second order derivatives in terms of `theta` are accordingly returned.

Warning

The horizontal straight line $\log y0$ is a lower asymptote for this link function as θ increases to ∞. Thus, when `inverse = TRUE` and `deriv = 0` entries at `theta` becoming η must be greater than $\log y0$. Else, `NaN` will be returned. See examples 2 and 3 below.

Note

Numerical instability may occur for values `theta` too close to zero or lower than $\log y0$. Use argument `bvalue` to replace them before computing the link.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

benini1, Links.

Examples

```
## E1. benini1Qlink() and its inverse ##
 p <- 0.50; y0 = 1.25         ## Modeling the median
 my.s <- seq(0, 5, by = 0.1)[-1]
  max(my.s - benini1Qlink(benini1Qlink(my.s, p = p, y0 = y0),
                          p = p, y0 = y0, inverse  =TRUE))    ## Zero

## E2. Plot of the benini1Qlink() transformation and its inverse     ##
## Note, inverse = TRUE implies that argument 'theta' becomes 'eta'. ##
## which must be greater than log(y0). Else, value less than log(y0) ##
## are replaced by NaN.                                              ##

 #--- THE LINK
 my.b <- seq(0, 5, by = 0.01)[-1]
 plot(benini1Qlink(theta = my.b, p = p, y0 = y0) ~ my.b,
      type = "l", col = "blue", lty = "dotted", lwd = 3,
      xlim = c(-0.1, 6), ylim = c(-0.1, 5), las = 1,
      main = c("Blue is benini1Qlink(), green is the inverse"),
      ylab = "eta = benini1Qlink", xlab = "theta")
 abline(h = 0, v = 0, lwd = 2)

 #--- THE INVERSE
 lines(my.b, benini1Qlink(theta = my.b, p = p, y0 = y0, inv = TRUE),
       col = "green", lwd = 2, lty = "dashed")
 #--- Tracing the identity function for double--check
 lines(my.b, my.b)


 ## E3. WARNING! The first two values are less than log(y0)  ##
  benini1Qlink(theta = c(0.10, 0.15, 0.25, 0.35) , p = p, y0 = y0, inverse  = TRUE)
```

`borel.tannerMlink` *Link functions for the mean of 1–parameter discrete distributions: The Borel–Tanner distribution.*

Description

Computes the `borel.tannerMlink` transformation, its inverse and the first two derivatives.

Usage

```
borel.tannerMlink(theta, Qsize = 1,
                  bvalue = NULL, inverse = FALSE,
                  deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`Qsize`	A positive integer. It is called Q. Same as `borel.tanner`. Default it 1.
`bvalue, inverse, deriv, short, tag`	Details at `Links`

Details

As with `zetaffMlink` or `yulesimonMlink`, this link function is part of a set of link functions in **VGAM** developed under a common methodology: by taking the logarithm of the mean of the corresponding distribution.

In particular, this link function emerges by computing the logarithm of the mean of the Borel–Tanner distribution. It is defined as

$$\texttt{borel.tannerMlink}(\texttt{a}) = -\log(\texttt{Q}^{-1} - \texttt{a}\texttt{Q}^{-1}),$$

where a, $0 < a < 1$, is a scale parameter as in `borel.tanner`.

The domain set of `borel.tannerMlink` is the open interval $(0, 1)$, except when `inverse = TRUE` and `deriv = 0`. See below for further details about this. Moreover, unlike `zetaffMlink` or `posPoiMlink`, the inverse of `borel.tannerMlink` can be written in closed–form.

Values of a (i.e. `theta`) out of range will result in `NaN` of `NA`.

If `theta` is a character, arguments `inverse` and `deriv` are discarded.

Value

For `deriv = 0`, the `borel.tannerMlink` transformation of `theta`, if `inverse = FALSE`. When `inverse = TRUE`, `theta` becomes η and the inverse of `borel.tannerMlink`, given by

$$1 - \frac{Q}{e^{\eta}},$$

is returned. Here, the domain set changes to $(0, \infty)$.

For `deriv = 1`, d `eta` / d `theta` as a function of `theta` if `inverse = FALSE`, else the reciprocal d `theta` / d `eta`.

Similarly, when `deriv = 2` the second order derivatives in terms of `theta` are returned.

References

Haight, F. and Brueuer, M. A. (1960) The Borel–Tanner distribution. *Biometrika*, **47**, 143–150.

Note

The vertical line $a = 1$ is an asymptote for this link function, which sharply grows for values of a too close to 1.0 from the left. For such cases, `Inf` might result when computing `borel.tannerMlink`.

This link function is useful to model any parameter in $(0, 1)$. Then, some problems may occur if there are covariates causing out of range values.

Author(s)

V. Miranda and T. W. Yee

See Also

borel.tanner, yulesimonMlink, zetaffMlink, posPoiMlink, Links.

Examples

```
## Example 1. Special values for theta (or eta, accordingly)  ##
a.par <- c(0, 1:10/10, 20, 1e1, Inf, -Inf, NaN, NA)

# The borel.tannerMlink transformation and the first two derivatives.
print(rbind(a.par,
 deriv1 = borel.tannerMlink(theta = a.par, inverse = FALSE, deriv = 1),
 deriv2 = borel.tannerMlink(theta = a.par, inverse = FALSE, deriv = 2)),
 digits = 2)

# The inverse of 'borel.tannerMlink()' and the first two derivatives.
# 'theta' turns into 'eta'.
print(rbind(a.par,
    Invderiv1 = borel.tannerMlink(theta = a.par, inverse = TRUE, deriv = 1),
    Invderiv2 = borel.tannerMlink(theta = a.par, inverse = TRUE, deriv = 2)),
    digits = 2)


## Example 2 ##
a.param <- c(0, 1, 5, 10, 1e2, 1e3)
rbind(a.values = a.param,
      inv.BT = borel.tannerMlink(theta = a.param, inverse = TRUE))

data.inv <- borel.tannerMlink(borel.tannerMlink(a.param, inv = TRUE)) - a.param
summary(data.inv)                     ## Should be zero



## Example 3. Some link functions in VGAM with domain set (0, 1) ##
a.param <- ppoints(100)


par(lwd = 2)
plot(a.param, borel.tannerMlink(a.param), ylim = c(-5, 7), xlim = c(-0.01, 1.01),
     type = "l", col = "gray10", ylab = "transformation",
     las = 1, main = "Some probability link functions")
lines(a.param, logffMlink(a.param), col = "blue")
lines(a.param, logitlink(a.param), col = "limegreen")
lines(a.param, probitlink(a.param), col = "purple")
lines(a.param, clogloglink(a.param), col = "chocolate")
lines(a.param, cauchitlink(a.param), col = "tan")
abline(v = c(0.5, 1), lty = "dashed")
abline(v = 0, h = 0, lty = "dashed")
```

```
legend(0.05, 7, c("borel.tanneMlink", "logffMlink", "logitlink", "probitlink",
                  "clogloglink", "cauchitlink"),
       col = c("gray10", "blue", "limegreen", "purple", "chocolate", "tan"),
      lwd = 1)
par(lwd = 1)
```

`break.VGAMextra` *Names/Value of linear predictors/parameters in time series family functions.*

Description

Splitting out the names of linear predictors or Numeric values for parameters in time series family functions in **VGAMextra**.

Usage

```
break.VGAMextra(eta      = NULL,
                M1       = NULL,
                noInter  = NULL,
                bOrder   = NULL,
                NOS      = NULL,
                lInter   = "identitylink",
                lvar     = "loglink",
                lsd      = "loglink",
                lcoeff1  = "rhobitlink",
                lcoeff2  = "rhobitlink",
                typeTS   = "AR",
                namesLP  = FALSE,
                Complete = FALSE,
                varArg   = NULL)
```

Arguments

`eta`	A matrix of dimensions `c(n, M)` storing the linear predictors values coming from the `vglm` fit. Here, M is the number of parameters. See `warning` below for further information.
`M1`	Number of parameters involved in the `vglm` fit.
`noInter`	Logical. To determine whether the intercept is estimated. If `'TRUE'`, the intercept is not estimated and set to 0.
`bOrder`	A vector. The order of the linear process fitted. Either a single number (if one response), or a vector (if multiple responses).
`NOS`	Integer. Number of respones set in the `vglm` call.
`lInter, lvar, lsd, lcoeff1, lcoeff2`	Link functions applied to parameters. Same as in `ARXff`, or `MAXff`.

typeTS	Character. Currently, options "AR" for Autoregressive, and "MA" for Moving Average processes are handled.
namesLP	Logical. This function returns either the names of linear the predictors/parameters (if namesLP = TRUE) or parameter values (default) broken down from the eta matrix.
Complete	Logical. If TRUE, columns of zeros are incorporated into the matrix eta. See below for further details.
varArg	Sames as in ARXff or MAXff

Details

Time series family functions in **VGAMextra** currently recycle the order set in the vglm. Particularly, it occurs when the number of responses is fewer than the specified order. For instance, if the order set in vglm is $c(1, 3)$, and 5 responses are managed, then the new order becomes $c(1, 3, 1, 3, 1)$.

Due to such flexibility, time series family functions require specific functions to unload the amount of code within each one.

Moreover, when the order is recycled, the matrix eta is *completed*, as if the order was the same for each response. This feature is enabled when Complete = TRUE. This 'common' order turns out to be the maximum order established in the vector order. This trick makes the family function to work properly. To return to the riginal 'order', eta is reduced in the same number of colums initially added.

break.VGAMextra works in this context. It may return either the names of the linear predictors/parameters, or the parameter values splitted out as a list. Thus, link functions entered in the vglm call must be passed down to this functions. For further details on link functions refer to CommonVGAMffArguments.

Value

A list containing either the names of the linear predictors or the parameters values (not linear predictors) unwrapped from tje eta matrix, as follows:

a) If namesLP = FALSE (default), value of parameters are returned in this order: the intercept (1), standard deviation and variance of the white noise (2, 3), and the coefficients (4).

b) If namesLP = TRUE, **names of linear predictors** are returned in the first entry, whereas **parameter names** are allocated to the second entry.

Yee and Wild (1996) provide more detailed information about the relationship between linear predictors and parameters within the VGLM statistical framework.

Warning

Note that library **VGAM** is definitely required.

Warning

Be aware of the dimensions of matrix eta. It is c(n, M), where n is the sample size, and M is the number of parameters. If multiple responses, then M equals the summation of parameters individually.

Author(s)

Victor Miranda and T. W. Yee

References

Yee, T. W. and Wild, C. J. (1996) Vector Generalized Additive Models. *Journal of the Royal Statistical Society*, Series B, Methodological, **58(3)**, 481–493.

See Also

ARXff, MAXff,

CommonVGAMffArguments, vglm.

Examples

```
library(VGAM)

eta     <- matrix(runif(100), nrow = 10, ncol = 10)
M1      <- c(5, 5)
noInter <- FALSE
bOrder  <- c(3, 3)
NOS     <- 2

### ONLY LINEAR PREDICTORS/PARAMETERS NAMES!
### RETURNED OBJECT IS A LIST !

break.VGAMextra(M1      = M1,
                noInter = noInter,
                bOrder  = bOrder,
                NOS     = NOS,
                typeTS  = "AR",
                namesLP = TRUE,
                varArg  = TRUE)

### PARAMETER VALUEs... "UNWRAPPED". Inverse link functions are applied.
###  Note that namesLP must be set to FALSE

break.VGAMextra(eta     = eta,
                M1      = M1,
                noInter = noInter,
                bOrder  = bOrder,
                NOS     = NOS,
                typeTS  = "AR",
                namesLP = FALSE,
                varArg  = TRUE)
```

checkTS.VGAMextra *Polynomial roots based on* transfer operators *in Vector Generalized Time Series Family Functions*

Description

checkTS.VGAMextra computes the polynomial roots as per *transfer operator* in Vector Generalized Time Series Family Functions in **VGAMextra**

Usage

```
checkTS.VGAMextra(thetaEst = NULL,
                  tsclass  = c("AR", "MA"),
                  chOrder  = 1,
                  NofS     = 1,
                  retmod   = TRUE,
                  pRoots   = TRUE)
```

Arguments

thetaEst	A vector of coefficients. Its lenght must be NofS * chOrder. If vglm is called, then thetaEst contains the estimated coefficients of the model specified in formula .
tsclass	Character indicating the model class to be checked. Presently, options "AR" and "MA" are handled.
chOrder	Positive integer. The order of polynomial associated to the underlying procees involved: either p or q, which apply for "AR" or "MA" rspectively.
NofS	A positive integer denoting the number of Time Series to verify. In the vglm environment, NofS is the number of responses given in formula.
retmod	Logical. If TRUE (default), the *Module* of all roots as per transfer operator in the process established in tsclass is returned. Else, essentially the roots are returned.
pRoots	Logical. If TRUE (default), the roots computed from estimated models are displayed along with the time series family function execution.

Details

Stationarity and/or Invertibility of time series (TS) are usually verified via the roots of the polynomial derived from the *transfer operators*.

In particular, checkTS.VGAMextra computes such roots via the coefficients estimated by vector generalized TS family functions available in **VGAMextra** (ARXff, and MAXff).

Specifically, checkTS.VGAMextra verifies whether the TS analyzed via vglm is *stationary* or *invertible*, accordingly.

Note that an autoregressive process of order-p [AR(p)] with coefficients $\theta_1, \ldots, \theta_p$ can be written in the form

$$\theta(B)Y_t = \varepsilon_t,$$

where

$$\theta(B) = 1 - \sum_{k=1}^{p} \theta_k B^k$$

Here, $\theta(B)$ is referred to as the *transfer operator* of the process, and $B^k Y_t = Y_{t-k},$, for $k = 0, 1, \ldots, p$, is the lagged single-function.

In general, an autoregressive process of order-p is *stationary* if the roots of

$$\theta(z) = 1 - \theta_1 z - \ldots - \theta_q z^q$$

lie *outside* the unit circle, i.e. $|z| > 1$.

Similarly, a moving-average process of order-q can be formulated (without loss of generality $\mu = 0$)

$$Y_t = \psi(B)\varepsilon_t,$$

where $\psi(B)$ is the *transfer operator*, given by

$$\psi(B) = 1 + \sum_{k=1}^{q} \psi_k B^k,$$

Note that $\psi_0 = 1$, and $B^k \varepsilon_t = \varepsilon t - k$.

Hence, a moving-average process of order-q [MA(q)], generally given by (note $\mu = 0$)

$$Y_t = \phi_1 \varepsilon_{t-1} + \ldots + \phi_q \varepsilon_{t-q} + \varepsilon_t,$$

is *invertible* if all the roots of

$$\phi(B) = 1 + \phi_1 B + \ldots + \phi_q B^q$$

lie *outside* the unit circle., i.e.m $|z| > 1$.

Parallel arguments can be stated for autoregressive moving aberage processes (ARMA). See Box & Jenkins (1970) for further details.

Value

A vector whose elements are the roots of polynomials *inherited* from *transfer operators* according to the process analyzed.

Alternatively, the modules of roots can by returned instead of merely roots via the `retmod` argument.

Warning

The argument `thetaEst` manages the coefficients of the TS model(s) in turn. Then, it must be `NofS * chOrder` length, where `NofS` is the number of responses established in the `vglm` call. Here the coefficients for each response must be sequentially groped.

A moving average process is always stationary (See Madsen (2007) for further details). Consequently, the `MAXff` in **VGAMextra** verifies (by default) only for invertibility. To enable this option set `nowarning = FALSE` in the `MAXff` call.

Similarly, `ARXff` verifies whether the TS data fitted is stationary, whereas `ARMAXff()` verifies both properties.

Note

For TS family functions in the VGLM/VGAM framework, `checkTS.VGAMextra` is called at the final iteration of Fisher scoring. It means that the MLE estimates are actually evaluated to verify whether the process is *stationary* or *invertible*.

If any root has module less than $1 + 1e - 5$, a `warning` is displayed for informative purposes.

Argument `thetaEst` manages the parameters of the TS model in turn.

Author(s)

Victor Miranda and T. W. Yee.

References

Box, G.E.P. and Jenkins, G.M. (1970) *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, USA.

Madsen, H (2007) *Time Series Analysis*. Chapman \& Hall/CRC, Boca Raton, Florida, USA.

Examples

```
# A moving average process order-3 with coeffs --> c(2.4, -5.6, 0.83)
#-------------------------#
# This is NOT invertible !
#-------------------------#

MAcoeffs <- c(2.4, -5.6, 0.83)
checkTS.VGAMextra(thetaEst = MAcoeffs,
                  tsclass = "MA",
                  chOrder = 3,
                  retmod = FALSE)


# AR process order-3 with coeffs --> c( 0.45, 0.45^2, 0.45^3 )
#-------------------------#
# This is stationary !
#-------------------------#
```

```
ARcoeffs <- c( 0.45 , 0.45^2 , 0.45^3 )
checkTS.VGAMextra(thetaEst = ARcoeffs,
                  tsclass = "AR",
                  chOrder = 3,
                  retmod = TRUE,
                  pRoots = TRUE)  # DEFAULT for 'pRoots'
```

`cm.ARMA` *Constraint matrices for vector generalized time series family functions.*

Description

Constraint matrices for coefficients of vector genelized time series family functions in **VGAMextra**.

Usage

```
cm.ARMA(Model      = ~ 1,
        Resp       =  1,
        lags.cm    =  2,
        offset     = -2,
        whichCoeff =  1,
        factorSeq  =  2)
```

Arguments

`Model`	A symbolic description of the model being fitted. Must match that formula specified in the `vglm()` call.
`lags.cm`	Vector of POSITIVE integers greater than 1 indicating the *order* for each response. It must match the *orders* entered in the `vglm` call. Its default value is `2`, assuming that a TS process of order greater than 1 is being fitted. If `lags.cm` < `2`, then NO constraints are required as only one coefficient (AR, MA or ARMA) is being estimated.
`offset`	Vector of integers specifying the position of the ARMA coefficient at which constraints initiate FOR each response. If negative, it is recycled and the absolute value is used. The default value is `-2`, which refers to the fourth position on the vector parameter, right after the drift or mean, the white noise sd, and the first ARMA coefficient. Particularly, if only one coefficient is being estimated, i.e, an AR or MA process of order-1 is being fitted, then NO restrictions over the (unique) coefficient are needed. Consequently, `abs(offset < 2)` leads to a message error.
`whichCoeff`	Vector of POSITIVE integers strictly less than '`abs(offset)`', each entry aplies to each response in the `vglm(...)` call. This argument allows the user to specify the *unrestricted* coefficient to be considered for constraints. For instance,

whichCoeff = 2 means that θ_2 is the required coefficient to compute the constraint matrices. By default, whichCoeff = -1 which implies that θ_1 is used for this purpose.

If whichCoeff is greater than or equal to abs(offset), an error message is displayed since constraints must be function of unrestricted parameters.

Resp — The number of responses in the model fitted. Must match the number of responses given in formula in the vglm call.

factorSeq — Vector of POSITIVE integers. Thus far, restrictions handled are *geometric sequences* and *arithmetic progressions*. Hence, factorSeq specifies either the initial *power* or *factor* at restrictions.

See below for further details.

Details

NOTE: Except for the Model, all arguments of length 1 are recycled when Resp ≥ 2.

Time Series family functions in **VGAMextra** that are derived from AR(p) or MA(q) processes include the *drift* term (or mean) and the *white noise* standard deviation as the first two elements in the vector parameter. For an MA(4), for example, it is given by

$$(\mu, \sigma_\varepsilon, \phi_1, \phi_2, \phi_3, \phi_4).$$

Thus, constraint matrices on coefficients can be stated from the *second* coefficient, i.e., from ϕ_2. This feature is specified with offset = -2 by default.

In other words, offset indicates the exact position at which parameter restrictions commence. For example, offset = -3 indicates that ϕ_3 is the first coefficient over which constraints are applied. Then, in order to successfully utilize this argument, it must be greater than or equal to 2 in absolute value. Otherwise, an error message will be displayed as no single restriction are amenable with ϕ_1 only.

Furthermore, if lags.cm = 1, i.e, a AR or MA process of order one is being fitted, then NO constraints are required either, as only one coefficient is directly considered.

Hence, the miminum absolute value for argument offset is 2

As for the factorSeq argument, its defaul value is 2. Let factorSeq = 4, lags.cm = 5, offset = -3, and whichCoeff = 1. The coefficient restrictions if a *geometric progression* is assumed are

$$\theta_3 = \theta_1^4,$$

$$\theta_4 = \theta_1^5,$$

$$\theta_5 = \theta_1^6,$$

If coefficient restrictions are in *arithmetic sequence*, constraints are given by

$$\theta_3 = 4 * \theta_1,$$

$$\theta_4 = 5 * \theta_1,$$

$$\theta_5 = 6 * \theta_1,$$

The difference lies on thelink function used: loglink for the first case, and identitylink for the latter.

Note that conditions above are equivalent to test the following two Null Hypotheses:

$$Ho : \theta_k = \theta_1^k$$

or

$$Ho : \theta_k = j * \theta_1$$

for $k = 3, 4, 5$.

Simpler hypotheses can be tested by properly setting all arguments in cm.ARMA(). For instance, the default list of constraint matrices returned by cm.ARMA() allows to test

$$Ho : \theta_k = \theta_1^j$$

for $k = 2$, in a TS model of order-2 with one response.

Value

A list of constraint matrices with specific restrictions over the AR(p), MA(q) or ARMA (p, q) coefficients. Each matrix returned is conformable with the VGAM/VGLM framework.

Paragrpah above means that each constraint matrix returned by cm.ARMA() is full-rank with *M* rows (number of parameters), as required by **VGAM**. Note that constraint matrices within the VGAM/VGLM framework are *M* by *M* identity matrices by default.

Restrictions currently handled by cm.ARMA() are (increasing) arithmetic and geometric progressions.

Warning

Hypotheses above can be tested by properly applying *parameter link functions*. If the test

$$Ho : \theta_k = \theta_1^k,$$

arises, then constraint matrices returned by cm.ARMA() are conformable to the use of loglink.

On the other hand, the following hypothesis

$$Ho : \theta_k = k * \theta_1,$$

properly adapts to the link function identitylink. $k = 2, 3, \ ldots$.

For further details on parameter link functions within **VGAM**, see CommonVGAMffArguments.

Note

cm.ARMA() can be utilized to compute constraint matrices for many VGLTSM fmaily functions, e.g., ARXff and MAXff in **VGAMextra**.

More improvements such as restrictions on the *drift parameter* and *white noise standard deviation* will be set later.

Author(s)

Victor Miranda and T. W. Yee

References

Yee, T. W. and Hastie, T. J. (2003) Reduced-rank vector generalized linear models. *Statistical Modelling*, **3**, 15–41.

Yee, T. W. (2008) The VGAM Package. *R News*, **8**, 28–39.

See Also

loglink, rhobitlink, CommonVGAMffArguments.

Examples

```
#############
# Example 1.
#############
# Constraint matrices for a TS family function (AR or MA)
# with 6 lagged terms.
# Restriction commences at the third position (theta[3]) powered to
# or multiplied by 4. Intercept-only model.
position   <- -3
numberLags <-  6
myfactor   <-  4
cm.ARMA(offset = position, lags.cm = numberLags, factorSeq = myfactor)

# With one covariate
cm.ARMA(Model =  ~ x2, offset = position,
        lags.cm = numberLags, factorSeq = myfactor)


# Or 2 responses...
cm.ARMA(offset = position, lags.cm = numberLags,
        factorSeq = myfactor, Resp = 2)


# The following call causes an ERROR.
# cm.ARMA(offset = -1, lags.cm = 6, factorSeq = 2)


##############
# Example 2.
##############

# In this example, the use of constraints via 'cm.ARMA()' is
# included in the 'vglm' call. Here, two AR(2) models are fitted
# in the same call (i.e. two responses), where different constraints
# are set, as follows:
# a) list(ar = c(theta1, theta1^2)) and
# b) list(ar = c(theta2, theta2^2 )).

# 2.0 Generate the data.
set.seed(1001)
```

```
nn     <- 100
# A single covariate.
covdata <- data.frame(x2 =  runif(nn))

theta1 <- 0.40; theta2 <- 0.55
drift  <- c(0.5, 0.75)
sdAR   <- c(sqrt(2.5), sqrt(2.0))

# Generate AR sequences, TS1 and TS2, considering Gaussian white noise

# Save both in a data.frame object: the data.
tsdata  <-
  data.frame(covdata, # Not used
             TS1 = arima.sim(nn,
                             model = list(ar = c(theta1, theta1^2)),
                             rand.gen = rnorm,
                             mean = drift[1], sd = sdAR[1]),
             TS2 = arima.sim(nn,
                             model = list(ar = c(theta2, theta2^2)),
                             rand.gen = rnorm,
                             mean = drift[2], sd = sdAR[2]))

# 2.1 Fitting both time series with 'ARXff'... multiple responses case.
fit1 <- vglm(cbind(TS1, TS2) ~ 1,
             ARXff(order = c(2, 2), type.EIM = "exact"),
             data = tsdata,
             trace = TRUE)

Coef(fit1)
coef(fit1, matrix = TRUE)
summary(fit1)

## Same length for both vectors, i.e. no constraints.
length(Coef(fit1))
length(coef(fit1, matrix = TRUE))




###2.2 Now, fit the same models with suitable constraints via 'cm.ARMA()'
# Most importantly, "loglink" is used as link function to adequately match
# the relationship between coefficients and constraints. That is:
# theta2 = theta1^2, then log(theta2) = 2 * log(theta1).

fit2 <- vglm(cbind(TS1, TS2) ~ 1,
             ARXff(order = c(2, 2), type.EIM = "exact", lARcoeff = "loglink"),
             constraints = cm.ARMA(Model = ~ 1,
                                   Resp = 2,
                                   lags.cm = c(2, 2),
                                   offset  = -2),
             data = tsdata,
             trace = TRUE)
Coef(fit2)
```

```
coef(fit2, matrix = TRUE)
summary(fit2)

# NOTE, for model 1, Coeff2 = Coeff1^2, then log(Coeff2) = 2 * log(Coeff1)
( mycoef <- coef(fit2, matrix = TRUE)[c(3, 4)] )
2 * mycoef[1] - mycoef[2]    # SHOULD BE ZERO

# Ditto for model 2:
( mycoef <- coef(fit2, matrix = TRUE)[c(7, 8)] )
2 * mycoef[1] - mycoef[2]    # SHOULD BE ZERO

## Different lengths, due to constraints
length(Coef(fit2))
length(coef(fit2, matrix = TRUE))
```

`dmultinorm` *Density for the multivariate Normal distribution*

Description

Density for the multivariate Normal distribution

Usage

```
dmultinorm(vec.x, vec.mean = c(0, 0),
           mat.cov = c(1, 1, 0),
           log = FALSE)
```

Arguments

`vec.x`	For the R–multivariate Normal, an R–vector of quantiles.
`vec.mean`	The vector of means.
`mat.cov`	The vector of variances and covariances, arranged in that order. See below for further details.
`log`	Logical. If `TRUE`, the logged values are returned.

Details

This implementation of the multivariate (say R–dimensional) Normal density handles the variances and covariances, instead of correlation parameters.

For more than one observation, arrange all entries in matrices accordingly.

For each observation, `mat.cov` is a vector of length $R \times (R+1)/2$, where the first R entries are the variances $\sigma^2 i$, $i = 1, \ldots, R$, and then the covariances arranged as per rows, that is, cov_{ij} $i = 1, \ldots, R, j = i+1, \ldots, R$.

By default, it returns the density of two independent standard Normal distributions.

Value

The density of the multivariate Normal distribution.

Warning

For observations whose covariance matrix is not positive definite, `NaN` will be returned.

Author(s)

Victor Miranda

See Also

`binormal`.

Examples

```
###
### Two - dimensional Normal density.
###
set.seed(180228)
nn  <- 25
mean1 <- 1; mean2 <- 1.5; mean3 = 2
var1 <- exp(1.5); var2 <- exp(-1.5); var3 <- exp(1); cov12 = 0.75
dmvndata <- rbinorm(nn, mean1 = 1, mean2 = 1.5, var1 = var1, var2 = var2,
                    cov12 = cov12)

## Using dbinorm() from VGAM.
d2norm.data <- dbinorm(x1 = dmvndata[, 1], x2 = dmvndata[, 2],
                       mean1 = mean1, mean2 = mean2, var1 = var1, var2 = var2,
                       cov12 = cov12)
## Using dmultinorm().
d2norm.data2 <- dmultinorm(vec.x = dmvndata, vec.mean = c(mean1, mean2),
                       mat.cov = c(var1, var2, cov12))
summary(d2norm.data)
summary(d2norm.data2)
##
## 3--dimensional Normal.
##
dmvndata <- cbind(dmvndata, rnorm(nn, mean3, sqrt(var3)))

d2norm.data3 <- dmultinorm(dmvndata, vec.mean = c(mean1, mean2, mean3),
                      mat.cov = c(var1, var2, var3, cov12, 0, 0))

hist(d2norm.data3)
summary(d2norm.data3)
```

`ECM.EngleGran` *VGLTSM family function for the Two–dimensional Error–Correction Model (Engle and Granger, 1987) for $I(1)$–variables*

Description

Estimates a bidimensional error-correction model of order–(K, L), as proposed by Engle–Granger (Two step–approach; 1987), with bivariate normal errors by maximum likelihood estimation using Fisher scoring.

Usage

```
ECM.EngleGran(ecm.order = c(1, 1),
              zero = c("var", "cov"),
              resids.pattern = c("intercept", "trend",
                                 "neither", "both")[1],
              lag.res = 1,
              lmean = "identitylink",
              lvar  = "loglink",
              lcov  = "identitylink",
              ordtsDyn = 0)
```

Arguments

`ecm.order`	Length–2 (positive) integer vector. The order of the ECM model.
`zero`	Integer or character–string vector. Details at `zero`.
`resids.pattern`	Character. How the static linear regression $y_{2,t} \sim y_{1,t}$ must be settle to estimate the residuals $\widehat{z}_t$. The default is a linear model with intercept, and no trend term. See below for details.
`lag.res`	Numeric, single positive integer. The error term for the long–run equilibrium path is lagged up to order `lag.res`. See below for further details.
`lmean, lvar, lcov`	Same as `MVNcov`.
`ordtsDyn`	Positive integer. Allows to compare the estimated coefficients with those provided by the package 'tsDyn'. See below for further details.

Details

This is an implementation of the two–step approach as proposed by Engle–Granger [1987] to estimate an order–(K, L) bidimensional error correction model (ECM) with bivariate normal errors.

This ECM class models the dynamic behaviour of two cointegrated $I(1)$-variables, say $y_{1,t}$ and $y_{2,t}$ with, probably, $y_{2,t}$ a function of $y_{1,t}$. Note, the response must be a two–column matrix, where the

first entry is the regressor, i.e, $y_{1,t}$ above, and the regressand in the second colum. See Example 2 below.

The general specification of the ECM class described by this family function is

$$\Delta y_{1,t}|\Phi_{t-1} = \phi_{0,1} + \gamma_1 \widehat{z}_{t-k} + \sum_{i=1}^{K} \phi_{1,i} \Delta y_{2,t-i} + \sum_{j=1}^{L} \phi_{2,j} \Delta y_{1,t-j} + \varepsilon_{1,t},$$

$$\Delta y_{2,t}|\Phi_{t-1} = \psi_{0,1} + \gamma_2 \widehat{z}_{t-k} + \sum_{i=1}^{K} \psi_{1,i} \Delta y_{1,t-i} + \sum_{j=1}^{L} \psi_{2,j} \Delta y_{2,t-j} + \varepsilon_{2,t}.$$

Under the binormality assumption on the errors $(\varepsilon_{1,t}, \varepsilon_{2,t})^T$ with covariance matrix V, model above can be seen as a VGLM fitting linear models over the conditional means, $\mu_{\Delta y_{1,t}} = E(\Delta y_{1,t}|\Phi_{t-1})$ and $\mu_{\Delta y_{2,t}} = E(\Delta y_{2,t}|\Phi_{t-1})$, producing

$$(\Delta y_{1,t}|\Phi_{t-1}, \Delta y_{2,t}|\Phi_{t-1})^T \sim N_2(\mu_{\Delta y_{1,t}}, \mu_{\Delta y_{2,t}}, \mathrm{V})$$

The covariance matrix is assumed to have elements σ_1^2, σ_2^2, and Cov_{12}.

Hence, the parameter vector is

$$\boldsymbol{\theta} = (\phi_{0,1}, \gamma_1, \phi_{1,i}, \phi_{2,j}, \psi_{0,1}, \gamma_2, \psi_{1,i}, \psi_{2,j}, \sigma_1^2, \sigma_2^2, \mathrm{Cov}_{12})^T,$$

for $i = 1, \ldots, K$ and $j = 1, \ldots, L$.

The linear predictor is

$$\boldsymbol{\eta} = (\mu_{\Delta y_{1,t}}, \mu_{\Delta y_{2,t}}, \texttt{loglink}\ \sigma_1^2, \texttt{loglink}\ \sigma_2^2, \mathrm{Cov}_{12})^T.$$

The estimated cointegrated vector, $\widehat{\boldsymbol{\beta}^\star} = (1, -\widehat{\boldsymbol{\beta}})^T$ is obtained by linear regression depending upon `resids.pattern`, as follows:

1) $y_{2,t} = \beta_0 + \beta_1 y_{1,t} + z_t$, if `resids.pattern = "intercept"`,

2) $y_{2,t} = \beta_1 y_{1,t} + \beta_2 t + z_t$, if `resids.pattern = "trend"`,

3) $y_{2,t} = \beta_1 y_{1,t} + z_t$, if `resids.pattern = "neither"`, or else,

4) $y_{2,t} = \beta_0 + \beta_1 y_{1,t} + \beta_2 t + z_t$, if `resids.pattern = "both"`,

where $\widehat{\boldsymbol{\beta}} = (\widehat{\beta_0}, \widehat{\beta_1}, \widehat{\beta_2})^T$, and z_t assigns the error term.

Note, the *estimated residuals*, $\widehat{z}_t$ are (internally) computed from any of the linear models 1) – 4) selected, and then lagged up to order `alg.res`, and embedded as explanatories in models $\Delta y_{1,t}|\Phi_{t-1}$ and $\Delta y_{3,t}|\Phi_{t-1}$ above. By default, $\widehat{z}_{t-1}$ are considered (as `lag.res = 1`), although it may be any lag $\widehat{z}_{t-k}$, for $k > 0$. Change this through argument `lag.res`.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Note

Reduced–Rank VGLMs (RR-VGLMs) can be utilized to aid the increasing number of parameters as K and L grows. See `rrvglm`.

By default, σ_1^2, σ_2^2 and Cov_{12} are intercept–only. Set argument `zero` accordingly to change this.

Package **tsDyn** also has routines to fit ECMs. However, the bivariate–ECM handled (similar to that one above) differs in their parametrization: **tsDyn** considers the *current* estimated residual, $\widehat{z}_t$ instead of $\widehat{z}_{t-1}$ in models $\Delta y_{1,t}|\Phi_{t-1}$ and $\Delta y_{2,t}|\Phi_{t-1}$.

See Example 3 below which compares ECMs fitted with **VGAMextra** and **tsDyn**.

Author(s)

Victor Miranda

References

Engle, R.F. and Granger C.W.J. (1987) Co-integration and error correction: Representation, estimation and testing. *Econometrica*, **55**(2), 251–276.

Pfaff, B. (2011) *Analysis of Integrated and Cointegrated Time Series with* R. Seattle, Washington, USA: *Springer*.

See Also

`MVNcov`, `rrvglm`, `CommonVGAMffArguments`, `Links`, `vglm`.

Examples

```
## Example 1. Comparing the Engle -- Granger procedure carried oud by two procedures.
##            ECM.EngleGran() makes easier the fitting process.
## Here, we will use:
## A) The R code 4.2, in Chapter 4, Pfaff (2011).
##    This code 1) generates artificial data and 2) fits an ECM, following
##    the Engle --Granger procedure.
## B) The ECM.EngleGran() family function to fit the same model assuming
##    bivariate normal innovations.
## The downside in the R code 4.2 is the assumption of no--correlation among
## the errors. These are generated indenpendently.
## A)
## STEP 1. Set up the data (R code as in Pfaff (2011)).
nn <- 100
set.seed(123456)
e1 <- rnorm(nn)   # Independent of e2
e2 <- rnorm(nn)
y1 <-  cumsum(e1)
y2 <- 0.6 * y1 + e2
lr.reg <- lm(y2 ~ y1)
error <- residuals(lr.reg)
error.lagged <- error[-c(nn - 1, nn)]
dy1 <- diff(y1)
```

```
dy2 <- diff(y2)
diff.dat <- data.frame(embed(cbind(dy1, dy2), 2))
colnames(diff.dat) <- c('dy1', 'dy2', 'dy1.1', 'dy2.1')

##  STEP 2. Fit the ECM model, using lm(), R code as in Pfaff (2011).
ecm.reg <- lm(dy2 ~ error.lagged + dy1.1 + dy2.1, data = diff.dat)

summary(ecm.reg)


## B) Now, using ECM.EngleGran() and VGLMs, the steps at A) can be skipped.
## Enter the I(1)--variables in the response vector only, putting down the
## the dependent variable from the I(1) set, i.e. y2, in the second column.

coint.data <- data.frame(y1 = y1, y2 = y2)
fit.ECM <- vglm(cbind(y1, y2) ~ 1, ECM.EngleGran, data = coint.data, trace = TRUE)

## Check coefficients ##
coef(fit.ECM, matrix = TRUE)  ## Compare 'Diff2' with summary(ecm.reg)
coef(summary(ecm.reg))

head(depvar(fit.ECM))   # The estimated differences (first order)
vcov(fit.ECM)
constraints(fit.ECM, matrix = TRUE)

## Not run:
### Example 2.  Here, we compare ECM.EngleGran() from VGAMextra with VECM() from
##              package "tsDyn" when fitting an ECM(1, 1). We will make use of
##              the argument 'ordtsDyn' so that the outcomes can be compared.

library("tsDyn")  # Need to be installed first.
fit.tsDyn1 <- with(coint.data, VECM(cbind(y2, y1), lag = 1, estim = "2OLS")) #  MODEL 1
summary(fit.tsDyn1)

### Fit same model using ECM.EngleGran(). NOTE: Set ordtsDyn = 1 !!          #  MODEL 2
fit.ECM.2 <- vglm(cbind(y1, y2) ~ 1, ECM.EngleGran(ecm.order = c(1, 1),
                 resids.pattern = "neither", ordtsDyn = 1),
                 data = coint.data, trace = TRUE)

coef.ECM.2 <- coef(fit.ECM.2, matrix = TRUE)
fit.tsDyn1$coefficients                      ## From pakage 'tsDyn'.
t(coef.ECM.2[, 1:2][c(2, 1, 4, 3), ][, 2:1]) ## FROM VGAMextra


### Example 3. An ECM(2, 2), with residuals estimated by OLS, with NO intercept
###            and NO trend term. The data set is 'zeroyld', from package tsDyn.
###            ECM.EngleGran() and with VECM() willbe compared again.
data(zeroyld, package = "tsDyn")

# Fit a VECM with Engle-Granger 2OLS estimator:
vecm.eg <- VECM(zeroyld, lag=2, estim = "2OLS")
summary(vecm.eg)
```

```
# For the same data, fit a VECM with ECM.EngleGran(), from VGAMextra.
# Set ordtsDyn = 1 for compatibility!
fit.ECM.3 <- vglm(cbind(long.run, short.run) ~ 1, ECM.EngleGran(ecm.order = c(2, 2),
                                  resids.pattern = "neither", ordtsDyn = 1),
                  data = zeroyld, trace = TRUE)
coef.ECM.3 <- coef(fit.ECM.3, matrix = TRUE)

#### Compare results
vecm.eg$coefficients                                  # From tsDyn
t(coef.ECM.3[, 1:2][c(2, 1, 5, 3, 6, 4 ),][, 2:1]) # FROM VGAMextra

## End(Not run)
```

`expMlink` *Link functions for the mean of 1–parameter continuous distributions: The exponential distribution.*

Description

Computes the `expMlink` transformation, its inverse and the first two derivatives.

Usage

```
expMlink(theta, location = 0, bvalue = NULL, inverse = FALSE,
         deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ although may be η depending on the other parameters. See below for further details.
`location`	This is a known location parameter. Same as `location` in `exponential`.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

This is a link function to model the mean of the exponential distribution, `exponential`. It is defined as

$$\eta = \log(A + \lambda^{-1}),$$

where $\lambda > 0$ is a rate parameter and A is a known location parameter, same as `exponential`.

Numerical values of λ out of range may result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the `expMlink` transformation of `theta` when `inverse = FALSE`. If `inverse = TRUE`, then the inverse `exp(theta - A)^(-1)`.

For `deriv = 1`, d `eta` / d `theta` when `inverse = FALSE`. If `inverse = TRUE`, then d `theta` / d `eta` as a function of `theta`.

Similarly, when `deriv = 2`, the second derivatives in terms of `theta` are returned.

Note

Numerical instability may occur for values `theta` too close to zero. Use argument `bvalue` to replace them before computing the link.

If `theta` is character, then arguments `inverse` and `deriv` are ignored. See `Links` for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`exponential`, `Links`.

Examples

```
##  E1. Modelling the mean of the exponential distribution  ##
   set.seed(17010402)
   nn <- 100
   edata <- data.frame(x2 = runif(nn) - 0.5, x3 = runif(nn) - 0.5)
   edata <- transform(edata, eta = 0.2 - 0.7 * x2 + 1.9 * x3)

           #----- The mean is a function of 'x2' and 'x3' ------#
   edata <- transform(edata, rate = expMlink(eta, inverse = TRUE))

   edata <- transform(edata, y = rexp(nn, rate = rate))
   with(edata, stem(y))
   with(edata, hist(y))

   exp.fit <- vglm(y ~ x2 + x3, exponential(link = "expMlink"),
                   data = edata, zero = NULL, trace = TRUE, crit = "log")
  coef(exp.fit, matrix = TRUE)
  summary(exp.fit)

## E2. expMlink() and its inverse ##
   theta <- 0.1 + 1:5
   location <- 1.5
   my.diff <- theta - expMlink(expMlink(theta = theta,
                location = location), location = location, inverse  =TRUE)
   summary(my.diff)     # Zero


 ## E3. Special values in a matrix ##
```

```
theta <- matrix(c(Inf, -Inf, NA, NaN, 1 , 2), ncol = 3, nrow = 2)
expMlink(theta = theta, location = location)
```

---

expQlink *Link functions for the quantiles of several 1–parameter continuous distributions.*

Description

Computes the expQlink transformation, its inverse and the first two derivatives.

Usage

```
expQlink(theta, p = stop("Argument 'p' must be entered."),
         bvalue = NULL, inverse = FALSE,
         deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. This is θ although may be η depending on the other parameters. See below for further details.
p	Numeric. A prespecified number between 0 and 1. The particular p–quantile to be modelled. For example, $p = 0.5$ means that the median is considered by this link function.
bvalue, inverse, deriv, short, tag	See Links.

Details

This is a link function to model any fixed quantile, say ξ_p, of the exponential distribution. It is called the expQlink transformation and is defined as

$$\log(1-p)^{-1/\lambda},$$

where λ is positive as in exponential.

Numerical values of λ or p out of range may result in Inf, -Inf, NA or NaN.

Value

With deriv = 0, the expQlink transformation of theta for prespecified p when inverse = FALSE. If inverse = TRUE, then the inverse -log(1 - p)/theta.

For deriv = 1, this link function returns d eta / d theta when inverse = FALSE. If inverse = TRUE, then d theta / d eta as a function of theta.

Similarly, when deriv = 2, the second derivatives in terms of theta are returned.

Note

Numerical instability may occur for values `theta` too close to zero. Use argument `bvalue` to replace them before computing the link.

If `theta` is character, then arguments `inverse` and `deriv` are ignored. See `Links` for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`exponential`, `Links`.

Examples

```
## E1. expQlink() and its inverse ##
   p <- 0.25            # Modelling the first quartile.
   my.theta <- seq(0, 5, by = 0.1)[-1]
   my.diff <- my.theta - expQlink(expQlink(my.theta, p = p), p = p, inverse  =TRUE)
   summary(my.diff)     # Zero


 ## E2. Special values ##
   expQlink(theta = c(Inf, -Inf, NA, NaN), p = p)


 ##  E3. Plot of expQlink() for different quantiles  ##
   plot(expQlink(my.theta, p = p) ~ my.theta,
        type = "l", lty = "dotted", col = "blue", lwd = 2,
        main = "expQink(p) transformation", xlab = "theta", ylab = "expQLink",
        xlim = c(-0.5, 5), ylim = c(-0.5, 5))
   abline(h = 0, v = 0, lwd = 2)
  lines(my.theta, expQlink(my.theta, p = 0.50),  col = "green", lty = "dotted", lwd = 2)
   lines(my.theta, expQlink(my.theta, p = 0.75),  col = "red", lty = "dotted", lwd = 2)
   legend(2, 4,  c("p = 0.25", "p = 0.50", "p = 0.75"), col = c("blue", "green", "red"),
          lwd = c(2, 2, 2), lty = c("dotted", "dotted", "dotted"))
```

`gamma1Qlink` *Link functions for the quantiles of several 1–parameter continuous distributions*

Description

Computes the `gamma1Qlink` transformation, its inverse and the first two derivatives.

Usage

```
gamma1Qlink(theta, p = stop("Argument 'p' must be specified."),
            bvalue = NULL, inverse = FALSE,
            deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. It is θ by default although it could be η depending upon other arguments. See `Links` for further details about this.
`p`	A numeric vector of p–quantiles (numbers between 0 and 1) to be modeled by this link function.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

This link function has been specifically designed to model any p–quantile of the 1–parameter gamma distribution, `gamma1`, in the VGLM/VGAM context. It is defined as

$$\eta = \log \texttt{qgamma}(\texttt{p}, \texttt{shape} = s),$$

where s is a positive shape parameter as in `gamma1`, whilst `qgamma`() is the quantile function `qgamma`.

The inverse of the `gamma1Qlink` cannot be expressed in closed form. Instead, the inverse image, s_η, of η is numerically approximated by `newtonRaphson.basic`.

Numerical values of s or p out of range will result in `Inf`, `-Inf`, `NA` or `NaN` correspondingly.

Arguments `inverse` and `deriv` are dismissed if `theta` is character.

Value

For `deriv` = `0`, the `gamma1Qlink` transformation of `theta`, when inverse = FALSE. If `inverse` = `TRUE`, then `theta` becomes η, and therefore, the approximate inverse image of η is returned.

For `deriv` = `1`, the partial derivative d `eta` / d `theta` is returned, if `inverse` = `FALSE`. If `inverse` = `TRUE`, then the reciprocal d `theta` / d `eta` as a function of `theta`.

If deriv = 2, then the second order derivatives as a function of `theta`.

Note

Numerical instability may occur for values `theta` too large, or too close to 0.0. Use argument `bvalue` to replace them before computing the link.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`gamma1`, `qgamma`, `Links`.

Examples

```
## E1. gamma1QLink() and values causing NaNs or out of range  ##

p <- 0.75                             # The third quartile is of interest.
my.s <- seq(0, 5, by = 0.1)[-1]

max(my.s - gamma1Qlink(gamma1Qlink(my.s, p = p), p = p, inverse  =TRUE)) ## Zero

## E2. Special values of theta ##
gamma1Qlink(theta = c(-0.15, -0.10, 0, 1:10) , p = p, inverse  = FALSE)  ## NaNs
gamma1Qlink(theta = c(-5, -3, 0, 1:10) , p = p, inverse  = TRUE)        ## Out of range


## E3. Plot of gamma1QLink() and its inverse. ##


  # gamma1Qlink()
  plot(gamma1Qlink(theta = my.s, p = p) ~ my.s,
       type = "l", col = "blue", lty = "dotted", lwd = 3,
       xlim = c(-0.1, 5), ylim = c(-5, 15), las = 1,
       main = c("Blue is gamma1Qlink(), green is the inverse"),
       ylab = "gamma1Qlink transformation", xlab = "theta")
   abline(h = 0, v = 0, lwd = 2)

   # The inverse
   lines(my.s, gamma1Qlink(theta = my.s, p = p, inverse = TRUE),
         col = "green", lwd = 2, lty = "dashed")

    # The identity function, for double-checking.
   lines(my.s, my.s, lty = "dotted")
```

---

gammaRff *2–parameter Gamma Distribution*

Description

Estimates the 2–parameter gamma distribution by maximum likelihood. One linear predictor models the mean.

Usage

```
gammaRff(zero = "shape", lmu = "gammaRMlink",
         lrate = NULL, lshape = "loglink",
          irate = NULL,   ishape = NULL, lss = TRUE)
```

Arguments

`zero`	Specifies the parameters to be modelled as intercept–only. See `CommonVGAMffArguments`.
`lmu`	The link function applied to the gamma distribution mean, i.e., `gammaRMlink`.
`lrate, lshape, irate, ishape, lss`	Same as `gammaR`.

Details

This family function slightly enlarges the functionalities of `gammaR` by directly modelling the mean of the gamma distribution. It performs very much like `gamma2`, but involves the ordinary (not reparametrized) density, given by

$$f(y;\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} e^{-\beta y} y^{\alpha-1},$$

Here, α and β are positive shape and rate parameters as in `gammaR`.

The default linear predictors are $\eta 1 = \texttt{gammaRMlink}(\alpha;\beta) = \log\mu = \log(\alpha/\beta)$, and $\eta 2 = \log\alpha$, unlike $\eta 1 = \log\beta$ and $\eta 2 = \log\alpha$ from `gammaR`.

`lmu` overrides `lrate` and no link other than `gammaRMlink` is a valid entry (`lmu`). To mimic `gammaR` simply set `lmu = NULL` and `lrate = "loglink"`. The mean (μ) is returned as the fitted values.

`gammaRff` differs from `gamma2`. The latter estimates a re-parametrization of the gamma distribution in terms μ and α. This **VGAM** family function does not handle censored data.

Value

An object of class `"vglm"`. See `vglm-class` for full details.

Note

The parameters α and β match the arguments `shape` and $rate$ of `rgamma`.

Multiple responses are handled.

Author(s)

V. Miranda and Thomas W. Yee.

References

Yee, T. W. (2015) *Vector Generalized Linear and Additive Models: With an Implementation in R*. Springer, New York, USA.

See Also

`gammaRMlink`, `CommonVGAMffArguments`, `gammaR`, `gamma2`, `Links`.

Examples

```
### Modelling the mean in terms of x2, two responses.

  set.seed(2017022101)
  nn <- 80
  x2 <- runif(nn)
  mu <- exp(2 + 0.5 * x2)

# Shape and rate parameters in terms of 'mu'
  shape <- rep(exp(1), nn)
  rate <- gammaRMlink(theta = log(mu), shape = shape,
                          inverse = TRUE, deriv = 0)

# Generating some random data
  y1 <- rgamma(n = nn, shape = shape, rate =  rate)
  gdata <- data.frame(x2 = x2, y1 = y1)
  rm(y1)

# lmu = "gammaRMlink" replaces lshape, whilst lrate = "loglink"
  fit1 <- vglm(cbind(y1, y1) ~ x2,
               gammaRff(lmu = "gammaRMlink", lss = TRUE, zero = "shape"),
               data = gdata, trace = TRUE, crit = "log")
   coef(fit1, matrix = TRUE)
   summary(fit1)

# Comparing fitted values with true values.
  compare1 <- cbind(fitted.values(fit1)[, 1, drop = FALSE], mu)
  colnames(compare1) <- c("Fitted.vM1", "mu")
  head(compare1)


### Mimicking gammaR. Note that lmu = NULL.
  fit2 <- vglm(y1 ~ x2, gammaRff(lmu = NULL, lrate = "loglink",
                          lshape = "loglink", lss = FALSE, zero = "shape"),
               data = gdata, trace = TRUE, crit = "log")

# Compare fitted values with true values.
  compare2 <- with(gdata, cbind(fitted.values(fit2), y1, mu))
  colnames(compare2) <- c("Fitted.vM2", "y", "mu")
  head(compare2)


### Fitted values -- Model1 vs Fitted values -- Model2
  fit1vsfit2 <- cbind(fitted.values(fit1)[, 1, drop = FALSE],
                      fitted.values(fit2))
  colnames(fit1vsfit2) <- c("Fitted.vM1", "Fitted.vM2")
  head(fit1vsfit2)

### Use gamma2()
   fit3 <- vglm(y1 ~ x2, gamma2,
               data = gdata, trace = TRUE, crit = "log")
```

```
fit1.fit3 <- cbind(fitted.values(fit1)[, 1, drop = FALSE],
                   fitted.values(fit2), fitted.values(fit3))
colnames(fit1.fit3) <- c("Fitted.vM1", "Fitted.vM2", "Fitted.vM3")
head(fit1.fit3)
```

`gammaRMlink` *Link functions for the mean of 2–parameter continuous distributions: The gamma distribution.*

Description

Computes the `gammaRMlink` transformation, its inverse and the first two derivatives.

Usage

```
gammaRMlink(theta, shape = NULL, wrt.param = NULL,
            bvalue = NULL, inverse = FALSE,
            deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ ('rate' parameter) but iy may be η depending on the other parameters. See below for further details.
`shape`	The shape parameter. Same as `gammaRff`.
`wrt.param`	Positive integer, either 1 or 2. The partial derivatives are computed with respect to one of the two linear predictors involved with this link. Further details listed below.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

The link to model the mean of the 2–parameter gamma distribution.

The `gammaRMlink` transformation, *for given* α ('shape' parameter), is defined as

$$\eta = \eta(\alpha; \beta) = \log \frac{\alpha}{\beta},$$

where $\beta > 0$ is a *rate* parameter.

This link is expressly a function of β, i.e. θ, therefore α (*shape*) must be entered at every call.

Numerical values of α or β out of range may result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For deriv = 0, the gammaRMlink transformation of theta, i.e. β, when inverse = FALSE. If inverse = TRUE, then θ becomes η, and the inverse, α * exp(-theta), *for given* α, is returned.

For deriv = 1, theta becomes $\theta = (\beta, \alpha)$ $=(\theta 1, \theta 2)$, and $\eta = (\eta 1, \eta 2)$, and then, the argument wrt.param must be considered:

A) If inverse = FALSE, then d eta1 / d theta1 when wrt.param = 1, and d eta1 / d theta2 if wrt.param = 2.

B) For inverse = TRUE, this function returns d theta1 / d eta1 and d theta2 / d eta1 conformably arranged in a matrix, if wrt.param = 1, as a function of θ_i, $i = 1, 2$. Also, when wrt.param = 2, a matrix with columns $d$theta1 / d eta2 and $d$theta2 / d eta2 is returned.

Similarly, when deriv = 2, the second derivatives in terms of theta are returned.

Note

Numerical instability may occur for values theta too close to zero. Use argument bvalue to replace them before computing the link.

If theta is character, then arguments inverse and deriv are ignored. See Links for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

gammaRff, gammaR, Links.

Examples

```
   eta <- seq(-3, 3, by = 0.1) # this is eta = log(mu(b, a)).
   shape  <- rep(exp(0.8), length(eta))    # 'shape' argument.

## E1. Get 'rate' values.
  theta <- gammaRMlink(theta = eta, shape = shape, inverse = TRUE)  # rate

## Not run:
## E2. Plot theta vs. eta, 'shape' fixed.
  plot(theta, eta, type = "l", las = 1, ylab = "",
  main = "gammaRMlink(theta; shape)")

## End(Not run)

## E3. gammaRMlink() and its inverse ##
   etabis  <- gammaRMlink(theta = theta, shape = shape, inverse = FALSE)
   my.diff <- eta - etabis
   summary(my.diff)     # Zero

 ## E4. Special values arranged in a matrix ##
   bbeta <- matrix(eta[1:9], ncol = 3, nrow = 3)  #Ensure equal dimensions.
```

```
alpha <- matrix(c(Inf, -Inf, NA, NaN, -1 , 1, 0, -2, 2), ncol = 3, nrow = 3)
# The gammaRMlink transformation (log(a/b))
gammaRMlink(theta = bbeta, shape = alpha, inv = FALSE)   # NaNs produced.
# Same as
log(alpha/bbeta)
```

`gen.betaIImr` *Generalized Beta Distribution of the Second Kind family function*

Description

Maximum likelihood estimation of the 4-parameter generalized beta II distribution using Fisher scoring.

Usage

```
gen.betaIImr(lscale    = "loglink",
             lshape1.a = "loglink",
             lshape2.p = "loglink",
             lshape3.q = "loglink",
             iscale    = NULL,
             ishape1.a = NULL,
             ishape2.p = NULL,
             ishape3.q = NULL,
             imethod   = 1,
             lss       = TRUE,
             gscale    = exp(-5:5),
             gshape1.a = exp(-5:5),
             gshape2.p = exp(-5:5),
             gshape3.q = exp(-5:5),
             probs.y   = c(0.25, 0.50, 0.75),
             zero      = "shape" )
```

Arguments

`lscale`, `lshape1.a`, `lshape2.p`, `lshape3.q`
: Parameter link functions applied to the shape parameter `a`, scale parameter `scale`, shape parameter `p`, and shape parameter `q`. All four parameters are positive. See `Links` for more choices.

`iscale`, `ishape1.a`, `ishape2.p`, `ishape3.q`
: Optional initial values for `b`, `a`, `p` and `q`. Default is `NULL` for all of them, meaning initial values are computed internally.

`imethod`
: Initializing method to internally compute the initial values. Currently, only `method = 1` is handled.

`gscale, gshape1.a, gshape2.p, gshape3.q`
: Grid search initial values. See `CommonVGAMffArguments` for further information.

`zero`
: Numeric or Character vector. Position(s) or name(s) of the parameters/linear predictors to be modeled as intercept-only. Details at `CommonVGAMffArguments`.

`lss, probs.y`
: See `CommonVGAMffArguments` for important information.

Details

This distribution is most useful for unifying a substantial number of size distributions. For example, the Singh-Maddala, Dagum, Fisk (log-logistic), Lomax (Pareto type II), inverse Lomax, beta distribution of the second kind distributions are all special cases. Full details can be found in Kleiber and Kotz (2003), and Brazauskas (2002). The argument names given here are used by other families that are special cases of this family. Fisher scoring is used here and for the special cases too.

The 4-parameter generalized beta II distribution has density

$$f(y) = a y^{ap-1} / [b^{ap} B(p,q) \{1 + (y/b)^a\}^{p+q}]$$

for $a > 0$, $b > 0$, $p > 0$, $q > 0$, $y \geq 0$. Here B is the beta function, and b is the scale parameter `scale`, while the others are shape parameters. The mean is

$$E(Y) = b\,\Gamma(p + 1/a)\,\Gamma(q - 1/a)/(\Gamma(p)\,\Gamma(q))$$

provided $-ap < 1 < aq$; these are returned as the fitted values.

Value

An object of class `"vglmff"` (see `vglmff-class`). The object is used by modelling functions such as `vglm`, and `vgam`.

Warning

`zero` can be a **numeric** or a **character** vector specifying the position(s) or the name(s) (partially or not) of the linear predictors modeled as intercept–only. Numeric values can be entered as usual. If names are used, note that the linear predictors in this family function are

`c("scale", "shape1.a", "shape2.p", "shape3.q")`.

For simplicity, using names rather than numeric vectors is recommended.

Note

Paramaters `"shape1.a"`, `"shape2.p"`, `"shape3.q"` are modeled as intercept only, by default.

If the self-starting initial values fail, try experimenting with the initial value arguments, `iscale`, `ishape1.a`, `ishape2.p` and `ishape3.q` whose default is `NULL`. Also, the constraint $-ap < 1 < aq$ may be violated as the iterations progress so it is worth monitoring convergence, e.g., set `trace = TRUE`.

Successful convergence depends on choosing good initial values. This process might be difficult for this distribution, since 4 parameters are involved. Presently, only `method = 1` is internally handled to set initial values. It involves *grid search*, an internal implementation of the well-known grid search algorithm for exhaustive searching through a manually specified subset of the hyperparameter space.

Default value of `lss` is `TRUE` standing for the following order: location (b), shape1.a (a), shape2.p (p), shape3.q (q). In order to match the arguments of existing R functions, the option `lss = FALSE` might be set leading to switch the position of location (b) and shape1.a (a), only.

Author(s)

T. W. Yee and V. Miranda.

References

Brazauskas, V. (2002) Fisher information matrix for the Feller-Pareto distribution. *Statistics & Probability Letters*, **59**, 159–167.

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley Series in Probability and Statistics. Hoboken, New Jersey, USA.

McDonald, J. B. and Xu, Y. J. (1995) A generalization of the beta distribution with applications. *Journal of Econometrics*. **66**, p.133–152.

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. *Econometrica*, **52**, 647–663.

See Also

betaff, betaII, dagum, sinmad, fisk, lomax, inv.lomax, paralogistic, inv.paralogistic, genbetaIIDist.

Examples

```
#------------------------------------------------------------------------ #
# An example.- In this data set parameters 'shape1.a' and 'shape3.q' are
# generated in terms of x2.

set.seed(1003)
nn <- 200
gdata1  <- data.frame(x2 = runif(nn))
gdata   <- transform(gdata1,
           y1 = rgen.betaII(nn, scale = exp(1.1), shape1.a = exp(1.2 + x2),
                            shape2.p = exp(0.7) , shape3.q = exp(2.1 - x2)),
           y2 = rgen.betaII(nn, scale = exp(2.0), shape1.a = exp(1.8 + x2),
                            shape2.p = exp(2.3) , shape3.q = exp(1.9 - x2)),
           y3 = rgen.betaII(nn, scale = exp(1.5), shape1.a = exp(1.8),
                            shape2.p = exp(2.3) , shape3.q = exp(1.3)))

#------------------------------------------------------------------------#
# A single intercept-only model. No covariates.
# Note the use of (optional) initial values.
fit  <- vglm(y2 ~ 1,   #y3 ~ 1
             gen.betaIImr(lss = TRUE,
                          # OPTIONAL INITIAL VALUES
                           #iscale    = exp(1.5),
```

```
                            #ishape1.a = exp(1.8),
                            #ishape2.p = exp(2.3),
                            #ishape3.q = exp(1.3),

                           imethod = 1),
              data = gdata, trace = TRUE, crit = "loglik")

Coef(fit)
coef(fit, matrix = TRUE)
summary(fit)

#-------------------------------------------------------------------------#
# An intercept-only model. Two responses.
fit1 <- vglm(cbind(y2, y2) ~ 1,   # cbind(y1, y2)
             gen.betaIImr(lss = TRUE),
             data = gdata, trace = TRUE, crit = "loglik")

Coef(fit1)
coef(fit1, matrix = TRUE)
summary(fit1)
vcov(fit1, untransform = TRUE)

#-------------------------------------------------------------------------#
# An example incorporating one covariate. Constraints are set accordingly.
# x2 affects shape1.a and shape3.q.
# Note that the first option uses 'constraints', whilst in the second
# choice we use the argument 'zero' to 'set' the same constraints.

### Option 1.
c1 <- rbind(0, 1, 0, 0)
c2 <- rbind(0, 0, 0, 1)
mycons <- matrix( c(c1, c2), nc = 2, byrow = FALSE)

fit2 <- vglm(y1 ~ x2, gen.betaIImr(lss = TRUE, zero = NULL),
             data = gdata, trace = TRUE, crit = "loglik",
             constraints = list(x2 = mycons ))

coef(fit2, matrix = TRUE)
summary(fit2)
vcov(fit2)
constraints(fit2)


### Option 2.
fit3 <- vglm(y1 ~ x2,
             gen.betaIImr(lss = TRUE,
                          zero = c("scale", "shape2.p")),
             data = gdata, trace = TRUE, crit = "loglik")

coef(fit3, matrix = TRUE)
summary(fit3)
vcov(fit3)
constraints(fit3)
```

genbetaIIDist *The Generalized Beta Distribution of the Second King*

Description

Density, distribution function, inverse distribution (quantile function) and random generation for the Generalized Beta of the Second Kind (GB2).

Usage

```
dgen.betaII(x, scale = 1.0, shape1.a = 1.0, shape2.p = 1.0, shape3.q = 1.0,
           log = FALSE)
pgen.betaII(q, scale = 1.0, shape1.a = 1.0, shape2.p = 1.0, shape3.q = 1.0,
           lower.tail = TRUE, log.p = FALSE)
qgen.betaII(p, scale = 1.0, shape1.a = 1.0, shape2.p = 1.0, shape3.q = 1.0,
           lower.tail = TRUE, log.p = FALSE)
rgen.betaII(n, scale = 1.0, shape1.a = 1.0, shape2.p = 1.0, shape3.q = 1.0)
```

Arguments

`x, q`	Vector of quantiles.
`p`	Vector of probabilities.
`n`	Number of observations. If `length(n) > 1`, its length is taken to be the numbre required.
`scale, shape1.a, shape2.p, shape3.q`	Strictly positive scale and shape parameters.
`log, log.p, lower.tail`	Same meaning as in `Beta`

Details

The GB2 Distribution is defined by the probability density (pdf)

$$f(y) = \frac{a x^{ap-1}}{b^{ap} B(p,q)[1+(y/b)^a]^{p+q}},$$

for $y > 0$, and $b, a, p, q > 0$. Here, $B(p, q)$ is the beta function as in `beta`.

The GB2 Distribution and the Beta Distribution (see `Beta`) are linked, as follows: Let X be a random variable with the Beta density and parameters $p = shape_1$ and $q = shape_2$. Then, introducing additional $b = scale$ and $a = shape$ parameters, the variable

$$Y = \frac{(x/b)^a}{1+(x/b)^a}$$

has the GB2 Distribution, with parameters b, a, p, q.

The GB2 k^{th} moment exists for $-ap < k < aq$ and is given by

$$E(Y^k) = \frac{b^k B(p + k/a, q - k/a)}{B(p, q)}$$

or, equivalently,

$$E(Y^k) = \frac{b^k \Gamma(p + k/a)\Gamma(q - k/a)}{\Gamma(p)\Gamma(q)}).$$

Here, $\Gamma(\cdot)$ is the gamma function as in `gamma`.

Value

`dgen.betaII()` returns the density (p.d.f), `pgen.betaII()` gives the distribution function (p.d.f), `qgen.betaII()` gives the quantile function (Inverse Distribution function), and `rgen.betaII()` generates random numbers from the GB2 distribution.

Note

Values of the `shape2.p` parameter moderately close to zero may imply obtaning numerical values too close to zero or values represented as zero in computer arithmetic from the function `rgen.betaII()`.

Additionally, for specific values of the arguments `x, q, p` such as `Inf, -Inf, NaN` and `NA`, the functions `qgen.betaII()`, `pgen.betaII()` and `qgen.betaII()` will return the limit when the argument tends to such value.

In particular, the quantile `qgen.betaII()` retunrs zero for negative values and Inf for missed probabilities greater than 1.0.

Author(s)

V. Miranda and T. W. Yee

References

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, **ch.6**, p.255. Dover, New York, USA.

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley Series in Probability and Statistics. Hoboken, New Jersey, USA.

McDonald, J. B. and Xu, Y. J. (1995) A generalization of the beta distribution with applications. *Journal of Econometrics*, **66**, p.133–152.

McDonald, J. B. (1984) Some generalized functions for the size distribution of income. *Econometrica*, **52**, p.647–663.

See Also

`Beta`, `beta`.

Examples

```
# Setting parameters to both examples below.
b  <- exp(0.4)      # Scale parameter.
a  <- exp(0.5)      # Shape1.a
p  <- exp(0.3)      # Shape2.p
q  <- exp(1.4)      # Shape3.q

# (1) _______________
probs.y <- seq(0.0, 1.0, by = 0.01)
data.1  <- qgen.betaII(p = probs.y, scale = b, shape1.a = a,
                       shape2.p = p, shape3.q = q)
max(abs(pgen.betaII(q = data.1, scale = b, shape1.a = a,
                    shape2.p = p, shape3.q = q)) - probs.y) # Should be 0.

# (2)_________________
xx    <- seq(0, 10.0, length = 200)
yy    <- dgen.betaII(xx, scale = b, shape1.a = a, shape2.p = p, shape3.q = q)
qtl   <- seq(0.1, 0.9, by = 0.1)
d.qtl <- qgen.betaII(qtl, scale = b, shape1.a = a, shape2.p = p, shape3.q = q)
plot(xx, yy, type = "l", col = "red",
     main = "Red is the GB2 density, blue is the GB2 Distribution Function",
     sub  = "Brown dashed lines represent the 10th, ..., 90th percentiles",
     las = 1, xlab = "x", ylab = "", xlim = c(0, 3), ylim = c(0,1))
abline(h = 0, col = "navy", lty = 2)
abline(h = 1, col = "navy", lty = 2)
lines(xx, pgen.betaII(xx, scale = b, shape1.a = a,
                      shape2.p = b, shape3.q = q), col= "blue")
lines(d.qtl, dgen.betaII(d.qtl, scale = b, shape1.a = a,
                       shape2.p = p, shape3.q = q),
                       type ="h", col = "brown", lty = 3)
```

`geometricffMlink` *Link functions for the mean of 1–parameter discrete distributions: The Geometric Distribution.*

Description

Computes the `geometricffMlink` transformation, including its inverse and the first two derivatives.

Usage

```
geometricffMlink(theta, bvalue = NULL, inverse = FALSE,
                 deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See below for further details.
`bvalue, inverse, deriv, short, tag`	Details at `Links`

Details

This is a natural link function to model the mean of the (discret) geometric distribution, geometric, defined as the logarithmm of its mean, i.e.,

$$\eta = -\log \frac{p}{1-p} = -\texttt{logit}(p).$$

Here, p is the probability of succes, as in geometric.

While this link function can be used to model any parameter lying in $(0, 1)$, it is particularly useful for event-rate geometric data where the mean can be written in terms of some rate of events, say $\lambda = \lambda(\mathbf{x})$, as

$$\mu = \lambda(\mathbf{x})t,$$

and the time t (as $\log t$) can be easily incorporated in the analysis as an offset.

Under this link function the domain set for p is $(0, 1)$. Hence, values of ρ too close to the extremes, or out of range will result in `Inf`, `-Inf`, `NA` or `NaN`. Use argument `bvalue` to adequately replace them before computing the link function.

If `theta` is a character, arguments `inverse` and `deriv` are disregarded.

Value

For `deriv = 0`, the `geometricffMlink` transformation of `theta` when `inverse = FALSE`. When `inverse = TRUE` then `theta` becomes η, and `exp(-theta) / (exp(-theta) - 1)` is returned.

For `deriv = 1`, d `eta` / d `theta`, if `inverse = FALSE`, else the reciprocal d `theta` / d `eta` as a function of `theta`.

For `deriv = 2` the second order derivatives are correspondingly returned.

Warning

Numerical instability may occur if covariates are used leading to values of p out of range. Try to overcome this by using argument `bvalue`.

Note

This function may return `Inf` of `-Inf` for values of p too close to 0 and 1 respectively.

Author(s)

V. Miranda and T. W. Yee

See Also

geometric, Links, logitlink, logffMlink.

Examples

```
### Example 1  ###
my.probs <- ppoints(100)
geol.inv <-
    geometricffMlink(theta = geometricffMlink(theta = my.probs), # the inverse
                                inverse = TRUE) - my.probs
summary(geol.inv)     ## zero

###  Example 2. Special values of 'prob'  ###
my.probs <- c(-Inf, -2, -1, 0, 0.25, 0.75, 1.0, 5, Inf, NaN, NA)
rbind(probs = my.probs,
      geoffMlink = geometricffMlink(theta = my.probs),
      inv.geoffl = geometricffMlink(theta = my.probs, inverse = TRUE))


###  Example 3 Some probability link functions  ###


my.probs <- ppoints(100)

par(lwd = 2)
plot(my.probs, logitlink(my.probs), xlim = c(-0.1, 1.1), ylim = c(-5, 8),
     type = "l", col = "limegreen",
     ylab = "transformation", las = 1, main = "Some probability link functions")
lines(my.probs, geometricffMlink(my.probs), col = "gray50")
lines(my.probs, logffMlink(my.probs), col = "blue")
lines(my.probs, probitlink(my.probs), col = "purple")
lines(my.probs, clogloglink(my.probs), col = "chocolate")
lines(my.probs, cauchitlink(my.probs), col = "tan")
abline(v = c(0.5, 1), lty = "dashed")
abline(v = 0, h = 0, lty = "dashed")
legend(0.1, 8,
      c("geometricffMlink", "logffMlink","logitlink", "probitlink",
      "clogloglink", "cauchitlink"),
      col = c("gray50", "blue", "limegreen", "purple", "chocolate", "tan"),
      lwd = 1, cex = 0.5)
par(lwd = 1)
```

HKdata — *Air pollution and hospital admissions due to respiratory and cardiovascular causes, Hong Kong.*

Description

Daily air pollution levels and hospital admissions due to respiratory and cardiovascular causes, between 1 January 1994 and 31 December 1997, Hong Kong.

Usage

```
data(HKdata)
```

Format

This is a subset of the data analyzed in Xia and Tong (2006) which stores the following time series:

no Time vector,

cardio, resp Integer. Daily hospital admissions due to respiratory and cardiovascular causes, 1 January 1994 and 31 December 1997.

no2, so2, rsp, o3 Numeric. Daily mean average of NO_2, SO_2, respirable suspended particles (rsp; that is PM_{10}), and O_3, in parts per billion (ppb).

temp, hum Numeric. Daily mean average of temperature (Celsius deg.) and humidity (%)

mon, tue, wed, thur, fri, sat Factors with two levels. Weekdays/weekends indicator variables.

Source

Data set retrieved from

https://blog.nus.edu.sg/homepage/research/

References

Xia, Y. and Tong, H. (2006) Cumulative effects of air pollution on public health. *Statistics in Medicine*. **25(29)**, 3548-3559.

Examples

```
data(HKdata)
summary(HKdata[, -1])
```

---

inv.chisqDist *The Inverse Chi–squared Distribution*

Description

Density, CDF, quantile function and random number generator for the Inverse Chi–squared distribution.

Usage

```
dinv.chisq(x, df, log = FALSE)
pinv.chisq(q, df, lower.tail = TRUE, log.p = FALSE)
qinv.chisq(p, df, lower.tail = TRUE, log.p = FALSE)
rinv.chisq(n, df)
```

Arguments

`x, q, p, n`	Same as `Chisquare`.
`df, lower.tail, log, log.p`	Same as `Chisquare`.

Details

The inverse chi–squared distribution with non–negative `df` = ν degrees of freedom implemented here has density

$$f(x;\nu) = \frac{2^{-\nu/2} x^{-\nu/2-1} e^{-1/(2x)}}{\Gamma(\nu/2)},$$

where $x > 0$, and Γ is the `gamma` function.

The mean is $1/(\nu-2)$, for $\nu > 2$, and the variance is given by $2/[(\nu-2)^2(\nu-4)]$, for $\nu > 4$.

Also, as with `Chisquare`, the degrees of freedom can be non–integer.

Value

`dinv.chisq` returns the density, `pinv.chisq` returns the distribution function, `qinv.chisq` gives the quantiles, and `rinv.chisq` generates random numbers from this distribution.

Source

Specifically, it is the probability distribution of a random variable whose reciprocal follows a chi–squared distribution, i.e.,

If $Y \sim$ chi–squared(ν), then $1/Y \sim$ Inverse chi–squared(ν).

As a result, `dinv.chisq, pinv.chisq, qinv.chisq` and `rinv.chisq` use the functions `Chisquare` as a basis. Hence, invalid arguments will lead these functions to return `NA` or `NaN` accordingly.

Note

Yet to do: A *non–central* parameter as an argument, if amenable.

Two similar versions of the Inverse chi–squared distribution with ν degrees of freedom may be found in the literature, as follows:

Let $Y \sim$ chi–squared(ν), then

I. $1/Y \sim$ Inverse chi–squared(ν), and II. $\nu/Y \sim$ Inverse chi–squared(ν).

Here, the former, which is the popular version, has been implemented.

Author(s)

V. Miranda

References

Johnson, N.L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*. Chapters 18 (volume 1) and 29 (volume 2). Wiley, New York.

See Also

Chisquare, gamma.

Examples

```
##  Example 1  ##
nn <- 50; df <- 1.4
data.1   <- ppoints(nn)
data.q   <- qinv.chisq(-data.1, df = df, log.p = TRUE)
data.p   <- -log(pinv.chisq(data.q, df = df))
max(abs(data.p - data.1))     # Should be zero


##  Example 2  ##

xx    <- seq(0, 3.0, len = 301)
yy    <- dinv.chisq(xx, df = df)
qtl   <- seq(0.1, 0.9, by = 0.1)
d.qtl <- qinv.chisq(qtl, df = df)
plot(xx, yy, type = "l", col = "orange",
     main = "Orange is density, blue is cumulative distribution function",
     sub  = "Brown dashed lines represent the 10th, ... 90th percentiles",
     las = 1, xlab = "x", ylab = "", ylim = c(0, 1))
abline(h = 0, col= "navy", lty = 2)
lines(xx, pinv.chisq(xx, df = df), col = "blue")
lines(d.qtl, dinv.chisq(d.qtl, df = df), type ="h", col = "brown", lty = 3)
```

---

inv.chisqff *Inverse Chi–squared Distribution.*

Description

Maximum likelihood estimation of the degrees of freedom for an inverse chi–squared distribution using Fisher scoring.

Usage

```
inv.chisqff(link = "loglink", zero = NULL)
```

Arguments

link, zero	link is the link function applied to the degrees of freedom, leading to the unique linear predictor in this family function. By default, the link is loglink. zero allows to model the single linear predictor as intercept–only. For further details, see CommonVGAMffArguments.

Details

The inverse chi–squared distribution with $df = \nu \geq 0$ degrees of freedom implemented here has density

$$f(x;\nu) = \frac{2^{-\nu/2} x^{-\nu/2-1} e^{-1/(2x)}}{\Gamma(\nu/2)},$$

where $x > 0$, and Γ is the gamma function. The mean of Y is $1/(\nu - 2)$ (returned as the fitted values), provided $\nu > 2$.

That is, while the expected information matrices used here are valid in all regions of the parameter space, the regularity conditions for maximum likelihood estimation are satisfied only if $\nu > 2$. To enforce this condition, choose `link = logoff(offset = -2)`.

As with, chisq, the degrees of freedom are treated as a parameter to be estimated using (by default) the link loglink. However, the mean can also be modelled with this family function. See inv.chisqMlink for specific details about this.

This family **VGAM** function handles multiple responses.

Value

An object of class `"vglmff"`. See vglmff-class for further details.

Warning

By default, the single linear/additive predictor in this family function, say $\eta = \log dof$, can be modeled in terms of covariates, i.e., `zero = NULL`. To model η as intercept–only set `zero = "dof"`.

See zero for more details about this.

Note

As with chisq or Chisquare, the degrees of freedom are non–negative but allowed to be non–integer.

Author(s)

V. Miranda.

See Also

loglink, CommonVGAMffArguments, inv.chisqMlink, zero.

Examples

```
set.seed(17010504)
dof   <- 2.5
yy    <- rinv.chisq(100, df = dof)
ics.d <- data.frame(y = yy)            # The data.
```

```
fit.inv <- vglm(cbind(y, y) ~ 1, inv.chisqff,
                data = ics.d, trace = TRUE, crit = "coef")
Coef(fit.inv)
summary(fit.inv)
```

`inv.chisqMlink` *Link functions for the mean of 1–parameter continuous distributions: The inverse chi–squared distribution.*

Description

Computes the `inv.chisqMlink` transformation, its inverse and the first two derivatives.

Usage

```
inv.chisqMlink(theta, bvalue = NULL, inverse = FALSE,
               deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ by default but may be η depending on the other parameters. See below for further details.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

This link functions models the mean of the inverse chi–squared distribution, `inv.chisqff`.

It is defined as

$$\eta = -\log(df - 2),$$

where df denotes the (non–negative) degrees of freedom, as in `inv.chisqff`.

Notice, however, that $df > 2$ is required for the mean of this distribution to be real. Consequently, the domain set for `df` for this link function is $(2, \infty)$.

Numerical values of df out of range will result in `NA` or `NaN`.

Value

For `deriv = 0`, the `inv.chisqMlink` transformation of `theta` when `inverse = FALSE`. If `inverse = TRUE`, then the inverse `exp(-theta) + 2`.

For `deriv = 1`, d `eta` / d `theta` when `inverse = FALSE`. If `inverse = TRUE`, then d `theta` / d `eta` as a function of `theta`.

When `deriv = 2`, the second derivatives in terms of `theta` are returned.

Note

Numerical instability may occur for values `theta` too large or possibly, too close to 2. Use argument `bvalue` to replace them before computing the link.

If `theta` is character, then arguments `inverse` and `deriv` are ignored. See `Links` for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`inv.chisqff`, `Links`.

Examples

```
##  E1. Modelling the mean of the exponential distribution  ##
   set.seed(17010502)
   dof <- 2.5
   isq.data <- data.frame(x2 = runif(100, 0, 1))
   isq.data <- transform(isq.data, y = rinv.chisq(n = 100, df = dof + x2))


   hist(isq.data$y)


   fit.inv <- vglm(y ~ x2, family = inv.chisqff(link = "inv.chisqMlink"),
                   data = isq.data, trace = TRUE )
   coef(fit.inv, matrix = TRUE)
   summary(fit.inv)

## E3. Special values in a matrix ##
  (theta <- matrix(c(Inf, -Inf, NA, NaN, 1 , 2, 3, 4), ncol = 4, nrow = 2))
  inv.chisqMlink(theta = theta)   ## NaNs for df = theta <= 2

## E2. inv.chisqMlink() and its inverse ##
  theta <- 0.1 + 1:5  # dof = df
  my.diff <- theta - inv.chisqMlink(inv.chisqMlink(theta = theta), inverse  =TRUE)
  summary(my.diff)     # Zero
```

`invgamma2mr` *2 - parameter Inverse Gamma Distribution*

Description

Estimates the 2-parameter Inverse Gamma distribution by maximum likelihood estimation.

Usage

```
invgamma2mr(lmu      = "loglink",
            lshape   = logofflink(offset = -2),
            parallel = FALSE,
            ishape   = NULL,
            imethod  = 1,
            zero     = "shape")
```

Arguments

lmu, lshape	Link functions applied to the (positives) *mu* and *shape* parameters (called μ and a respectively), according to gamma2. See CommonVGAMffArguments for further information.
parallel	Same as gamma2. Details at CommonVGAMffArguments.
ishape	Optional initial value for *shape*, same as gamma2
imethod	Same as gamma2.
zero	Numeric or character vector. Position or name(s) of the parameters/linear predictors to be modeled as intercept–only. Default is "shape". Details at CommonVGAMffArguments.

Details

The Gamma distribution and the Inverse Gamma distribution are related as follows:Let *X* be a random variable distributed as $Gamma(a, \beta)$, where $a > 0$ denotes the *shape* parameter and $\beta > 0$ is the *scale* paramater. Then $Y = 1/X$ is an Inverse Gamma random variable with parameters *scale* = a and *shape* = $1/\beta$.

The Inverse Gamma density function is given by

$$f(y; \mu, a) = \frac{(a-1)^a \mu^a}{\Gamma(a)} y^{-a-1}\, e^{-\mu(a-1)/y},$$

for $\mu > 0$, $a > 0$ and $y > 0$. Here, $\Gamma(\cdot)$ is the gamma function, as in gamma. The mean of *Y* is $\mu = \mu$ (returned as the fitted values) with variance $\sigma^2 = \mu^2/(a-2)$ if $a > 2$, else is infinite. Thus, the *link function* for the *shape* parameter is logloglink. Then, by default, the two linear/additive predictors are $\eta_1 = \log(\mu)$, and $\eta_2 = \log(a)$, i.e in the VGLM context, $\eta = (log(\mu), loglog(a)$

This **VGAM** family function handles *multiple* reponses by implementing Fisher scoring and unlike gamma2, the working-weight matrices are *not* diagonal.

The Inverse Gamma distribution is right-skewed and either for small values of a (plus modest μ) or very large values of μ (plus moderate $a > 2$), the density has values too close to zero.

Value

An object of class "vglmff" (see vglmff-class). The object is used by modelling functions such as vglm and vgam.

Warning

Note that `zero` can be a **numeric** or a **character** vector specifying the position of the names (partially or not) of the linear predictor modeled as intercept only. In this family function such names are

`c("mu", "shape")`.

Numeric values can be entered as usual. See `CommonVGAMffArguments` for further details.

Note

The response must be strictly positive.

If `mu` and `shape` are vectors, then `rinvgamma(n = n, shape = shape, scale = mu/(shape - 1)` will generate random inverse gamma variates of this parameterization, etc.; see `invgammaDist`.

Given the math relation between the Gamma and the Inverse Gamma distributions, the parameterization of this **VGAM** family function underlies on the parametrization of the 2-parameter gamma distribution described in the monograph

Author(s)

Victor Miranda and T. W. Yee

References

McCullagh, P. and Nelder, J. A. (1989) *Generalized Linear Models*, 2nd ed. London, UK. Chapman & Hall.

See Also

`invgammaDist`, `gamma2` for the 2-parameter gamma distribution, `GammaDist`, `CommonVGAMffArguments`,

Examples

```
#-------------------------------------------------------------------------#
# Essentially fitting a 2-parameter inverse gamma distribution
# with 2 responses.

set.seed(101)
y1 = rinvgamma(n = 500, scale = exp(2.0), shape = exp(2.0))
y2 = rinvgamma(n = 500, scale = exp(2.5), shape = exp(2.5))
gdata <- data.frame(y1, y2)

fit1 <- vglm(cbind(y1, y2) ~ 1,
            family = invgamma2mr(zero = NULL,

                                 # OPTIONAL INITIAL VALUE
                                 # ishape = exp(2),

                                 imethod = 1),
            data = gdata, trace = TRUE)
```

```
Coef(fit1)
c(Coef(fit1), log(mean(gdata$y1)), log(mean(gdata$y2)))
summary(fit1)
vcov(fit1, untransform = TRUE)

#-------------------------------------------------------------------------#
# An example including one covariate.
# Note that the x2 affects the shape parameter, which implies that both,
# 'mu' and 'shape' are affected.
# Consequently, zero must be set as NULL !

x2    <- runif(1000)
gdata <- data.frame(y3 = rinvgamma(n = 1000,
                                   scale = exp(2.0),
                                   shape = exp(2.0 + x2)))

fit2 <- vglm(y3 ~ x2,
             family = invgamma2mr(lshape = "loglink", zero = NULL),
             data = gdata, trace = TRUE)

coef(fit2, matrix = TRUE)
summary(fit2)
vcov(fit2)
```

`invgammaDist` *The Inverse Gamma Distribution*

Description

Density, distribution function, quantile function and random numbers generator for the Inverse Gamma Distribution.

Usage

```
dinvgamma(x, scale = 1/rate, shape, rate = 1, log = FALSE)
pinvgamma(q, scale = 1/rate, shape, rate = 1, lower.tail = TRUE, log.p = FALSE)
qinvgamma(p, scale = 1/rate, shape, rate = 1, lower.tail = TRUE, log.p = FALSE)
rinvgamma(n, scale = 1/rate, shape, rate = 1)
```

Arguments

`x, q, p, n`	Same as `GammaDist`.
`scale, shape`	Scale and shape parameters, same as `GammaDist`. Both must be positive.
`rate`	Same as `GammaDist`.
`log, log.p, lower.tail`	Same as `GammaDist`.

Details

The Inverse Gamma density with parameters scale = b and shape = s is given by

$$f(y) = \frac{b^s}{\Gamma(s)} y^{-s-1} e^{-b/y},$$

for $y > 0$, $b > 0$, and $s > 0$. Here, $\Gamma(\cdot)$ is the gamma function as in `gamma`

The relation between the Gamma Distribution and the Inverse Gamma Distribution is as follows:

Let X be a random variable distributed as Gamma (b, s), then $Y = 1/X$ is distributed as Inverse Gamma $(1/b, s)$. It is worth noting that the math relation between the *scale* paramaters of both, the Inverse Gamma and Gamma distributions, is inverse.

Thus, algorithms of *dinvgamma(), pinvgamma(), qinvgamma()* and *rinvgamma()* underlie on the algorithms `GammaDist`.

Let Y distributed as Inverse Gamma (b, s). Then the k^{th} moment of Y exists for $-\infty < k < s$ and is given by

$$E[Y^k] = \frac{\Gamma(s-k)}{\Gamma(s)} b^k.$$

The mean (if $s > 1$) and variance (if $s > 2$) are

$$E[Y] = \frac{b}{(s-1)}; \quad Var[Y] = \frac{b^2}{(s-1)^2 \times (s-2)}.$$

Value

`dinvgamma()` returns the density, `pinvgamma()` gives the distribution function, `qinvgamma()` gives the quantiles, and `rinvgamma()` generates random deviates.

Warning

The order of the arguments *scale* and *shape* does not match `GammaDist`.

Note

Unlike the `GammaDist`, small values of a (plus modest μ) or very large values of μ (plus moderate $a > 2$), generate Inverse Gamma values so near to zero. Thus, `rinvgamma` in `invgammaDist` may return either values too close to zero or values represented as zero in computer arithmetic.

In addition, function `dinvgamma` will return zero for $x = 0$, which is the limit of the Inverse Gamma density when $'x'$ tends to zero.

Author(s)

V. Miranda and T. W. Yee.

References

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley Series in Probability and Statistics. Hoboken, New Jersey, USA.

See Also

GammaDist, gamma.

Examples

```
# Example 1._____________________
n        <- 20
scale    <- exp(2)
shape    <- exp(1)
data.1   <- runif(n, 0, 1)
data.q   <- qinvgamma(-data.1, scale = scale, shape = shape, log.p = TRUE)
data.p   <- -log(pinvgamma(data.q, scale = scale, shape = shape))
arg.max  <- max(abs(data.p - data.1))     # Should be zero


# Example 2._____________________
scale <- exp(1.0)
shape <- exp(1.2)
xx    <- seq(0, 3.0, len = 201)
yy    <- dinvgamma(xx, scale = scale, shape = shape)
qtl   <- seq(0.1, 0.9, by = 0.1)
d.qtl <- qinvgamma(qtl, scale = scale, shape = shape)
plot(xx, yy, type = "l", col = "orange",
     main = "Orange is density, blue is cumulative distribution function",
     sub  = "Brown dashed lines represent the 10th, ... 90th percentiles",
     las = 1, xlab = "x", ylab = "", ylim = c(0, 1))
abline(h = 0, col= "navy", lty = 2)
lines(xx, pinvgamma(xx, scale = scale, shape = shape), col = "blue")
lines(d.qtl, dinvgamma(d.qtl, scale = scale, shape = shape),
      type ="h", col = "brown", lty = 3)
```

---

invweibull2mr *2- parameter Inverse Weibull Distribution*

Description

Maximum likelihood estimation of the 2-parameter Inverse Weibull distribution. No observations should be censored.

Usage

```
invweibull2mr(lscale  = loglink,
              lshape  = logofflink(offset = -2),
              iscale  = NULL,
              ishape  = NULL,
              imethod = 2,
```

```
lss     = TRUE,
gscale  = exp(-4:4),
gshape  = exp(-4:4),
probs.y = c(0.25, 0.50, 0.75),
zero    = "shape")
```

Arguments

lscale, lshape	Parameter link functions applied to the (positive) shape parameter (called a below) and (positive) scale parameter (called b below). Given that the shape parameter must be greater than 2, lshape = logofflink(offset = -2) by default. See Links for more choices.
iscale, ishape	Optional initial values for the shape and scale parameters.
gscale, gshape	See CommonVGAMffArguments.
lss, probs.y	Details at CommonVGAMffArguments.
imethod	Initializing method internally implemented. Currently only the values 1 and 2 are allowed and NO observations should be censored.
zero	Numeric or character vector. The position(s) of the name(s) of the parameters/linear predictors to be modeled as intercept–only. Default is "shape". Details at CommonVGAMffArguments

Details

The Weibull distribution and the Inverse Weibull distributions are related as follows:

Let X be a Weibull random variable with paramaters scale =b and shape =a. Then, the random variable $Y = 1/X$ has the Inverse Weibull density with parameters scale = $1/b$ and shape = a.

The Inverse weibull density for a response Y is given by

$$f(y; a, b) = a(b^a)y^{-a-1} \exp[-(y/b)^(- a)]$$

for $a > 0$, $b > 0$, $y > 0$.

The mean, that is returned as the fitted values, (if $a > 1$) and the variance (if $a > 2$) are

$$E[Y] = b\,\Gamma(1 - 1/a); \quad Var[Y] = b^2\,[\Gamma(1 - 2/a) - (\Gamma(1 - 1/a))^2].$$

Fisher scoring is used to estimate both parameters. Although the expected information matrices used are valid in all regions of the parameter space, the regularity conditions for maximum likelihood estimation (MLE) are satisfied only if $a > 2$ (according to Kleiber and Kotz (2003)). If this is violated then a warning message is issued. To enforce $a > 2$, it has been set by default that lshape = logofflink(offset = 2).

As a result of the math relation between the Weibull and the Inverse Weibull distributions, regularity conditions for inference for the latter, are the following:

if $a \leq 1$ then the MLE's are not consisten, if $1 < a < 2$ then MLEs exist but are not assymptotically normal, if $a = 2$, the MLE's exist and are normal and asymptotically efficient but the convergence rate is slower compared when $a > 2$. If $a > 2$, then the MLE's have classical asymptotic properties.

Value

An object of class "vglmff" (see vglmff-class). The object is used to model special models such as vglm and vgam.

Warning

Note that zero can be a **numeric** or a **character** vector specifying the position of the names (partially or not) of the linear predictor modeled as intercept only. In this family function these names are

c("scale", "shape").

Numeric values can be entered as usual. See CommonVGAMffArguments for further details. For simplicity, the second choice is recommended.

If the shape parameter is less than two (i.e. less than exp(0.69315)), then misleading inference may result ! (see above the regularity condition for the 'variance'), e.g., in the summary and vcov of the object.

However, the larger the shape parameter is (for instance, greater than exp(2.5), plus reasonable scale), the more unstable the algorithm may become. The reason is that inverse weibull densities under such conditions are highly peaked and left skewed. Thus, density values are too close to zero (or values represented as zero in computer arithmetic).

Note

By default, the shape paramater is modeled as intercept only.

Successful convergence depends on having reasonably good initial values. If the initial values chosen by this function are not good, make use the two initial value arguments, iscale and ishape.

This **VGAM** family function currently handles *multiple responses* however, it does not handle censored data. This feature will be considered in a later version of the package.

The Inverse Weibull distribution, which is that of $Y = 1/X$ where X has the Weibull density, is known as the log-Gompertz distribution.

Author(s)

Victor Miranda and T. W. Yee.

References

Harper, W. V., Eschenbach, T. G. and James, T. R. (2011) Concerns about Maximum Likelihood Estimation for the Three-Parameter Weibull Distribution: Case Study of Statistical Software. *The American Statistician*, **65(1)**, 44-54.

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*, Hoboken, NJ, USA: Wiley-Interscience.

Johnson, N. L. and Kotz, S. and Balakrishnan, N. (1994) *Continuous Univariate Distributions*, 2nd edition, Volume 1, New York: Wiley.

See Also

invweibullDist, weibullR.

Examples

```
#------------------------------------------------------------------------#
# Here, covariate 'x2' affects the scale parameter.
# See how data is generated.

set.seed(102)
wdata <- data.frame(x2 = runif(nn <- 1000))  # Complete data
wdata <- transform(wdata,
           y1 = rinvweibull(nn, scale = exp(2.5 - (0.5) * x2),
                            shape = exp(1.5) ),

           y2 = rinvweibull(nn, scale = exp(1.5 + x2),
                            shape = exp(1.25) ))

#------------------------------------------------------------------------#
# Fitting the Inverse Weibull distribution accordingly.
# Note that multiple responses are handled.

fit1 <- vglm(cbind(y1, y2) ~ x2,
             invweibull2mr(zero = "shape",
                           # OPTIONAL INITIAL VALUE. Be carefull here when
                           # entered initial value. Sensitive distribution
                             ishape = exp(1.2),
                          lss = TRUE),
             data = wdata, trace = TRUE, crit = "log")

coef(fit1, matrix = TRUE)
vcov(fit1)
summary(fit1)

###   A second option (producing same results!!) might be to use the
###   constraints argument in the 'vglm()' call. Note that 'x2' affects
###   the scale parameter only.

fit2 <- vglm(y1 ~ x2,
             invweibull2mr(zero = NULL),
             data = wdata, trace = TRUE,
             constraints = list(x2 = rbind(1, 0)))

coef(fit2, matrix = TRUE)
vcov(fit2)
summary(fit2)
constraints(fit2)
```

`invweibullDist` *The Inverse Weibull Distribution*

Description

Density, distribution function, quantile function and random numbers generator for the Inverse Weibull Distribution.

Usage

```
dinvweibull(x, scale = 1, shape, log = FALSE)
pinvweibull(q, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
qinvweibull(p, scale = 1, shape, lower.tail = TRUE, log.p = FALSE)
rinvweibull(n, scale = 1, shape)
```

Arguments

`x, q, p, n`	Same as `Weibull`.
`scale, shape`	Scale and shape parameters, same as `Weibull`. Both must be positive.
`log, log.p, lower.tail`	Same as `Weibull`.

Details

The Inverse Weibull density with parameters scale = b and shape = s, is

$$f(y) = s b^s y^{-s-1} \exp\left[-(y/b)^{-s}\right],$$

for $y > 0$, $b > 0$, and $s > 0$.

The Weibull distribution and the Inverse Weibull distributions are related as follows:

Let X be a Weibull random variable with paramaters scale =b and shape =s. Then, the random variable $Y = 1/X$ has the Inverse Weibull density with parameters scale = $1/b$ and shape = s. Thus, algorithms of *[dpqr]-Inverse Weibull* underlie on `Weibull`.

Let Y be a r.v. distributed as Inverse Weibull (b, s). The k^{th} moment exists for $-\infty < k < s$ and is given by

$$E[Y^k] = b^k\, \Gamma(1 - k/s).$$

The mean (if $s > 1$) and variance (if $s > 2$) are

$$E[Y] = b\, \Gamma(1 - 1/s); \quad Var[Y] = b^2\, [\Gamma(1 - 2/s) - (\Gamma(1 - 1/s))^2].$$

Here, $\Gamma(\cdot)$ is the gamma function as in `gamma`.

Value

`dinvweibull()` returns the density, `pinvweibull()` computes the distribution function, `qinvweibull()` gives the quantiles, and `rinvweibull()` generates random numbers from the Inverse Weibull distribution.

Warning

The order of the arguments of *[dpqr]-Inverse Weibull* does not match those in `Weibull`.

Note

Small values of *scale* or *shape* will provide Inverse Weibull values too close to zero. Then, function `rinvweibull()` with such characteristics will return either values too close to zero or values represented as zero in computer arithmetic.

The Inverse Weibull distribution, which is that of X where $1/X$ has the Weibull density, is known as the log-Gompertz distribution. Thus, in order to emphazise the continuity concept of the Inverse Weibull density, if $x = 0$, then `dinvweibull` returns zero, which is the limit of such a density when $'x'$ tends to zero.

Author(s)

V. Miranda and T. W. Yee.

References

Kleiber, C. and Kotz, S. (2003) *Statistical Size Distributions in Economics and Actuarial Sciences*. Wiley Series in Probability and Statistics. Hoboken, New Jersey, USA.

Abramowitz, M. and Stegun, I. A. (1972) *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. ch.6, p.255. Dover, New York, USA.

See Also

`Weibull`, `gamma`.

Examples

```
#(1) _______________
n        <- 20
scale    <- exp(2)
shape    <- exp(1)
data.1   <- runif(n, 0, 1)
data.q   <- qinvweibull(-data.1, scale = scale, shape = shape, log.p = TRUE)
data.p   <- -log(pinvweibull(data.q, scale = scale, shape = shape))
arg.max  <- max(abs(data.p - data.1))     # Should be zero


#(2)_________________
 scale  <- exp(1.0)
  shape <- exp(1.2)
  xx    <- seq(0, 10.0, len = 201)
  yy    <- dinvweibull(xx, scale = scale, shape = shape)
  qtl   <- seq(0.1, 0.9, by =0.1)
  d.qtl <- qinvweibull(qtl, scale = scale, shape = shape)
  plot(xx, yy, type = "l", col = "red",
       main = "Red is density, blue is cumulative distribution function",
       sub  = "Brown dashed lines represent the 10th, ... 90th percentiles",
       las = 1, xlab = "x", ylab = "", ylim = c(0,1))
  abline(h = 0, col= "navy", lty = 2)
  lines(xx, pinvweibull(xx, scale = scale, shape = shape), col= "blue")
  lines(d.qtl, dinvweibull(d.qtl, scale = scale, shape = shape),
```

```
type ="h", col = "brown", lty = 3)
```

KPSS.test *KPSS tests for stationarity*

Description

The Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test for the null hypothesis that the series x is `level` or `trend` stationary

Usage

```
KPSS.test(x, type.H0 = c("level", "trend")[1],
             trunc.l = c("short", "large")[1],
             show.output = TRUE)
```

Arguments

`x`	Numeric. A univariate series.
`type.H0`	The null hypothesis to be tested: either `level` or `trend` stationarity.
`trunc.l`	The lag truncation parameter. See below for more details.
`show.output`	Logical. Should the results be displyed? Default is `TRUE`.

Details

To test the null hypothesis that a univariate time series is level–stationary or stationary around a deterministic trend. The alternative states the existence of a unit root.

Under this methodology, the series, say $\{y_t;\ t = 1, \ldots, T\}$ is assumed to be decomposed as

$$y_t = \rho t + \xi_t + \varepsilon_t,$$

that is, as the sum of a deterministic trend, a random walk (ξ_t), and a stationary error ($\varepsilon_t \sim N(0, \sigma_z^2)$. Hence, this test reduces to simply test the hypothesis that $\{\xi_t\}$ is stationary, that is, $H_0 : \sigma_z^2 = 0$.

The test statistic combines the one–sided Lagrange multiplier (LM) statistic and the *locally best invariant* (LBI) test statistic (Nabeya and Tanaka, (1988)). Its asymptotic distribution is discussed in Kwiatkowski et al. (1992), and depends on the 'long–run' variance σ^2. The test statistic is given by

$$\eta = T^{-2} \sum_i S_i^2 / \widehat{\sigma}^2 = T^{-2} \sum_i S_i^2 / s^2(l).$$

where $s^2(l)$ is a consistent estimate of σ^2, given by

$$s^2(l) = (1/T) \sum_{t=1}^{T} \varepsilon_t^2 + (2/T) \sum_{s=1}^{l} w(s, l) \sum_{t=s+1}^{T} \varepsilon_t \varepsilon_{t-s}.$$

Here, $w(s,l) = 1 - s/(l+1)$, where `l` is taken from `trunc.l`, the lag–truncation parameter. The choice `"short"` gives the smallest integer not less than $3\sqrt{T}/11$, or else, $9\sqrt{T}/11$, if `trunc.l = "large"`.

Note, here the errors, ε_t, are estimated from the regression x 1 (level) or x $1 + t$ (trend), depending upon the argument `type.H0`.

Unlike other software using linear interpolates, here the p–values for both, trend and level stationarity, are interpolated by cubic spline interpolations from the tail critical values given in Table 1 in Kwiatkowski et al. (1992). The interpolation takes place on η.

Value

A list with the following:

1) Test statistic and P-value,

2) Critical values,

3) Residuals, ε_t.

Note

There is no standard methodology to select an appropriate value for `trunc.l`, however, satisfactory results have been found for `trunc.l` proportional to $T^{1/2}$. See Andrews, D.W.K. (1991) for a discussion on this. Empirically, this parameter may be *suggested* by the problem in turn, and should be large enough to approximate the true dynamic behaviour of the series.

Author(s)

Victor Miranda.

References

Andrews, D.W.K. (1991) Heteroskedasticiy and autocorrelation consistent covariance matrix estimation. *Econometrica*, **59**, 817–858.

Kwiatkowski, D., Phillips, P.C.B., Schmidt, P., and Shin, Y. (1992) Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, **54**, 159–178.

Nabeya, S. and Tanaka, K. (1988) Asymptotic theory of a test for the constancy regression coefficients against the random walk alternative. *Annals of Statistics*, **16**, 218–235.

Phillips, P.C.B. and Perron, P. (1988) Testing for a unit root in time series regression. *Biometrika*, **75**, 335–346.

Phillips, P.C.B. (1987) Time series with unit roots. *Econometrica*, **55**, 277–301

See Also

`checkTS.VGAMextra`.

Examples

```
set.seed(2802)
test <- KPSS.test(rnorm(20), type.H0 = "trend")
class(test)

test$crit.value
```

`logffMlink` *Link functions for the mean of 1–parameter discrete distributions: The Logarithmic Distribuion.*

Description

Computes the `logffMlink` transformation, including its inverse and the first two derivatives.

Usage

```
logffMlink(theta, bvalue = NULL,
           alg.roots = c("Newton-Raphson", "bisection")[1],
           inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ by default although could be η depending on other parameters. See below for details.
`bvalue`	This is a boundary value. See below. Also refer to `Links` for additional details.
`alg.roots`	Character. The iterative algorithm to find the inverse of this link function. Default is the first (Newton–Raphson). Optionally, the bisection method is also available. See below for more details.
`inverse, deriv, short, tag`	Details at `Links`

Details

This link function arises as a natural link function for the mean, μ, of the logarithmic (or log-series) distribution, `logff`. It is defined for any value of the shape parameter s (i.e. `theta` in the VGLM/VGAM context), $0 < s < 1$, as the logarithm of $\mu = \mu(s)$. It can be easily shown that `logffMlink` is the difference of two common link functions: `logitlink` and `cloglog link`.

It is particularly usefull for event–rate data where the expected number of events can be modelled as

$$\mu = \mu(s) = \lambda t.$$

Here λ is the standardized mean (or event-rate) per unit time, t is the timeframe observed, whereas μ and s are the mean and the shape parameter of the logarithmic distribution respectively. The logarithm is then applied to both sides so that t can be incorporated in the analysis as an offset.

While `logffMlink` is not the canonical link function of the logarithmic distribution, it is certainly part of the canonical link, given by the composite

$$\log \circ (g^{-1}) \circ \log,$$

where g^{-1} denotes the inverse of `logffMlink`.

The domain set of this link function is $(0, 1)$. Therefore, values of `theta` (that is s) too close to 0 or to 1 or out of range will result in `Inf`, `-Inf`, `NA` or `NaN`. Use argument `bvalue` to adequately replace them before computing the link function.

Particularly, if `inverse = TRUE` and `deriv = 0`, then s becomes η, and therefore the domain set turns to $(0, \infty)$.

If `theta` is a character, then arguments `inverse` and `deriv` are disregarded.

Value

For `deriv = 0`, the `logffMlink` transformation of `theta`, i.e., `logitlink(theta) - cloglog(theta)`, if `inverse = FALSE`.

When `inverse = TRUE` the vector entered at `theta` becomes η and, then, this link function returns a unique vector θ_η such that

$$\texttt{logffMlink}(\theta_\eta) = \eta,$$

i.e., the inverse image of η. Specifically, the inverse of `logffMlink` cannot be written in closed–form, then the latter is equivalent to search for the roots of the function

$$\texttt{logff.func}(\theta) = \texttt{logffMlink}(\theta) - \eta$$

as a function of θ. To do this, the auxiliary function `logff.func` is internally generated. Then, with the method established at `alg.roots`, either Newton–Raphson or bisection, this link function approximates and returns the inverse image θ_η (of given η), which plays the role of the inverse of `logffMlink`. In particular, for $\eta = 0$ and $\eta =$`Inf`, it returns 0 and 1 respectively.

For `deriv = 1`, d `eta` / d `theta` as a function of `theta` if `inverse = FALSE`, else the reciprocal d `theta` / d `eta`.

Similarly, when `deriv = 2` the second order derivatives are correspondingly returned.

Both, first and second derivatives, can be written in closed–form.

Warning

`logffMlink` is a monotonically increasing, convex, and strictly positive function in $(0, 1)$ such that the horizontal axis is an asymptote. Therefore, when the inverse image of η is required, each entry of η (via argument `theta`) must be non-negative so that $\texttt{logff.func}(\theta; \eta) = \texttt{logffMlink}(\theta) - \eta$ is *shifted down*. This fact allows this function to uniquely intersect the horizontal axis which

guarantees to iteratively find the corresponding root θ_η, i.e., the inverse image of η. Else, NaN will be returned.

See example 3. It is the plot of logffMlink in $(0, 1)$ for $\eta = 1.5$.

Besides, the vertical straight line theta$= 1$ is also an asymptote. Hence, this link function may grow sharply for values of theta too close to 1. See Example 4 for further details.

Note

To find the inverse image θ_η of η, either newtonRaphson.basic or bisection.basic is called.

This link function can be used for modelling any parameter lying between 0.0 and 1.0. Consequently, when there are covariates, some problems may occur. For example, the method entered at alg.roots to approximate the inverse image may converge at a slow rate. Similarly if the sample size is small, less than 20 say. Try another link function, as logitlink, in such cases.

Author(s)

V. Miranda and T. W. Yee

See Also

logff, newtonRaphson.basic, bisection.basic, Links, cloglogLink, logitlink.

Examples

```
## Example 1  ##
set.seed(0906)
Shapes <- sort(runif(10))
logffMlink(theta = Shapes, deriv = 1)    ## d eta/d theta, as function of theta

logldata.inv <-
       logffMlink(theta = logffMlink(theta = Shapes), inverse = TRUE) - Shapes

summary(logldata.inv)                  ## Should be zero


## Example 2 Some probability link funtions ##
s.shapes <- ppoints(100)


par(lwd = 2)
plot(s.shapes, logitlink(s.shapes), xlim = c(-0.1, 1.1), type = "l", col = "limegreen",
     ylab = "transformation", las = 1, main = "Some probability link functions")
lines(s.shapes, logffMlink(s.shapes), col = "blue")
lines(s.shapes, probitlink(s.shapes), col = "purple")
lines(s.shapes, cloglogLink(s.shapes), col = "chocolate")
lines(s.shapes, cauchitlink(s.shapes), col = "tan")
abline(v = c(0.5, 1), lty = "dashed")
abline(v = 0, h = 0, lty = "dashed")
legend(0.1, 4.5, c("logffMlink","logitlink", "probitlink", "cloglogLink",
"cauchitlink"),
```

```
       col = c("blue", "limegreen", "purple", "chocolate", "tan"), lwd = 1)
par(lwd = 1)



## Example 3. Plot of 'logffMlink()' with eta = 1.5.  ##
m.eta1.5 <- logffMlink(theta = s.shapes, deriv = 0) - 1.5


 plot(m.eta1.5 ~ s.shapes, type = "l", col = "limegreen",
      las = 1, lty = 2, lwd = 3, xlim = c(-0.1, 1.0), ylim = c(-2, 3),
      xlab = "shape parameter, s, in (0, 1).",
      ylab = "logffMlink(s) - 1.5",
      main = "logff.func(s; 1.5) = logffMlink(s) - 1.5, in (0, 1)")
 abline(h = 0, v = 0)
 abline(v = 1.0, lty = 2)



## Example 4. Special values of theta, inverse = FALSE ##
s.shapes <- c(-Inf, -2, -1, 0.0, 0.25, 0.5, 1, 10, 100, Inf, NaN, NA)
rbind(s.shapes, logffMlink(theta = s.shapes))
```

MAXff *VGLTSMs Family Functions: Order–q Moving Average Model with covariates*

Description

Estimates the intercept, standard deviation (or variance) of the random noise (not necessarily constant), and the conditional–mean model coefficients of an order–q moving average (MA) process with covariates (MAX(q)) by maximum likelihood estimation using Fisher scoring.

Usage

```
MAXff(order    = 1,
      zero     = c(if (nomean) NULL else "Mean", "MAcoeff"),
      xLag     = 0,
      type.EIM = c("exact", "approximate")[1],
      var.arg  = TRUE,
      nomean   = FALSE,
      noChecks = FALSE,
      lmean    = "identitylink",
      lsd      = "loglink",
      lvar     = "loglink",
      lMAcoeff = "identitylink",
      imean    = NULL,
      isd      = NULL,
      ivar     = NULL,
      iMAcoeff = NULL)
```

Arguments

`order`	The order 'q' of the MA model, recycled if needed. By default `q = 1`.
`zero`	Integer or character–string vector. Same as `ARIMAXff`. Details at `zero`.
`xLag`	Same as `ARIMAXff`.
`type.EIM, var.arg`	Same as `ARIMAXff`.
`nomean`	Logical. `nomean = TRUE` supresses estimation of the *mean* (intercept of the conditional–mean model).
`noChecks`	Logical. Same as `ARIMAXff`.
`lmean, lsd, lvar, lMAcoeff`	Link functions applied to the mean (intercept), the standard deviation or variance of the random noise, and the MA coefficients (conditional–mean model). Note, `lmean` plays the role of `ldrift`.
`imean, isd, ivar, iMAcoeff`	Same as `ARIMAXff`. Note, `imean` plays the role of `idrift`.

Details

Similar to `ARXff`, this family function fits an order–q moving average model with covariates (MAX(q)), another special case of the class VGLM–ARIMA (Miranda and Yee, 2018). Observations, Y_t, are modelled as

$$Y_t|\Phi_{t-1} = \mu_t + \phi_1\varepsilon_{t-1} + \ldots + \phi_q\varepsilon_{t-q} + \varepsilon_t,$$

where μ_t is the (possibly time–dependent) intercept, modelled as $\mu_t = \mu + \boldsymbol{\beta}^T\boldsymbol{x}_t$, and the errors are mean–zero Gaussian: $\varepsilon_t|\Phi_{t-1} \sim N(0, \sigma^2_{\varepsilon_t|\Phi_{t-1}})$. The symbol Φ_t denotes the history of the joint process $\left(Y_t, \boldsymbol{X}^T_{t+1}\right)$, at time t for a time–varying covariate vector $\boldsymbol{x}_t$.

At each step of Fisher scoring, the *exact* log-likelihood based on model above is computed through `dMAq`.

The linear predictor by default is

$$\boldsymbol{\eta} = \left(\mu_t, \log\sigma^2_{\varepsilon_t|\Phi_{t-1}}, \phi_1, \ldots, \phi_q\right)^T.$$

The unconditional mean of the process is simply $E(Y_t) = \mu$, provided no covariates added.

This family function is not restricted to the noise to be strictly white noise (in the sense of *constant variance*). That is, covariates may be incorporated in the linear predictor $\log\sigma^2_{\varepsilon_t|\Phi_{t-1}}$. Also, it handles *multiple responses* so that a matrix can be used as the response. For further details on VGLM/VGAM–link functions, such as `logitlink`, refer to `Links`.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Warning

By default, a moving-average model of order-1 is fitted.

If different, the `order` is recycled up to the number of responses entered in the `vglm` \ `vgam` call has been matched.

Successful convergence depends on reasonably setting initial values. If initial values computed by the algorithm are not adequate, make use of the the optional initial values (`imean`, `isd`, etc.)

For constraints on the paramaters see `cm.ARMA`.

Note

Further choices for the random noise, besides Gaussian, will be implemented over time.

`zero` can be either an *integer* vector or a vector of **character strings** specifying either the position(s) or name(s) (partially or not) of the parameter(s) modeled as intercept-only. For `MAXff`, the parameters are placed and named as follows (by convention):

`c("Mean", "noiseVar" or "noiseSD", "MAcoeff")`.

Users can modify the `zero` argument accordingly. For simplicity, the second choice recommended. See `CommonVGAMffArguments` for further details on `zero`.

If **no** constraints are entered in the fitting process, (e.g., via `cm.ARMA`) this family function internally verifies by default whether the estimated series is invertible (since `noChecks = FALSE`). To ignore this step, set `noChecks = TRUE`. If the estimated MA process is non-invertible, MLE coefficients will conform with the corresponding invertible MA model.

Further details about these checks are shown within the `summary()` output.

Author(s)

Victor Miranda and Thomas W. Yee.

References

Miranda, V. and Yee, T.W. (2018) Vector Generalized Linear Time Series Models. *In preparation*.

Madsen, H. (2008) *Time Series Analysis* Florida, USA: *Chapman & Hall*. (Sections 5.3 to 5.5).

Tsay, R. (2013) An Introduction to Analysis of Financial data with R. New Jersey, USA: *Wiley* Sections 2.2 to 2.4.

See Also

`ARIMAXff`, `ARXff`, `checkTS.VGAMextra`, `CommonVGAMffArguments`, `Links`, `vglm`, `vgam`.

Examples

```
set.seed(2)
nn    <- 130
### Coefficients
phi1  <-  0.34; phi2 <- -1.19; phi3 <- 0.26
### Intercept
mu    <- c(-1.4, 2.3)
```

```
### Noise standar deviations (Two responses)
sdMA  <- c(sqrt(6.5), sqrt(4.0))
### A single covariate.
Xcov <- runif(nn)

# Generating two MA processes, TS1 and TS2, Gaussian noise.
# Note, the SD noise for TS2 is function of Xcov.

y1   <- mu[1] + arima.sim(nn,
                          model = list( ma = c(phi1, phi1^2)),
                          rand.gen = rnorm, sd = exp(sdMA[1]) )
y2   <- mu[2] + arima.sim(nn,
                          model = list( ma = c(phi1, phi2, phi3) ),
                          rand.gen = rnorm, sd = exp(Xcov + sdMA[2]) )
# OUR DATA
tsdata <- data.frame(x2 = Xcov , TS1 = y1, TS2 = y2)

#------------------------------------------------------------------------#
# 1. A simple MA(3) to compare with 'arima()'.

myfit0 <- vglm(TS1 ~ 1,
               MAXff(order = 3, type.EIM = "exact",
                    var.arg = FALSE),
               #constraints = cm.ARMA(Model = ~ 1,
               #                      lags.cm = 2,
               #                      Resp = 1),
               data = tsdata, trace = TRUE)

Coef(myfit0)[c(3, 4, 1)]
fitArima <- arima(tsdata$TS1, order = c(0, 0, 2))
coef(fitArima)

AIC(myfit0); BIC(myfit0)

# ------------------------------------------------------------------------#
# 2. Estimate an MA(3), intercept-only, using initial values.

myfit <- vglm(TS2 ~ 1,
              MAXff(order = 3, type.EIM = c("exact", "approx")[1],
                   # Optional initial values.
                    imean = 3,
                    iMAcoeff = c(0.3, -0.2, 0.25),
                   var.arg = TRUE),
              data = tsdata, trace = TRUE)

Coef(myfit)
summary(myfit)
constraints(myfit)


#---------------------------------------#
# Same model fitted using arima()
#---------------------------------------#
```

```
fitArima <- arima(tsdata$TS2, order = c(0, 0, 3))
coef(fitArima)


#------------------------------------------------------------------------#
# 3. An MAX(3) with one covariate, testing its effect over the
#    standard deviation of the Gaussian noise. Note the 'zero' argument.

myfit1 <- vglm(TS2 ~ x2,
               # Or Multiple responses!
               # cbind(TS1, TS2) ~ 1,
               MAXff(order = 3, type.EIM = "exact", xLag = 1,
                    # Optional initial values:
                    # idev.mean = 1.4,
                    # iMAcoeff = c(2.3, -1.2, 0.25), isd = 1.6,

                    # NOTE THE ZERO ARGUMENT:
                    zero = c("Mean", "MAcoeff"),

                    var.arg = TRUE),
               data = tsdata, trace = TRUE)

coef(myfit1, matrix = TRUE)
summary(myfit1)
vcov(myfit1)

constraints(myfit1)

#------------------------------------------------------------------------#
# Model above CANNOT be fitted using arima()
#------------------------------------------------------------------------#
```

`maxwellQlink` *Link functions for the quantiles of several 1–parameter continuous distributions*

Description

Computes the `maxwellQlink` transformation, its inverse and the first two derivatives.

Usage

```
maxwellQlink(theta, p = stop("Argument 'p' must be specified."),
             bvalue = NULL, inverse = FALSE,
             deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. See below for further details.
p	Numeric. A single value between 0 and 1. It is the p–quantile to be modeled by this link function.
bvalue, inverse, deriv, short, tag	See Links.

Details

In the VGLM/VGAM quantile regression context, this link function can be used to model any p–quantile of the Maxwell distribition. It is the maxwellQlink transformation given by

$$\sqrt{2\,\texttt{qgamma(p, 1.5) / a.}}$$

Here, a is positive parameter as in maxwell whereas qgamma is the quantile function of the gamma distribution.

Numerical values of a or p out of range will result in Inf, -Inf, NA or NaN correspondingly.

In particular, arguments inverse and deriv are disregarded if theta is character. Also, if inverse = TRUE and deriv = 0, then argument theta becomes η. See Links for further details about this.

Value

For deriv = 0, the maxwellQlink transformation of theta, when inverse = FALSE. If inverse = TRUE, then the inverse given by 2*qgamma(p, 1.5) / theta^2 is returned.

For deriv = 1, this function returns the derivative d eta / d theta, if inverse = FALSE. Else, the reciprocal d theta / d eta as a function of theta.

If deriv = 2, then the second order derivatives in terms of theta are accordingly returned.

Note

Numerical instability may occur for values theta too close to zero. Use argument bvalue to replace them before computing the link.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

maxwell, Links.

Examples

```
## E1. maxwellQlink() and its inverse ##
 p <- 0.25        ## Modeling the first quartile
 my.a <- seq(0, 5, by = 0.1)[-1]
 max(my.a - maxwellQlink(maxwellQlink(my.a, p = p), p = p, inverse  =TRUE)) ## Zero
```

```
## E2. The first two values are negative, NaN is returned ##
  maxwellQlink(theta = c(-0.15, -0.10, 0.25, 0.35) , p = p, inverse  = FALSE)
  maxwellQlink(theta = c(-0.15, -0.10, 0.25, 0.35) , p = p, inverse  = TRUE)

## E3. Plot of the maxwellQlink() and its inverse                    ##
## Note, inverse = TRUE implies that argument 'theta' becomes 'eta'. ##


 #--- THE LINK

 plot(maxwellQlink(theta = my.a, p = p) ~ my.a,
      type = "l", col = "blue", lty = "dotted", lwd = 3,
      xlim = c(-0.1, 10), ylim = c(-0.1, 5), las = 1,
      main = c("Blue is maxwellQlink(), green is the inverse"),
      ylab = "eta = maxwellQlink", xlab = "theta")
 abline(h = 0, v = 0, lwd = 2)

 #--- THE INVERSE
 lines(my.a, maxwellQlink(theta = my.a, p = p,inv = TRUE),
        col = "green", lwd = 2, lty = "dashed")
 lines(my.a, my.a)     # Tracing the identity function for double--check
```

MVNcov *Multivariate Normal Distribution Family Function*

Description

Maximum likelihood estimation of the Multivariate Normal distribution. The covariances (not correlation coefficients) are included in the parameter vector.

Usage

```
MVNcov(zero = c("var", "cov"),
       lmean = "identitylink",
       lvar  = "loglink",
       lcov  = "identitylink")
```

Arguments

`zero` Integer or character–string vector. Which linear predictors are intercept–only. Details at `zero` or `CommonVGAMffArguments`.

`lmean, lvar, lcov` VGLM–link functions applied to the means, variances and covariances.

Details

For the K–dimensional normal distribution, this fits a linear model to the K means μ_j $j = 1, \ldots, K$, which are the first entries in the linear predictor. The variances σ_j^2 $j = 1, \ldots, K$ and then the covariances cov_{ij} $i = 1, \ldots, K, j = i + 1, \ldots, K$, are next aligned. The fitted means are returned as the fitted values.

The log–likelihood is computed via `dmultinorm`, an implementation of the multivariate Normal density.

The score and expected information matrices are internally computed at each Fisher scoring step, using its vectorized form.

The response should be an K–column matrix. The covariances may be any real number so that the `identitylink` is a reasonable choice. For further details on VGLM/VGAM–link functions, see `Links`.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Note

Unlike other implementations, e.g., `binormal` from **VGAM** in terms of ρ and standard deviations, `MVNcov` estimates the variances and covariances, modeled as intercept–only. See argument `zero`, whose default is `c("var", "cov")`, to change this.

Thus far, there is no guarantee that the estimated var–cov matrix will be positive–definite. Proper procedures to validate this will be incorporated shortly, such as the `@validparams` slot.

Although the function has been tested on $K \leq 5$ data sets, it is recommended that $K \leq 3$, unless the data are *nice* and n is sufficiently large.

Author(s)

Victor Miranda.

See Also

`dmultinorm`, `binormal`, `CommonVGAMffArguments`, `vglm`.

Examples

```
# K = 3.
set.seed(180227)
nn  <- 85
mvndata <- data.frame(x2 = runif(nn), x3 = runif(nn))
mvndata <- transform(mvndata,
                     y = rbinorm(nn, mean1 = 2 - 2 * x2 + 1 * x3,
                          mean2 = 2 - 1.5 * x3,
                          var1 = exp(1.0), var2 = exp(-0.75),
                          cov12 = 0.5 * exp(1.0) * exp(-0.75)))
mvndata <- transform(mvndata, y3 = rnorm(nn, mean = 2 + x2, sd = exp(1.5)))
```

```
colnames(mvndata) <- c("x2", "x3", "y1", "y2", "y3")

mvnfit <- vglm(cbind(y1, y2, y3) ~ x2 + x3, MVNcov, data = mvndata, trace = TRUE)
(mvncoef  <- coef(mvnfit, mat = TRUE))

## Check variances and covariances: var1, var2 and var3.
exp(mvncoef[c(10, 13, 16)])  # True are var1 = exp(1.0) = 2.718,
                             # var2 = exp(-0.75) = 0.472
                             # and var3 = exp(1.5)^2 = 20.08554
vcov(mvnfit)
constraints(mvnfit)
summary(mvnfit)
```

`newtonRaphson.basic` *Newton–Raphson algorithm*

Description

Newton–Raphson algorithm to approximate the roots of univariate real–valued functions.

This function is vectorized.

Usage

```
newtonRaphson.basic(f, fprime, a, b,
                    tol = 1e-8, n.Seq = 20,
                    nmax = 15, ...)
```

Arguments

`f`	A univariate function whose root(s) are approximated. This is the target function. Must return a vector.
`fprime`	A function. The first derivative of `f`. Must return a vector.
`a, b`	Numeric vectors. Upper and lower real limits of the open interval (a, b) where the root(s) of `f` will be searched. Notice, entries `Inf`, `-Inf`, `NA` and `NaN` are not handled. These vectors are subject to be recycled if `a` and `b` lenghts differ.
`tol`	Numeric. A number close to zero to test whether the approximate roots from iterations k and $(k + 1)$ are close enough to stop the algorithm.
`n.Seq`	Numeric. The number of equally spaced initial points within the interval (`a`, `b`) to internally set up initial values for the algorithm.
`nmax`	Maximum number of iterations. Default is 15.
`...`	Any other argument passed down to functions `f` and `fprime`.

Details

This is an implementation of the well–known Newton–Raphson algorithm to find a real root, r, $a < r < b$, of the function f.

Initial values, r_0 say, for the algorithm are internally computed by drawing '`n.Seq`' equally spaced points in (a, b). Then, the function `f` is evaluated at this sequence. Finally, r_0 results from the closest image to the horizontal axis.

At iteration k, the $(k+1)^{th}$ approximation given by

$$r^{(k+1)} = r^{(k)} - \texttt{f}(r^{(k),\ldots})/\texttt{fprime}(r^{(k)}, \ldots)$$

is computed, unless the approximate root from step k is the desired one.

`newtonRaphson.basic` approximates this root up to a relative error less than `tol`. That is, at each iteration, the relative error between the estimated roots from iterations k and $k+1$ is calculated and then compared to `tol`. The algorithm stops when this condition is met.

Instead of being single real values, arguments `a` and `b` can be entered as vectors of length n, say $\texttt{a} = c(a_1, a_2, \ldots, a_n)$ and $\texttt{b} = c(b_1, b_2, \ldots, b_n)$. In such cases, this function approaches the (supposed) root(s) at each interval (a_j, b_j), $j = 1, \ldots, n$. Here, initial values are searched for each interval (a_j, b_j).

Value

The approximate roots in the intervals (a_j, b_j). When $j = 1$, then a single estimated root is returned, if any.

Note

The explicit forms of the target function `f` and its first derivative `fprime` must be available for the algorithm.

`newtonRaphson.basic` does not handle yet numerically approximated derivatives.

A warning is displayed if no roots are found, or if more than one root might be lying in (a_j, b_j), for any $j = 1, \ldots, n$.

If `a` and `b` lengths differ, then the recyling rule is applied. Specifically, the vector with minimum length will be extended up to match the maximum length by repeating its values.

Author(s)

V. Miranda.

See Also

`bisection.basic`

Examples

```
# Find the roots in c(-0.5, 0.8), c(0.6, 1.2) and c(1.3, 4.1) for the
# f(x) = x * (x - 1) * (x - 2). Roots: r1 = 0, and r2 = 1, r3 = 2.

f <- function(x) x * (x - 1) * (x - 2)
fprime <- function(x) 3 * x^2 - 6 * x + 2

# Three roots.
newtonRaphson.basic(f = f, fprime  = fprime,
                    a = c(-0.5, 0.6, 1.3),
                    b = c(0.8, 1.2, 4.1))             ## 0.0, 1.0 and 2.0

# Recycling rule. Intervals analysed are (-0.5, 1.2) and (0.6, 1.2)
newtonRaphson.basic(f = f, fprime  = fprime,
                    a = c(-0.5, 0.6), b = c(1.2))

## Warning: There is more than one root in (-0.5, 1.2)!
```

`normal1sdff` *Estimation and Inference for Conditional Quantiles of a 1–parameter Univariate Normal Distribution.*

Description

Maximum likelihood estimation of the standard deviation, including inference for conditional quantiles, of a univariate normal distribution.

Usage

```
normal1sdff(zero = NULL, link = "loglink",
            fixed.mean = 0, p.quant = NULL,
            var.arg = FALSE)
```

Arguments

`zero`	Allows to model the single linear predictor in this family function as intercept–only. See below for important details about this.
`link`	This is the link function applied to the standard deviation. If `var.arg` is `TRUE`, then `link` is applied to the variance. The default is `loglink`. For inference on conditional quantiles entered at `p.quant`, however, it must be manually changed to `normal1sdQlink`. See below for further details.
`fixed.mean`	Numeric, a vector or a matrix. It allocates the (fixed) mean of the response in the fitting process. See below for further details.
`p.quant`	Numeric. A prototype vector of probabilities indicating the quantiles of interest, when quantile regression is to be performed.
`var.arg`	If `TRUE`, then the variance is estimated, else the standard deviation is used.

Details

This family function is a variant of uninormal to estimate the standard deviation of a Normal distribution with *known* mean. The estimated values are returned as the fitted values, unlike some other family functions where the mean is returned as *fitted values*. However, here the mean is assumed to be known.

By default, the response is supposedly *centered* on its mean, that is, fixed.mean$= 0$. Change this accordingly: For a single response or multiple responses, fixed.mean must be a numeric vector where each entry is the mean of each response, only *if* the mean is *fixed*. When the mean is not constant, fixed.mean must be matrix with the number of columns matching the number of responses.

Quantile regression: The (single) linear/additive predictor by default is the log of the standard deviation. However, if quantile regression is of primary interest, then the response must be entered using the function Q.reg, and the corresponding p–quantiles through p.quant in the vglm or vgam call. Additionally, set normalsdQlink as the link function via the argument link.

This family **VGAM** function handles multiple responses.

Value

An object of class "vglmff". See vglmff-class for further details.

Warning

Be aware of the argument zero: by default, the single linear/additive predictor in this family function, say η, can be modeled in terms of covariates, i.e., zero = NULL. To model η as intercept–only, set zero = "sd".

See zero for more details about this.

Author(s)

V. Miranda.

See Also

normal1sdQlink, loglink, uninormal, CommonVGAMffArguments, zero, vgam, vglm.

Examples

```
set.seed(121216)
my.mean <- -1      #  Mean (CONSTANT)
my.sd   <- 2.5
y <- rnorm(100, mean = my.mean, sd = 2.0)      # Generate some data.
normdat <- data.frame(y = y)                   # Setting up our data.


# Plotting the data
  plot(y, main = c("Y ~ Normal ( mean(known), sd = 2.5 ). "),
       ylab = "The data", pch = 20,
       xlim = c(0, 100), ylim = c(-7, 7), col = "blue")
  abline(h = 0, v = 0, lwd = 2, col = "black")
```

```
### EXAMPLE 1. Estimate the SD with two responses. The mean is fixed. ###

fit1 <- vglm(cbind(y, y) ~ 1, family = normal1sdff(fixed.mean = my.mean),
            data = normdat, trace = TRUE, crit = "coef")
Coef(fit1)
summary(fit1)


### EXAMPLE 2. Quantile regression. The link normal1sdQlink() is used. ###

my.p <- c(25, 50, 75) / 100  # Quantiles 25%, 50% and 75% are of interest.

fit2 <- vglm(Q.reg(y, length.arg = 3) ~ 1,
             family = normal1sdff(fixed.mean = my.mean, p.quant = my.p,
                                  link = normal1sdQlink),
             data = normdat, trace = TRUE, crit = "coef")
 summary(fit2)
 head(predict(fit2))
 constraints(fit2)


### EXAMPLE 3. Complete the plot. Quantiles matching. ###


( my.c3Q <- coef(fit2, matrix = TRUE) )
with(normdat, lines(rep(my.c3Q[1], 100), col = "tan"   , lty = "dotted", lwd = 2))
with(normdat, lines(rep(my.c3Q[2], 100), col = "orange", lty = "dotted", lwd = 2))
with(normdat, lines(rep(my.c3Q[3], 100), col = "brown1", lty = "dotted", lwd = 2))
legend(20, 7.0, c("Percentil 75", "Percentil 50", "Percentil 25"),
       col = c("brown1", "orange", "tan"),
       lty = rep("dotted", 3), lwd = rep(2, 3), cex = 0.75)
```

`normal1sdQlink` *Link functions for the quantiles of several 1–parameter continuous distributions.*

Description

Computes the `normal1sdQlink` transformation for the Univariate Normal Distribution, its inverse and the first two derivatives.

Usage

```
normal1sdQlink(theta,  mean = stop("Please, enter the fixed 'mean'."),
               p = stop(" Please, enter argument 'p'."),
               bvalue = NULL, inverse = FALSE, deriv = 0,
               short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ by default although it could be η depending upon other arguments. See below for further details.
`mean`	A numeric vector or a matrix. It is the (known) fixed mean of the Normal distribution of interest. See below for further details.
`p`	Numeric vector of p–quantiles to be modelled by this link function.
`bvalue, inverse, deriv, short, tag`	Details at `Links`.

Details

This link function is necessarily required by `normal1sd` if quantile regression is to be performed. It computes the `normal1sdQlink` transformation, defined as

$$\texttt{mean} + \sqrt{2}\sigma \cdot \texttt{erf}^{-1}(2\texttt{p} - 1).$$

Here, $\texttt{erf}^{-1}$ denotes the inverse of the error function `erf`, and σ is the standard deviation (`theta`) as in `normal1sd`. Technically, `normal1sdQlink` can be used for quantile regression over any vector of p–quantiles of Normally distributed data with *known* mean.

See `normal1sd` for further details about the latter.

Values of p out of the open interval $(0, 1)$ or non–positive values of `theta` will result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

When `deriv = 0`, the `normal1sdQlink` transformation of `theta`, if `inverse = FALSE`. Conversely, if `inverse = TRUE`, `theta` becomes η and the inverse transformation given by `(theta - mean)`$/\sqrt{2}$ $\texttt{erf}^{-1}(2\texttt{p}-1)$ is returned.

For `deriv = 1`, d `eta` / d `theta` if `inverse = FALSE`. Else, this function returns d `theta` / d `eta` as a function of `theta`.

For `deriv = 2`, the second order derivatives are accordingly returned.

Warning

If `p` is a vector, then the recycling rule applies *only* if `theta` is entered as a matrix. Else, only the first entry in `p` is considered.

Note

When `inverse = TRUE`, the reciprocal of the error function, `erf`, evaluated at $2\texttt{p}-1$ is required. However, the result is `Inf` for $\texttt{p} = 0.5$. Here, in consequence, the limit of `erf` when `p` tends to 0.5 is returned to avoid numerical issues.

Author(s)

V. Miranda

See Also

normal1sd, erf, Links.

Examples

```
###   Example 1  ###
theta <- seq(0, 3, by = 0.1)[-1]  # This is sigma, then must be positive.
mean  <- -2.5                     # Intentionally, a negative value for mu.
p     <- 0.25                     # Modelling the first quartile.

eta <- normal1sdQlink(theta = theta, p = p, mean = mean)
inv.eta <- normal1sdQlink(theta = eta, p = p, mean = mean, inverse = TRUE)
summary(inv.eta - theta)          ## Should be 0


###  Example 2. Special values of theta, using argument 'bvalue'.  ###

theta <- c(-Inf, -5, -2.5, 0, 2.5, 5, Inf, NA, NaN)
my.matrix <- rbind(theta, normal1sdQlink(theta = theta, p = p, mean = mean),
                   normal1sdQlink(theta = theta, p = p, mean = mean, bvalue = 1e-5))
rownames(my.matrix) <- c("theta", "No 'bvalue'", "With 'bvalue'")
colnames(my.matrix) <- rep("", 9)

my.matrix   # Second row has NAs, whilst third row has NO NAs except for theta = NA
```

---

notDocumentedYetVGAMextra

Not-documented functions and classes in **VGAMextra**

Description

Those functions not documented yet in **VGAMextra** are aliased to this file.

Details

These functions are still under review or being tested, and will be documented over time.

Value

Overall, these functions returns objects required by time series family functions in **VGAMextra**.

Further details will be given shortly.

Author(s)

V. Miranda

posPoiMlink *Link functions for the mean of 1–parameter discrete distributions: The Positive Poisson Distribution.*

Description

Computes the posPoiMlink transformation, its inverse and the first two derivatives.

Usage

```
posPoiMlink(theta, bvalue = NULL,
            alg.roots = c("Newton-Raphson", "bisection")[1],
            inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. This is θ by default, although it becomes η sometimes, depending on the other parameters. See below for further details.
bvalue	Details at Links.
alg.roots	Character. The iterative method to find the inverse of this link function. Same as zetaffMlink.
inverse, deriv, short, tag	Details at Links

Details

This is a link function for the mean of the positive Poisson distribution. It is defined as

$$\eta = \texttt{posPoiMlink}(\lambda) = -\log(\lambda^{-1} - \lambda^{-1}\texttt{e}^{-\lambda}),$$

where $\lambda > 0$ stands for the single parameter of pospoisson, i.e. theta in the VGLM/VGAM context.

Notice, the mean of the positive Poisson is given by

$$\frac{\lambda}{1 - e^{-\lambda}}.$$

This link function comes up by taking the logarithm on both sides of this equation.

The domain set for λ is $(0, \infty)$. Hence, non–positive values of λ will result in NaN or NA. Use argument bvalue to properly replace them before computing the link function.

posPoiMlink tends to infinity as λ increases. Specially, its inverse grows at a higher rate. Therefore, large values of λ will result in Inf accordingly. See example 2 below.

If theta is a character, arguments inverse and deriv are disregarded.

Value

For `deriv = 0`, the `posPoiMlink` transformation of `theta`, if `inverse = FALSE`. When `inverse = TRUE`, `theta` becomes η and the inverse of `posPoiMlink` is needed but cannot be written in closed–form. Instead this link function returns the approximated inverse image of η, say θ_η, such that

$$\texttt{posPoiMlink}(\theta_\eta) = \eta.$$

Here, θ_η is iteratively computed as the unique root of the auxiliary function

$$f(\theta; \eta) = \texttt{posPoiMlink}(\theta) - \eta,$$

as a function of θ. This work is performed via Newton–Raphson or bisection, as per argument `alg.roots`.

For `deriv = 1`, d `eta` / d `theta` as a function of `theta` if `inverse = FALSE`, else the reciprocal d `theta` / d `eta`.

Similarly, when `deriv = 2` the second order derivatives are returned in terms of `theta`.

Warning

This link function is monotonic increasing in $(0, \infty)$ so that the horizontal axis is an asymptote. Then, in order to assure the root of the auxiliary

$$f(\theta; \eta) = \texttt{posPoiMlink}(\theta) - \eta$$

to be real, η must be positive. As a result, `posPoiMlink` is *shited–down* and hence intersecting the horizontal axis uniquely.

Note

This link function is useful to model any parameter in $(0, \infty)$. Some numerical issues may arise if there are covariates causing negative values the parameter. Try `identitylink` alternatively.

Author(s)

V. Miranda and T. W. Yee.

See Also

`pospoisson`, `newtonRaphson.basic`, `bisection.basic`, `Links`, `identitylink`.

Examples

```
## Example 1. Special values for theta (or eta, accordingly)  ##
m.lambda <- c(0, 0.5, 1, 10, 20, 25, 1e2, 1e3, Inf, -Inf, NaN, NA)

# The 'posPoiMlink' transformation and the first two derivatives.
print(rbind(m.lambda,
```

```
  deriv1 = posPoiMlink(theta = m.lambda, inverse = FALSE, deriv = 1),
  deriv2 = posPoiMlink(theta = m.lambda, inverse = FALSE, deriv = 2)),
  digits = 2)

# The inverse of 'posPoiMlink' and the first two derivatives.
print(rbind(m.lambda,
  Invderiv1 = posPoiMlink(theta = m.lambda, inverse = TRUE, deriv = 1),
  Invderiv2 = posPoiMlink(theta = m.lambda, inverse = TRUE, deriv = 2)),
  digits = 2)


## Example 2. The inverse of 'posPoiMlink' ##
m.lambda <- c(0, 1, 5, 10, 1e2, 1e3)
posPoiMlink(theta = posPoiMlink(m.lambda, inverse = TRUE))
pospoi.inv <- posPoiMlink(posPoiMlink(m.lambda, inverse = TRUE)) - m.lambda

summary(pospoi.inv)                     ## Should be zero.


## Example 3. Plot of 'posPoiMlink' and its first two derivatives ##
## inverse = FALSE, deriv = 0, 1, 2. ##


m.lambda <- seq(0, 35, by = 0.01)[-1]
y.lambda <- posPoiMlink(theta = m.lambda, deriv = 0)
der.1 <- posPoiMlink(theta = m.lambda, deriv = 1)
der.2 <- posPoiMlink(theta = m.lambda, deriv = 2)

plot(y.lambda ~ m.lambda, col = "black",
     main = "log(mu), mu = E[Y], Y ~ pospoisson(lambda).",
     ylim = c(-1, 10), xlim = c(-1, 26),
     lty = 1, type = "l", lwd = 3)
abline(v = 0, h = 0, col = "gray50", lty = "dashed")

lines(m.lambda, der.1, col = "blue", lty = 5, lwd = 3)
lines(m.lambda, der.2, col = "chocolate", lty = 4, lwd = 3)
legend(5, 9, legend = c("posPoiMlink", "deriv = 1", "deriv = 2"),
       col = c("black", "blue", "chocolate"), lty = c(1, 5, 4), lwd = c(3, 3, 3))
```

`Q.reg` *Conditional quantile regression with* **VGAM**

Description

Use this function to adequately confer the `formula` in **VGAM** when fitting quantile regression models.

Usage

```
Q.reg(y, pvector = NULL, length.arg = NULL)
```

Arguments

y	Numeric, a vector or a matrix. It is the response or dependent variable in the formula of the model to be fit, as in vglm or vgam. See below for further details.
pvector	A prototype vector. Entries are the conditional p–quantiles in the fitting process.
length.arg	A length–1 positive integer. It is the number of p–quantiles to be modelled.

Details

Conditional quantile regression can be carried out using family functions in **VGAM** and **VGAMextra**. The formula must be set up using this function, Q.reg. Here, the p–quantiles of interest may be entered via pvector. Alternatively, use argument length.arg by establishing the length of pvector.

Besides, the corresponding link must be entered. For example, gamma1Qlink is the proper link to fit models of conditional quantiles for data distributed as Gamma via the family function gamma1.

See examples for further details.

Value

A matrix, each column adequately arranged for regression on conditional quantiles, conforming with **VGAM**.

Indeed, this is equivalent to cbind(y, y, ...), where the total number of columns is, either the length of pvector, or length.arg.

Note

Link functions for quantile regression in **VGAM** require the vector of p–quantiles of interest via the argument p. See normal1sdQlink or maxwellQlink for instance.

Therefore, the integer entered at length.arg in this function, if utilized, must match the length of the vector p. Else, it will be recycled.

Author(s)

V. Miranda and T. W. Yee.

See Also

normal1sdQlink, maxwellQlink, gamma1Qlink, gamma1, vglm, vgam

Examples

```
###  Quantile regression with data distributed as Maxwell(s)  ###
 set.seed(12073)
 x2 <- seq(0, 100,length.out = 100)       # independent variable
 b0 <- 0.5                                # true intercept
 b1 <- 0.25                               # true slope
 b2 <- 0.02                               # true second order coef.
 alpha <- b0 + b1 * x2 + b2 * x2^2        # Quadratically modelling the parameters
```

```
nn <- 100                                  # Sample size

# The data as a data frame. #
mdata <- data.frame(y = rmaxwell(n = nn, rate = alpha), x2 = x2, x3 = x2^2)

# Quantile regression using our link function maxwellQlink(). #
# Quantiles 25%, 50%, 75% are of interest #
my.p <- c(0.25, 0.50, 0.75)

fit <- vglm(Q.reg(y, pvector = my.p) ~ x2 + x3,

# OPTIONALLY Q.reg(y, length = length(my.p)) ~ x2 + x3

            maxwell(link = maxwellQlink(p = my.p)),
            data = mdata,  trace = TRUE, crit = "coef")

coef(fit, matrix = TRUE)
summary(fit)
head(predict(fit))
constraints(fit)
```

`rayleighMlink` *Link functions for the mean of 1–parameter continuous distributions: The Rayleigh and the Maxwell distributions.*

Description

The `rayleighMlink` and the `maxwellMlink` transformations, their inverse and the first two derivatives.

Usage

```
rayleighMlink(theta, bvalue = NULL, inverse = FALSE,
              deriv = 0, short = TRUE, tag = FALSE)

maxwellMlink(theta, bvalue = NULL, inverse = FALSE,
             deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. It is θ by default, but it may be η depending upon other parameters. See `Links` for further details.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

rayleighMlink and maxwellMlink are link functions to model the mean of the Rayleigh distirbution, (rayleigh), and the mean of the Maxwell distribution, (maxwell), respectively.

Both links are somehow defined as the $\log$ theta plus an *offset*. Specifcally,

$$\texttt{rayleighMlink}(b) = \log(b * \gamma(0.5)/sqrt2),$$

where $b > 0$ is a scale parameter as in rayleigh; and

$$\texttt{maxwellhMlink}(b) = \log(a^{-1/2} * sqrt8/\pi).$$

Here, a is positive as in maxwell.

Non–positive values of a and/or b will result in NaN, whereas values too close to zero will return Inf or -Inf.

Value

For deriv = 0, the corresponding transformation of theta when inverse = FALSE. If inverse = TRUE, then theta becomes η, and the inverse transformations

I) exp(theta) * sqrt(2) / gamma(0.5) for rayleighMlink, and

II) 8 * exp(-2 * theta) / gamma(0.5)^2 for maxwellMlink,

are returned.

For deriv = 1, d eta / d theta when inverse = FALSE. If inverse = TRUE, then d theta / d eta as a function of theta.

When deriv = 2, the second derivatives in terms of theta are returned.

Note

Values of a or b out of range, e.g. when covariates involved, may cause numerical instability. Use argument bvalue to replace them before computing any link.

If theta is character, then arguments inverse and deriv are ignored. See Links for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

maxwell, rayleigh Links.

Examples

```
##  The link and its inverse ##
   theta <- 0.1 + 1:10
   eta   <- maxwellMlink(maxwellMlink(theta = theta), inverse  =TRUE)
   summary(eta - theta)     # Zero

   eta   <- rayleighMlink(rayleighMlink(theta = theta), inverse  =TRUE)
```

```
summary(eta - theta)     # Zero


## Modelling the mean of the Maxwell distribution  ##
set.seed(17010401)

rate <- maxwellMlink(theta = 2, inverse = TRUE)   # ~ 0.046
mdata <- data.frame(y = rmaxwell(1000, rate = rate ))

fit <- vglm(y ~ 1, maxwell(link = "maxwellMlink"),
            data = mdata, trace = TRUE, crit = "coef")

coef(fit, matrix = TRUE)
Coef(fit)
```

`rayleighQlink` *Link functions for the quantiles of several 1–parameter continuous distributions*

Description

Computes the `rayleighQlink` transformation, its inverse and the first two derivatives.

Usage

```
rayleighQlink(theta, p = stop("Argument 'p' must be specified."),
              bvalue = NULL, inverse = FALSE,
              deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. It is θ by default, although it may be η. See `Links` for additional details about this.
`p`	Numeric. A single value between 0.0 and 1.0. It is the p–quantile to be modeled by this link function.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

This link function directly models any p–quantile of the Rayleigh distribution specified by the argument `p`. It is called the `rayleighQlink` transformation defined as

$$b\sqrt{-2\log(1-p)},$$

where $b > 0$ is a scale parameter as in rayleigh.

Numerical values of b or p out of range may result in Inf, -Inf, NA or NaN.

If theta is character, then arguments inverse and deriv are discarded.

Value

For deriv = 0, the rayleighQlink transformation of theta, when inverse = FALSE. If inverse = TRUE, then this function returns theta / sqrt(-2 log(1 - p)).

For deriv = 1, then the function returns d eta / d theta, if inverse = FALSE. If inverse = TRUE, then d theta / d eta as a function of theta.

If deriv = 2, then the second order derivatives in terms of theta.

Note

Numerical instability may occur for values theta too close to zero. Use argument bvalue to replace them before computing the link.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

rayleigh, Links.

Examples

```
## E1. rayleighQlink() and its inverse ##
 p <- 0.50                           ## Modeling the median
 my.b <- seq(0, 5, by = 0.1)[-1]
 max(my.b - rayleighQlink(rayleighQlink(my.b, p = p), p = p, inverse  =TRUE)) ## Zero


## E2. Special values ##
 rayleighQlink(theta = c(Inf, -Inf, NA, NaN), p = p)


## E3. Use of argument 'bvalue' ##
 rayleighQlink(theta = seq(-0.2, 1.0, by = 0.1), p = p)  # WARNING: NaNs if theta <= 0
 rayleighQlink(theta = seq(-0.2, 1.0, by = 0.1), p = p, bvalue = .Machine$double.xmin)
```

summaryS4VGAMextra-methods
Summary methods for Vector Generalized Time Series Models

Description

S4 summary methods for models fitted with time series family functions from **VGAMextra**.

These function are all `methods` for objects of class `vglm` with signature `vgltsmff-class`.

Details

Implementation of vector generalized time series (TS) family functions (*vgltsmff*) in **VGAMextra** is entirely based on the structure of family functions of class `vglmff-class` from **VGAM**. More precisely, *vgltsmff* family functions can be created by calls of the form `new("vgltsmff",...)`, following the structure `vglmff-class`. See `vglmff-class` for additional details.

In this line, specific S4 dispatching methods are currently implemented at **VGAM** to show (or plot) essential statistical information about the model fitted.

For the generic `summary`, specifically, S4 methods for objects with signature `vgtsff` are incorporated in **VGAMextra** to display supplementary analyses commonly required by TS practicioners. That is, additional information to the default output shown by `summaryvglm` for family functions at **VGAM**, as follows:

a) The standard errors, which are computed from the asymptotic distribution of the MLE estimates, unlike the asymptotic approach (z-value) from **VGAM**.

b) Checks on stationarity and/or invertibility for autoregressive (AR), moving average (MA), and autoregressive moving-average (ARMA) models via the polynomial roots.

c) The AIC, AICC and BIC criteria for model identification.

Notice that, for intercept-only models in the 'vglm' context, the asypmtotic distribution of the estimates, either conditional or unconditional, will coincide with the theoretical distributions as long as n increases. In particular, for the AR(p) process, the MLEs and the Yule-Walker estimates will concur asymptotically.

Where covariates or parameter constraints are involved, the standard errors for the estimates from time series family functions at **VGAMextra** are calculated from the predicted values allocated in the slot `@predictors`, when `summary(...)` is called.

In this case, the *conditional* mean, $\mathrm{E}[\eta_j|\mathbf{x}]$ $\mathbf{x}$], is considered as the estimate, where:

$$\eta_j = \sum_{k=1}^{p} \beta_{(j)k} \times x_k,$$

for $j = 1, \ldots, M$.

Value

An object of class `summary.vglm` printed by specific methods defined at **VGAMextra** for objects with signature vgltsff-class.

Note

As for the intercept, notice that this is called *drift-term* at ARXff and ARMAXff, whilst it is refered as *intercept* in MAXff. This parameter is also estimated by TS family functions in **VGAMextra**. In the MA model, particularly, it is the mean of the process.

The drift-term, denoted as μ^*, is linearly linked to the mean, μ, of the AR and ARMA processes in turn, as follows:

$$\mu \to \frac{\mu^*}{1 - \sum \theta_i}.$$

Here, θ_i are the AR coefficients. Hence, the standard error for the *drift-term* is accordingly computed based on the asymptotic distribution of the mean. More precisely, the relation

$$V(\mu^*) = (1 - \sum \theta_i)^{-2} \times \frac{\sigma_\varepsilon^2}{n},$$

is considered, where σ_ε^2 is the variance of the random errors.

Finally, the AIC, AICC and BIC criteria are computed of the well-known expressions

$$AIC = (-2) \times Log - likelihood + 2 \times k$$

$$AICC = AIC + \frac{2\,k\,(k+1)}{n-k-1}$$

and

$$BIC = (-2) \times Log - likelihood + k \ \times \ ln(n)$$

with k denoting the number of parameters.

Author(s)

V. Miranda and T.W. Yee.

References

Woodward, H., Gray, H. and Elliot A. (2012) *Applied Time Series Analysis*. Taylor & Francis/CRC, Florida, USA.

See Also

vgtsff-class, summaryvlgm, ARXff, MAXff, ARMAXff, vglm.

Examples

```
#-------------------------------------------------------------------------#
# Fitting a simple Moving Average model to compare with arima().
#-------------------------------------------------------------------------#
set.seed(0628)
nn    <- 300
theta <- c(0.2, -0.37)  # Autoregressive coefficients
```

```
phi   <- c(0.25)        # MA coefficients.
mu    <- c(1.5, 0.85)   # Mean (not drift) of the MA process.
x2 <- runif(nn)

tsd1 <- mu[1]/(1 - sum(theta)) +
                  arima.sim(n = nn,
                            model = list(order = c(2, 0, 0),
                                         ar = theta),
                            sd = exp(1.5))
tsd2 <- mu[2]/(1 - sum(theta)) +
                  arima.sim(n = nn,
                            model = list(order = c(2, 0, 1),
                                         ar = theta, ma = phi),
                            sd = exp(1 + 2 * x2))

tsdata <- data.frame(TS1 = tsd1, TS2 = tsd2, x2 = x2)
head(tsdata)

    ###    An ARIMA(2, 0, 0) model, that is an AR(2) model    ###

#fit1 <- vglm(TS1 ~ 1,
#             ARIMAXff(order = c(2, 0, 0), var.arg = FALSE, type.EIM = "exact"),
#             data = tsdata,  crit = "log", trace = TRUE)

fit1 <- vglm(TS1 ~ 1,
             ARXff(order = 2, var.arg = FALSE, type.EIM = "exact"),
             data = tsdata,  crit = "log", trace = TRUE)
m.coe <- Coef(fit1)

## Using arima to compare to summary(vgtsff)
summary(fit1)
arima(tsdata$TS1, order = c(2, 0, 0)) ## Similar SE's than VGAMextra.


m.coe[1] / (1 - sum(m.coe[-(1:2)]))  # THIS IS SIMILAR TO THE INTERCEPT
                                     # ESTIMATED BY arima(): 1.1898

    ###    An ARIMA(2, 0, 1) models, that is an ARMA(2, 1)     ###
    ###   The errors standard deviation is a function of 'x2'  ###

### NOTICE: ARIMA and ARMA use the "identitylink" for coefficients ###
#fit2 <- vglm(TS2 ~ x2,
#             ARMAXff(order = c(2, 1), var.arg = FALSE, type.EIM = "exact",
#                     zero = NULL),
#            # constraints = list('x2' = rbind(0, 1, 0, 0, 0)),
#             data = tsdata,  crit = "loglikelihood", trace = TRUE)

#m.coe <- coef(fit2)
#coef(fit2, matrix = TRUE)

## Compare summary(vglm) to arima().
#summary(fit2)
#arima(tsdata$TS2, order = c(2, 0, 1))
```

`toppleMlink` *Link functions for the mean of 1–parameter continuous distribution: The Topp–Leone distribution.*

Description

Computes the `toppleMlink` transformation, its inverse and the first two derivatives.

Usage

```
toppleMlink(theta, bvalue = NULL, inverse = FALSE,
            deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. See `Links` and below for further details.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

The `toppleMlink` transformation arises as a link function to model the mean of the Topp–Leone distribution, `topple`. It is defined as

$$\eta = \texttt{logit}\left(\left(1 - \frac{4^s\Gamma(1+s)^2}{\Gamma(2+2s)}\right)/sup.tp\right).$$

Here, $0 < s < 1$ is a shape parameter as in `topple`, whereas $sup.tp$ is the *supremum* of

$$1 - \frac{4^s\Gamma(1+s)^2}{\Gamma(2+2s)},$$

in $(0, 1)$, as a function of s.

For numerical values of s out of $(0, 1)$, this link may result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the `toppleMlink` transformation of `theta` when `inverse = FALSE`. If `inverse = TRUE`, then `theta` becomes η, and the inverse transformation is required. However, it can't be expressed in close form. Therefore, the approximate *inverse image* of entered `theta` computed by `newtonRaphson.basic` is returned.

For `deriv = 1`, d `eta` / d `theta` when `inverse = FALSE`. If `inverse = TRUE`, then d `theta` / d `eta` as a function of `theta`.

Note

Values of s too close to zero or 1.0 may cause numerical instability. Use argument `bvalue` to replace them before computing the link.

If `theta` is character, then arguments `inverse` and `deriv` are ignored. See `Links` for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`topple`, `Links`, `newtonRaphson.basic`.

Examples

```
 ## E1. The toppleMlink() and its inverse ##
    theta <- ppoints(10)
    eta   <- toppleMlink(toppleMlink(theta = theta), inverse  =TRUE)
    summary(eta - theta)     # Zero


  ## E2. Some probability link functions ##

my.probs <- ppoints(100)

par(lwd = 2)
plot(my.probs, logitlink(my.probs), xlim = c(-0.1, 1.1), ylim = c(-5, 8),
     type = "l", col = "limegreen",
     ylab = "transformation", las = 1, main = "Some probability link functions")
lines(my.probs, geometricffMlink(my.probs), col = "gray50")
lines(my.probs, logffMlink(my.probs), col = "blue")
lines(my.probs, probitlink(my.probs), col = "purple")
lines(my.probs, clogloglink(my.probs), col = "chocolate")
lines(my.probs, cauchitlink(my.probs), col = "tan")
lines(my.probs, toppleMlink(my.probs), col = "black")
abline(v = c(0.5, 1), lty = "dashed")
abline(v = 0, h = 0, lty = "dashed")
legend(0.1, 8,
      c( "toppleMlink", "geometricffMlink", "logffMlink",
      "logitlink", "probitlink",
      "clogloglink", "cauchitlink"),
      col = c("black", "gray50", "blue", "limegreen", "purple", "chocolate", "tan"),
      lwd = 1, cex = 0.5)
par(lwd = 1)
```

toppleQlink *Link functions for the quantiles of several 1–parameter continuous distributions*

Description

Computes the toppleQlink transformation, its inverse and the first two derivatives.

Usage

```
toppleQlink(theta, p = stop("Argument 'p' must be specified."),
            bvalue = NULL, inverse = FALSE,
            deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. It is θ by default although it could be η depending upon other arguments. See Links for further details about this.
p	Numeric. A single value between 0 and 1. It is the p–quantile to be modeled by this link function.
bvalue, inverse, deriv, short, tag	See Links.

Details

This link function comforms with requirements of **VGAM** in order to be compatible within the VGLM/VGAM framework. That is, monotonic, onto, among other features. In this line, the toppleQlink transformation arises as the proper link to model any quantile of the Topp–Leone distribution topple. It is defined as

$$\frac{1-\sqrt{1-p^{1/s}}}{m.max}.$$

Here, s is a shape parameter lying in $(0,1)$ as in topple, whereas $m.max$ stands for the maximum in $(0,1)$ of

$$1-\sqrt{1-p^{1/s}}$$

as a function of s. Note, p is prespecified (fixed) between 0 and 1.

Numerical values of s or p out of range will result in Inf, -Inf, NA or NaN correspondingly.

Arguments inverse and deriv will be ignored if theta is character.

Value

For deriv = 0, the toppleQlink transformation of theta, when inverse = FALSE. If inverse = TRUE, then the inverse transformation log(p)/log(1 - (1 - theta * $m.max$)^2) is returned.

For deriv = 1, this function returns d eta / d theta, if inverse = FALSE. If inverse = TRUE, then the reciprocal d theta / d eta as a function of theta.

If deriv = 2, then the second order derivatives as a function of theta.

Warning

The expression $p^{1/s}$ tends rapidly to zero specially for values of s less than 0.005. Therefore, in such cases numerical values represented as zero may be returned when computing this link function, regardless the value of argument `inverse`.

Note

Numerical instability may occur for values `theta` too close to 0.0 or 1.0. Use argument `bvalue` to replace them before computing the link.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`topple`, `Links`.

Examples

```
## E1. toppleQLink() and its inverse. ##
my.s <- ppoints(50); p <- 0.75
max(my.s - toppleQlink(toppleQlink(my.s, p = p), p = p, inverse  =TRUE))   ## Zero
toppleQlink(theta = c(-0.15, -0.10, 0.25, 0.35) , p = p, inverse  = FALSE) ## NaNs
toppleQlink(theta = c(-0.15, -0.10, 0.25, 0.35) , p = p, inverse  = TRUE)  ## NaNs


## E2. toppleQLink() for different avlues of 'p'. ##

plot(toppleQlink(theta = my.s, p = 0.05) ~ my.s,
     type = "l", col = "blue", lty = "dotted", lwd = 3,
     xlim = c(-0.1, 1.65), ylim = c(-0.1, 1.1), las = 1,
     main = c("The toppleQlink() transformation"),
     ylab = "eta = toppleQlink", xlab = "theta")
abline(h = 0, v = 0, lwd = 2)
abline(h = 1, v = 1, lty = "dotted", col = "green")
lines(toppleQlink(theta = my.s, p = 0.25) ~ my.s, lwd = 2, lty = "dashed", col = "gray")
lines(toppleQlink(theta = my.s, p = 0.50) ~ my.s, lwd = 2, lty = "dashed", col = "brown")
lines(toppleQlink(theta = my.s, p = 0.75) ~ my.s, lwd = 2, lty = "dashed", col = "orange")
lines(toppleQlink(theta = my.s, p = 0.95) ~ my.s, lwd = 2, lty = "dashed", col = "gray50")
legend(1.1, 1.0, c("p = 0.05", "p = 0.25", "p = 0.50", "p = 0.75", "p = 0.95"),
       lwd = rep(2, 5), lty = rep("dashed", 5),
       col = c("blue", "gray", "brown", "orange", "gray50"))
```

`trinormalCovff` *Trivariate Normal Distribution Family Function*

Description

Estimates the means and the upper-half of the (symmetric) covariance matrix of a trivariate normal distribution by maximum likelihood.

Usage

```
trinormalCovff(zero = c("var", "cov"),
               lmean = "identitylink",
               lvar  = "loglink",
               lcov  = "identitylink")
```

Arguments

`zero`	The linear predictors modelled as intercept–only. See `zero` for further details.
`lmean, lvar, lcov`	Link functions applied to the means, variances (diagonal elements of the covariance matrix), and covariances (off-diagonal elements). See `Links` for more choices.

Details

This family function is similar to `trinormal`. The only difference is that the variances and covariances, instead of the standard deviations and correlation coefficients , are directly modelled and estimated. Similarly, `trinormalCovff` also fits linear models to the means of a trivariate normal distribution.

The fitted means are returned as the fitted values in the form of a three–column matrix. By default, the variances and covariances are modelled as intercept–only, where a `loglink` link is applied to the variances and a `identitylink` over the covariances.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Author(s)

Victor Miranda.

See Also

`trinormal`, `zero`, `Links`, `vglm`.

Examples

```
set.seed(123); nn <- 350
var1 <- exp(1.5); var2 <- exp(0.75); var3 <- exp(1.0)

### Artificial data, with two covariates.
tdata <- data.frame(x2 = runif(nn), x3 = runif(nn))
tdata <- transform(tdata,
                   y1 = rnorm(nn, 1 + 2 * x2, sd = sqrt(var1)),
                   y2 = rnorm(nn, 3 + 1 * x2, sd = sqrt(var2)),
                   y3 = rnorm(nn, 3 - 1 * x3, sd = sqrt(var2 * var3)))

### Fit the model using VGAMextra::trinormalCovff().
fit.trinormCovff <- vglm(cbind(y1, y2, y3) ~ x2 + x3,
                         trinormalCovff,
                         data = tdata, trace = TRUE)

summary(fit.trinormCovff)
vcov(fit.trinormCovff)

### Fitting the model using VGAM::trinormal()
fit.trinormVGAM <- vglm(cbind(y1, y2, y3) ~ x2 + x3,
                        trinormal,
                        data = tdata, trace = TRUE)

summary(fit.trinormVGAM)
vcov(fit.trinormVGAM)


#### Compare the estimated coefficients. Note that
#### trinormal() estimaes the sd's and correlation coeffs.
coef(fit.trinormCovff, matrix = TRUE)
coef(fit.trinormVGAM, matrix = TRUE)
```

---

uninormalff *Normal (distribution–specified) quantile regression*

Description

Distribution–specified quantile regression. An extension of `uninormal` from **VGAM**. It handles effectively `uninormalQlink` via the first linear predictor.

Usage

```
uninormalff(link1 = "identitylink", lsd = "loglink",
            percentile = 50,
            imethod = 1, isd = NULL, parallel = FALSE,
```

```
                smallno = 1.0e-5, zero = "sd")
```

Arguments

`link1`	Link function for the first linear predictor. By default `link1 = "identitylink"`, same as `lmean` from `uninormal`. Set `link1 = "uninormalQlink"` for normal quantile regression. See details below.
`percentile`	Numeric. A vector with the percentiles of interest, between 0 and 100. Used only when `link1 = "uninormalQlink"`.
`lsd, imethod, isd, parallel, smallno, zero`	Same as in `uninormal`, except that `"sd"` is the only accepted value for `zero`.

Details

An extension of `uninormal` adapted to handle `uninormalQlink`, for normal quantile regression (QR) via the first linear predictor.

The standard deviation only can be estimated. The second linear predictor is fixed to $\eta_2 = \log \sigma$, and `var.arg` is set internally to `FALSE`.

Unlike usual QR where the distribution of $Y|X$ is unspecified, `uninormalff()` estimates normal distributions at different quantiles (as entered in `percentile`) of the $Y|X$. For this, set `link1 = uninormaQlink`. To mimic `uninormal` set `link1 = "identitylink"` (default).

Initial developments of this work are in *Miranda & Yee (2019)*. See, e.g., `weibullRff`, for another example on distribution specified quantile regression with the two–parameter Weibull distribution.

Value

An object of class `"vglm"`. See `vglm-class` for full details.

Note

`Q.reg` must be used in the `vglm()` or `vgam()` to enter the response. See example below.

This **VGAM** family function does not handle censored data.

Author(s)

V. Miranda and Thomas W. Yee.

References

Miranda & Yee (2019) *New Link Functions for Distribution–Specific Quantile Regression Based on Vector Generalized Linear and Additive Models*. Journal of Probability and Statistics, Article ID 3493628.

Miranda & Yee (2021) *Two–Parameter Link Functions, With Application to Negative Binomial, Weibull and Quantile Regression*. In preparation.

See Also

uninormalQlink, uninormal, Q.reg, weibullQlink, weibullRff, CommonVGAMffArguments.

Examples

```
## Not run:

x2 <- seq(0,10,length.out = 100)             # independent variable
sig <- exp(0.5 + 0.15*x2)                    # non-constant variance
b_0 <- 10                                    # true intercept
b_1 <- 0.15                                  # true slope
set.seed(17221)                              # make the next line reproducible
e <- rnorm(100,mean = 0, sd = sig)         # normal random error with non-constant variance
y <- b_0 + b_1*x2 + e                        # dependent variable

## Data
ndata <- data.frame(y = y, x2 = x2)

## Some percentiles of interest
percentile <- c(10, 25, 50, 90)

## Normal quantile regression, zero = NULL
fit1 <- vglm(Q.reg(y, length.arg = 4) ~ x2,
            uninormalff(link1 = "uninormalQlink", percentile = percentile, zero = NULL),
             data = ndata, trace = TRUE)
#summary(fit1)
( my.coef3Q <- coef(fit1, mat = TRUE) )

## Plots - percentile curves.
plot(y ~ x2, pch = 19, ylim = c(-1, 25),
main =" Normal quantile regression")
abline(h = -3:25, v = 0, col = "gray", lty = "dashed")
with(ndata, lines(x2, my.coef3Q[1, 1] + my.coef3Q[2, 1] * x2,
                  col = "red", lty = "dotted", lwd = 4))
with(ndata, lines(x2, my.coef3Q[1, 3] + my.coef3Q[2, 3] * x2,
                  col = "orange", lty = "dotted", lwd = 4))
with(ndata, lines(x2, my.coef3Q[1, 5] + my.coef3Q[2, 5] * x2,
                  col = "blue", lty = "dotted", lwd = 4))
with(ndata, lines(x2, my.coef3Q[1, 7] + my.coef3Q[2, 7] * x2,
                  col = "brown", lty = "dotted", lwd = 4))
legend("topleft", c("90th", "50th", "25th", "10th"),
col = c("brown", "blue", "orange", "red"), lty = rep("dotted", 4), lwd = rep(4, 4))


## Mimicking 'VGAM:uninormal'
fit2 <- vglm(y ~ x2,  uninormalff(link1 = "identitylink", percentile = NULL, zero = NULL),
             data = ndata, trace = TRUE)


## End(Not run)
```

`uninormalQlink` *Quantile regression: Link function for the quantiles of the normal distribution.*

Description

Computes the `uninormalQlink` transformation, its inverse and the first two derivatives.

Usage

```
uninormalQlink(theta, percentile = stop("Enter percentiles."),
               sd = NULL, wrt.param = NULL,
               bvalue = NULL, inverse = FALSE,
               deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ ('mean' parameter) but iy may be η depending on the other parameters. See below for further details.
`percentile`	Numeric. A vector of percentiles of interest, denoted as `perc`.
`sd`	Numeric, positive. The 'standard deviation' parameter (required), denoted as σ.
`wrt.param`	Positive integer, either 1 or 2. The partial derivatives are computed with respect to one of the two linear predictors involved with this link. Further details listed below.
`bvalue, inverse, deriv, short, tag`	See `Links`.

Details

A 2-parameter link for the quantiles of the normal distribution. It can only be used within `uninormalff` as the first linear predictor. It is defined as

$$\texttt{uninormalQlink}(\mu;\sigma) = \eta_1(\mu;\sigma) = \mu + \sigma \cdot \Phi^{-1}(\texttt{perc}),$$

where Φ is the error function (see, e.g., `erf`), $\mu in(-\infty,\infty)$, and $\sigma > 0$. This link is expressly a function of $\theta = \mu$, therefore $sigma$ must be entered at every call.

Numerical values of σ out of range may result in `Inf`, `-Inf`, `NA` or `NaN`.

Value

For `deriv = 0`, the `uninormalQlink` transformation of `theta`, i.e. μ, when `inverse = FALSE`. If `inverse = TRUE`, then θ becomes η, and the inverse, $\eta - \sigma\Phi^{-1}(perc)$, *for given* σ, is returned.

When `deriv = 1` `theta` becomes $\theta = (\mu,\sigma) = (\theta_1,\theta_2)$, and $\eta = (\eta_1,\eta_2)$ with $\eta_2 = \log\ \sigma$, and the argument `wrt.param` must be considered:

A) If `inverse` = `FALSE`, then d `eta1` / d μ is returned when `wrt.param` = 1, and d `eta1` / d σ if `wrt.param` = 2.

B) For `inverse` = `TRUE`, this link returns d μ / d `eta1` and d σ / d `eta1` conformably arranged in a matrix, if `wrt.param` = 1, as a function of θ_i, $i = 1, 2$. When `wrt.param` = 2, then $d\mu$ / d `eta2` and $d\sigma$ / d `eta2` is returned.

For `deriv` = 2, the second derivatives in terms of `theta` are similarly returned.

Note

Numerical instability may occur for values of `sigma` too close to zero. Use argument `bvalue` to replace the former only before computing the link.

If `theta` is character, then arguments `inverse` and `deriv` are ignored. See `Links` for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

`uninormalff`, `uninormal`,

`Links`.

Examples

```
    eta <- seq(-3, 3, by = 0.1) # this is eta = log(Normal - Quantiles).
    sigma  <- exp(1)    # 'sigma' argument.
    percentile <- c(25, 50, 75, 95)  # some percentiles of interest.

 ## E1. Get 'mean' values.
   theta <- uninormalQlink(theta = eta, percentile = percentile,
                           sd = sigma, inverse = TRUE)  # Mu

 ## Not run:
 ## E2. Plot theta vs. eta, 'shape' fixed, for different percentiles.
plot(theta[, 1], eta, type = "l", las = 1, lty = 2, lwd = 3,
     ylim = c(-10, 10), xlim = c(-10, 10),
     main = "uninormalQlink(theta; shape), fixed 'shape'.",
     xlab = "Theta (scale)", ylab = "uninormalQlink")
abline(v = 0, h = 0, col = "red")
lines(theta[, 2], eta, lty = 2, lwd = 3, col = "blue")
lines(theta[, 3], eta, lty = 2, lwd = 3, col = "orange")
lines(theta[, 4], eta, lty = 2, lwd = 3, col = "red")
legend("bottomright", c("25th Perc", "50th Perc", "75th Perc", "95th Perc"),
      col = c("black", "blue", "orange", "red"), lty = c(2, 2, 2, 2),
      lwd = rep(3, 4))


## End(Not run)

 ## E3. uninormalQlink() and its inverse ##
```

```
etabis  <- uninormalQlink(theta = theta, percentile = percentile,
                          sd = sigma, inverse = FALSE)
my.diff <- eta - etabis
summary(my.diff)     # Zero
```

UtilitiesVGAMextra *Utility Functions for the* **VGAMextra** *Package*

Description

A set of common utility functions required by time series family functions at 'VGAMextra'.

Usage

```
Is.Numeric(x, isInteger  = FALSE, length.arg = NULL, Nnegative  = NULL)
is.FormulaAR(Model = ~ 1, Resp  = 1)
cross.gammas(x, y = NULL, lags = 1)
WN.lags(y, lags, to.complete = NULL)
extract.Residuals(object, TSprocess,...)
fittedVGAMextra(object,...)
weightsVGAMextra(object, type.w = "prior",...)
XLMmat(object,...)
```

Arguments

x	A vector of quantiles. Particularly, for Is.Numeric it is a single number (or vector) to be tested: Whether is numeric or not.
y	Vector of quantiles to be lagged. Then, the *cross - covariances* are computed from x and y_t, x and y_{t-1}, etcetera.
isInteger	Logical. If TRUE, it verifies that quantiles x are integers. Default is FALSE.
lags	Integer indicating the number of lags or *delays* to be applied to vector y. Then, calculate the cross-covariance between the pair of signals x and delayed samples computed from y.
length.arg	Integer. If length.arg > 0, it verifies that the length of x matches length.arg.
Model	Formula. A symbolic form of the models fitted by the vglm call. See formula for further details.
Nnegative	Logical. If TRUE, it verifies that x (all entries) are positive.
Resp	Integer. The number of *responses* in the Model. It must macth the number of respones entered in the vglm call.
object	An object of class 'vglm'. See vglm-class for details.

TSprocess	Logical, what time series model is being fitted. Choices are 'AR', 'MA', 'ARMA' and 'ARIMA'.
type.w	Character. What type of weights are to be used. Default is "prior". These are extracted from the slot @prior.weights of object.
to.complete	Use this argument to fill in the first 'p' observations when computing the lagged vectors in time series.
...	Additional parameters required by function extract.Residuals.

Details

A set of utility functions in **VGAMextra** for different purposes.

Specially for time series family functions in **VGAMextra** which involve specific checks on the majority of arguments entered by the user.

Value

Is.Numeric() returns a logical vector (or value) (TRUE or FALSE), after verifying whether quantiles x satisfies all conditions entered.

For is.FormulaAR(), this function returns a logical value, after verifying whether the expression entered for the Model argument in cm.ARMA is an object of class 'formula'.

Particularly, cross.gammas() computes either the single lagged covariance(s) from quantiles given in x or the lagged cross-covariance(s) from values given in x and y.

extract.Residuals() extracts the residuals of the process from slot @residuals, whilst

fittedVGAMextra and weightsVGAMextra return the fitted values and the weights from the vglm object, correspondingly.

isNA and inspectVGAMextra are essentially required when implementing link functions in **VGAMextra**.

Author(s)

V. Miranda and T. W. Yee.

See Also

cm.ARMA.

Examples

```
# Example 1.
myModel1 <- ~ x1 + x2
is.FormulaAR(myModel1)       # TRUE

test <- list( cbind(y1, y2) ~ x1, ~ x2 - 1)
is.FormulaAR(test)          # FALSE
is.FormulaAR(test[[1]], 2)  # TRUE

# Example 2.
```

```
x1 <- c(1:3, 4.5, -Inf)
Is.Numeric(x1)                                      # TRUE
Is.Numeric(x1, length.arg = 5)                      # TRUE
Is.Numeric(x1, length.arg = 5, isInteger = TRUE)    # FALSE
Is.Numeric(x1, length.arg = 5, Nnegative = TRUE)    # FALSE

# Example 3.
# Here, 'cross.gammas' computes Cov(x, y_{t - 1}), Cov(x, y_{t - 2}) and
# Cov(x, y_{t - 3}).

x <- runif(50)
y <- runif(50)
cross.gammas(x, y, lags = 3)
```

VARff *VGLTSM family function for the Order–p Vector Auto(R)egressive Model*

Description

Estimates an Order(p) Vector Autoregressive Models (VAR(p)) with white noise random errors by maximum likelihood estimation using Fisher scoring.

Usage

```
VARff(VAR.order = 1,
      zero = c("var", "cov"),
      lmean = "identitylink",
      lvar  = "loglink",
      lcov  = "identitylink")
```

Arguments

`VAR.order`	Length–1 (positive) integer vector. The order of the VAR to be fitted.
`zero`	Integer or character - string vector. Same as `MVNcov`. Details at `zero`.
`lmean, lvar, lcov`	Same as `MVNcov`.

Details

Let $\boldsymbol{x}_t = (x_{1,t}, \ldots, x_{K,t})^T$ be a time dependent vector of responses, with index $t = 1, \ldots, T$, and $\boldsymbol{\varepsilon}_t = (\varepsilon_{1,t}, \ldots, \varepsilon_{K,t})$ white noise with covariance matrix V.

`VARff` fits a linear model to the means of a K–variate normal distribution, where each variable, $x_{i,t}$, $i = 1, \ldots, K$, is a linear function of p–past lags of itself and past p–lags of the other variables. The model has the form

$$\boldsymbol{x}_t = \boldsymbol{\Phi_1}\boldsymbol{x}_{t-1} + \cdots + \boldsymbol{\Phi_p}\boldsymbol{x}_{t-p} + \boldsymbol{\varepsilon}_t,$$

where $\boldsymbol{\Phi_j}$ are $K \times K$ matrices of coefficients, $j = 1, \ldots, K$, to be estimated.

The elements of the covariance matrix are intercept–only by default.

Value

An object of class `"vglmff"` (see `vglmff-class`) to be used by VGLM/VGAM modelling functions, e.g., `vglm` or `vgam`.

Author(s)

Victor Miranda.

See Also

`MVNcov`, `zero`, `Links`, `ECM.EngleGran`, `vglm`.

Examples

```
set.seed(20170227)
nn <- 60
var.data <- data.frame(x2 = runif(nn, -2.5, 2.5))
var.data <- transform(var.data, y1 = rnorm(nn, 1.5 - 2 * x2, sqrt(exp(1.5))),
                                y2 = rnorm(nn, 1.0 - 1 * x2, sqrt(exp(0.75))),
                                y3 = rnorm(nn, 0.5 + 1 * x2, sqrt(exp(1.0))))

fit.var <- vglm(cbind(y1, y2, y3) ~ x2, VARff(VAR.order = 2),
                trace = TRUE, data = var.data)
coef(fit.var, matrix = TRUE)

summary(fit.var)
vcov(fit.var)
```

---

vgltsmff *Class of Vector Generalized Linear Time Series Models*

Description

Time series family functions for the **VGAMextra** package

Objects from the Class

Objects can be created by calling `new("vgltsmff"...)`

slots

Implementation of vector generalized linear time series (TS) family functions (*vgltsff*) at **VGAMextra** is entirely based on the structure of family functions of the class `vglmff-class`.

Hence, refer to `vglmff-class` for a thourugh description of slots and features involved when objects of class `"vgtsff"` are being created.

Methods

Thus far, the following methods for objects of class `"vgltsff-class"` are implemented:

`summary` Additional information to that displayed by the `summary` methods from **VGAM**. That is:

a) Standard errors based on the MLEs asymptotic distributions, and

b) Checks on stationarity and/or invertibility via the polynomial roots.

Currently, summary methods at **VGAMextra** have been implemented for:

`signature(VGAMff = "ARff")`: For ARX–types family functions.

`signature(VGAMff = "MAff")`: For MAX–types family functions.

`signature(VGAMff = "ARMAff")`: For ARMAX–like family functions.

See `summaryS4VGAMextra` for further details.

Note

Programmers to write VGAM/VGLM time series family functions are also allowed to write methods functions either for specific purposes, or to extend those current methods to print some extra output required.

In such cases, notice that the class `vgltsff-class` is *labeled* by an object of class `"character"` (a character vector) specified at the slot `@vfamily` within the family function. This is, in fact, one of the required slots by the class `vglmff-class`.

Additionally, practitioners are encouraged to mantain all previous conventions for naming the arguments in Ts family functions as specified at `vglmff-class`, e.g., `link` is the argument for parameter link functions, etc.

Author(s)

V. Miranda and T.W. Yee.

`weibullMlink` *Link functions for the mean of 2–parameter continuous distributions: The Weibull distribution.*

Description

Computes the `weibullMlink` transformation, its inverse and the first two derivatives.

Usage

```
weibullMlink(theta, shape = NULL, wrt.param = NULL,
             bvalue = NULL, inverse = FALSE,
             deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. This is θ ('scale' parameter) but it may be η depending on the other parameters. See below for further details.
shape	The shape parameter. Required for this link to work. See weibullRff.
wrt.param	Positive integer, either 1 or 2. The partial derivatives are computed with respect to one of the two linear predictors involved with this link. Further details listed below.
bvalue, inverse, deriv, short, tag	See Links.

Details

This is the link for the mean of the 2–parameter Weibull distribution, also known as the weibullMlink transformation. It can only be used within weibullRff and is defined as

$$\texttt{weibullMlink}(\beta;\alpha) = \eta(\beta;\alpha) = \log[\beta \cdot \Gamma(1 + 1/\alpha)],$$

for *given* α ('shape' parameter) where $\beta > 0$ is the *scale* parameter.

weibullMlink is expressly a function of β, i.e. θ, therefore α (*shape*) must be entered at every call.

Numerical values of α or β out of range may result in Inf, -Inf, NA or NaN.

Value

For deriv = 0, the weibmeanlnik transformation of theta, i.e., β, when inverse = FALSE. If inverse = TRUE, then θ becomes η, and the inverse, $\exp\left(theta - \log\Gamma(1 + 1/\alpha)\right)$, *for given* α, is returned.

When deriv = 1 theta becomes $\theta = (\beta, \alpha) = (\theta_1, \theta_2)$, and $\eta = (\eta_1, \eta_2)$ with $\eta_2 = \log\ \alpha$, and the argument wrt.param must be considered:

A) If inverse = FALSE, then d eta1 / $d\ \beta$ is returned when wrt.param = 1, and d eta1 / $d\ \alpha$ if wrt.param = 2.

B) For inverse = TRUE, this function returns $d\ \beta$ / d eta1 and $d\ \alpha$ / d eta1 conformably arranged in a matrix, if wrt.param = 1, as a function of θ_i, $i = 1, 2$. When wrt.param = 2, a matrix with columns $d\beta$ / d eta2 and $d\alpha$ / d eta2 is returned.

For deriv = 2, the second derivatives in terms of theta are likewise returned.

Note

Numerical instability may occur for values theta too close to zero. Use argument bvalue to replace them before computing the link.

If theta is character, then arguments inverse and deriv are ignored. See Links for further details.

Author(s)

V. Miranda and Thomas W. Yee.

See Also

weibullQlink, weibullRff, weibullR, lgamma, Links.

Examples

```
    eta <- seq(-3, 3, by = 0.1) # this is eta = log(mu(b, a)).
    shape  <- exp(1)    # 'shape' argument.

 ## E1. Get 'scale' values with A WARNING (not the same length)!
   theta <- weibullMlink(theta = eta, shape = shape, inverse = TRUE)  # Scale

 ## Not run:
 ## E2. Plot theta vs. eta, 'shape' fixed.
 plot(theta, eta, type = "l", ylab = "", col = "blue",
      main = paste0("weibullMlink(theta; shape = ",
                    round(shape, 3), ")"))
  abline(h = -3:3, v = 0, col = "gray", lty = "dashed")

## End(Not run)

 ## E3. weibullMlink() and its inverse ##
    etabis  <- weibullMlink(theta = theta, shape = shape, inverse = FALSE)
    summary(eta - etabis)     # Should be 0
```

---

weibullQlink *Weibull Quantile regression: Link function for the quantiles of the Weibull distribution.*

Description

Computes the weibullQlink transformation, its inverse and the first two derivatives.

Usage

```
weibullQlink(theta, percentile = stop("Enter percentiles."),
             shape = NULL, wrt.param = NULL,
             bvalue = NULL, inverse = FALSE,
             deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

theta	Numeric or character. Same as uninormalQlink
percentile	Same as uninormalQlink. Denoted below as perc.
shape	Numeric, positive. The shape parameter, required.
wrt.param	Same as in uninormalQlink
bvalue, inverse, deriv, short, tag	See Links.

Details

The ordinary scale–shape Weibull quantiles are directly modelled by this link, aka weibullQlink transformation. It can only be used within weibullRff as the first linear predictor, η_1, and is defined as

$$\texttt{weibullQlink}(\beta;\alpha) = \eta_1(\beta;\alpha) = \log\{\beta \cdot [(-\log(1 - \texttt{perc}))^{(1/\alpha)}]\},$$

for *given* α ('shape' parameter) where $\beta > 0$ is the *scale* parameter.

weibullQlink is expressly a function of β, i.e. θ, therefore α (*shape*) must be entered at every call.

Numerical values of α or β out of range may result in Inf, -Inf, NA or NaN.

Value

For deriv = 0, the weibullQlink transformation of theta, i.e. β, when inverse = FALSE. If inverse = TRUE, then θ becomes η, and the inverse, exp[theta - $(1/\alpha)log(-log(1 - perc))$], *for given* α, is returned.

When deriv = 1 theta becomes $\theta = (\beta, \alpha) = (\theta_1, \theta_2)$, and $\eta = (\eta_1, \eta_2)$ with $\eta_2 = \log\ \alpha$, and the argument wrt.param must be considered:

A) If inverse = FALSE, then d eta1 / d β is returned when wrt.param = 1, and d eta1 / d α if wrt.param = 2.

B) For inverse = TRUE, this link returns d β / d eta1 and d α / d eta1 conformably arranged in a matrix, if wrt.param = 1, as a function of θ_i, $i = 1, 2$. When wrt.param = 2, a matrix with columns $d\beta$ / d eta2 and $d\alpha$ / d eta2 is returned.

For deriv = 2, the second derivatives in terms of theta are similarly returned.

Note

See weibullMlink.

Author(s)

V. Miranda and Thomas W. Yee.

References

Miranda & Yee (2021) *Two–Parameter Link Functions, With Application to Negative Binomial, Weibull and Quantile Regression*. In preparation.

See Also

weibullRff, Q.reg, weibullR, weibmeanlink, Links.

Examples

```
    eta <- seq(-3, 3, by = 0.1) # this is eta = log(Weibull-quantiles).
    shape  <- exp(1)    # 'shape' argument.
    percentile <- c(25, 50, 75, 95)  # some percentiles of interest.

 ## E1. Get 'scale' values. Gives a warning (not of the same length) !
   theta <- weibullQlink(theta = eta, percentile = percentile,
                         shape = shape, inverse = TRUE)  # Scale

 ## Not run:
 ## E2. Plot theta vs. eta, 'shape' fixed, for different percentiles.
plot(theta[, 1], eta, type = "l", lwd = 3,
     ylim = c(-4, 4),
     main = paste0("weibullQlink(theta; shape = ", round(shape, 3), ")"),
     xlab = "Theta (scale)", ylab = "weibullQlink")
abline(h = -3:3, v = 0, col = "gray", lty = "dashed")
lines(theta[, 2], eta, lwd = 3, col = "blue")
lines(theta[, 3], eta, lwd = 3, col = "orange")
lines(theta[, 4], eta, lwd = 3, col = "red")
legend("bottomright", c("25th Perc", "50th Perc", "75th Perc", "95th Perc"),
      col = c("black", "blue", "orange", "red"),
      lwd = rep(3, 4))

## End(Not run)

 ## E3. weibullQlink() and its inverse ##
    etabis  <- weibullQlink(theta = theta, percentile = percentile,
                            shape = shape, inverse = FALSE)
    summary(eta - etabis)     # Should be 0 for each colum (percentile)
```

---

weibullRff *Distribution–specified quantile regression: 2–parameter Weibull Distribution*

Description

Estimates the 2–parameter Weibull distribution by maximum likelihood. An extension of weibullR from **VGAM**. Weibull quantile regression and Weibull–mean modelling are also handled via the first linear predictor.

Usage

```
weibullRff(link1 = c("loglink", "weibullMlink", "weibullQlink")[1],
           lshape = "loglink", percentile = 50,
```

```
imu = NULL, iscale = NULL, ishape = NULL,
lss = TRUE, nrfs = 1, probs.y = c(0.2, 0.5, 0.8),
imethod = 1, zero = "shape")
```

Arguments

link1	Link function for the first linear predictor. Default is link1 = "loglink", mimicking weibullR. The other options are the 2–parameter weibullQlink, applied to the Weibull quantile function, and the 2–parameter weibullMlink, applied to the Weibull mean function. See below for more details.
percentile	Numeric. A vector with the percentiles of interest, between 0 and 100. Used only in Weibull quantile regression, that is, when link1 = "weibullQlink".
lshape,imu, iscale, ishape, lss, nrfs, probs.y, imethod	Same as weibullR.
zero	Specifies the parameters to be modelled as intercept–only. Further details below. See CommonVGAMffArguments.

Details

weibullRff is a modified version of weibullR adapted to handle weibullQlink and weibullMlink, two 2-parameter linear predictors that model the Weibull mean and quantiles respectively.

The underlying density is the ordinary scale(β) & shape(α) Weibull density (see weibullR).

The second linear predictor is always $\eta_2 = \log\ \alpha$. The argument link1 handles the first linear predictor.

** **Mimicking** weibullR **

The default is link1 = "loglink", i.e., $\eta_1 = \log\ \beta = \log\ scale$, and $\eta_2 = \log\ \alpha = \log\ shape$, as with weibullR. The mean (μ) is returned as the fitted value.

** **Weibull quantile regression** **

For Weibull quantile regression set link1 = "weibullQlink" and enter a numeric vector of percentiles of interest via percentile. See examples.

NOTE: Enter the response using Q.reg. See example below. The Weibull quantiles are returned as the fitted values.

** **Weibull-mean modelling** **

For Weibull-mean modelling (viz. mean time to failure) set link1 = "weibullMlink". The mean (μ) is returned as the fitted value.

Value

An object of class "vglm". See vglm-class for full details.

Note

The parameters α and β match the arguments $shape$ and $scale$ from rweibull.

Multiple responses are handled.

This **VGAM** family function does not handle censored data.

Author(s)

V. Miranda and Thomas W. Yee.

References

Miranda & Yee (2021) *Two–Parameter Link Functions, With Application to Negative Binomial, Weibull and Quantile Regression*. In preparation.

See Also

Q.reg, weibullQlink, weibullMlink, weibullR, gamma,

CommonVGAMffArguments.

Examples

```
## Not run:
set.seed(18121)
nn <- 300
x2 <- sort(runif(nn, 0, 3))  # Predictor/covariate.
bb <- exp(1.1 + 0.2 * x2)    # Scale parameter as function of x2.
aa <- exp(1.0 - 0.35 * x2)     # Shape parameter as function of x2.
mymu <- bb * gamma(1 + 1/aa)  # The Weibull mean.

## Use weibullMlink to generate appropriate scale parameter.
newbb <- weibullMlink(theta = log(mymu), shape = aa, inverse = TRUE, deriv = 0)

## A single random response
wdata <- data.frame(y1 = rweibull(nn, shape = aa, scale = newbb), x2 = x2)

# Plotting the data / Histogram
plot(y1  ~ x2, xlim = c(0, 3.1), ylim = c(-1, 35),
     pch = 20, data = wdata, col = "black",
     main = "Weibull Quantile regression~ x2")
abline(h = 0, v = 0, col = "grey", lty = "dashed")
with(wdata, hist(y1, col = "red", breaks = 15))

## Weibull regression - percentile = c(25, 50, 75)
## Note the use of Q.reg.
fit1 <- vglm(Q.reg(y1, length.arg = 3) ~ x2,
             weibullRff(link1 = "weibullQlink", zero = NULL,
                                percentile = c(25, 50, 75)),
             trace = TRUE, data = wdata)
head(fitted(fit1))
summary(fit1)
my.coef3Q <- coef(fit1, mat = TRUE)

### Proportion of data below the estimated 25% Quantile line.
100 * (1 - (sum(wdat$y1 >= fitted(fit2)[, 1]) / nn))  # Around 25%
### Proportion of data below the estimated 50% Quantile line.
100 * (1 - (sum(wdat$y1 >= fitted(fit2)[, 2]) / nn))   # Around 50%
### Proportion of data below the estimated 75% Quantile line.
```

```
100 * (1 - ( sum(wdat$y1 >= fitted(fit2)[, 3]) / nn ))   # Around 75%

## The quantile plots ##
my.coef3Q <- coef(fit2, matrix = TRUE)
with(wdat, lines(x2, exp(my.coef3Q[1, 1] + my.coef3Q[2, 1] * x2),
                   col = "red", lty = "dotted", lwd = 4))
with(wdat, lines(x2, exp(my.coef3Q[1, 3] + my.coef3Q[2, 3] * x2),
                 col = "orange", lty = "dotted", lwd = 4))
with(wdat, lines(x2, exp(my.coef3Q[1, 5] + my.coef3Q[2, 5] * x2),
                 col = "blue", lty = "dotted", lwd = 4))

## Adding the 'mean' or expected Weibull regression line.
fit2 <- vglm(y1 ~ x2,
             weibullRff(link1 = "weibullMlink", zero = NULL),
             trace = TRUE, data= wdat)
my.coef3Q <- coef(fit2, mat = TRUE)
with(wdat, lines(x2, exp(my.coef3Q[1, 1] + my.coef3Q[2, 1] * x2),
                 col = "yellow", lty = "dashed", lwd = 3))


legend("topleft", c("25h Perc", "50th Perc", "Mean", "75th Perc"),
       col = c("red", "orange", "cyan", "blue"),
       lty = c("dashed", "dashed", "dashed", "dashed"), lwd = rep(4, 4))

## End(Not run)
```

`WN.InitARMA` *Estimated White Noise (WN) from the autoregressive moving-average model of order-(p, q) [ARMA(p, q)].*

Description

Estimates the unobserved white noise of the ARMA(p, q) model via the corresponding inverted process.

Also, provides the initial values of `ARXff`, `MAXff`, and `ARMAXff` family functions.

Usage

```
WN.InitARMA(tsData    = NULL,
            order     = c(1, 0, 1),
            whiteN    = FALSE,
            moreOrder = 0,
            updateWN  = FALSE)
```

Arguments

`tsData`	A univariate data frame containing the time series to be fitted according to an ARMA(p, q) process. Data must be of class "`ts`".

`order` A vector with three integer components. It is order of the ARMA model to be inverted. These entries, $c(p, d, q)$, are the AR order, the degree of differencing, and the MA order, respectively.

`whiteN` Logical. If `TRUE`, then the estimated white noise computed from the inverted ARMA model is returned. This option is enabled only for `MAXff`, `ARMAXff` family functions.

`moreOrder` A non-negative integer (might be zero) used to increment the order of the AR model initially fitted to estimate the residuals, i.e., an AR(p + `moreOrder`) model. Empirically, values of `moreOrder` > 2 do NOT improve accuracy of estimates. This assert, however, may vary for different time series family functions.

`updateWN` Logical. if `TRUE`, the white noise is *updated* through a second regression of Y_t on $Y_{t-1}, \ldots, Y_{t-p}, \widehat{\varepsilon_{t-1}}, \ldots, \widehat{\varepsilon_{t-q}}$.

Details

Overall, the autoregressive moving average process of order $c(p, q)$, shortly denoted as ARMA(p, q), with *intercept* μ can be expressed as

$$y_t = \mu + \theta_1 y_{t-1} + \ldots + \theta_p y_{t-p} + \phi_1 \varepsilon_{t-1} + \ldots + \phi_q \varepsilon_{t-q} + \varepsilon_t.$$

It is well known that it can be expressed in terms of an autoregressive process of infinite order, AR(∞), by recursive substitutions. For instance, given a mean-zero ARMA(1, 1),

$$y_t = \theta_1 y_{t-1} + \phi_1 \varepsilon_{t-1} + \varepsilon_t, \qquad (1)$$

one may express

$$\varepsilon_{t-1} = Y_{t-1} - (\theta_1 y_{t-2} + \phi_1 \varepsilon_{t-2}$$

Substituting this equation in (1) yields the initial inverted process, as follows:

$$y_t = \psi_1 y_{t-1} + \psi_2 y_{t-2} + f(\varepsilon_{t-2}, \varepsilon_t).$$

where f is a function of ε_{t-2} and ε_t.

Repeated substitutions as above produces the so-called *inverted process*,

$$y_t = \sum_{k=1}^{\infty} \psi_k y_{t-k} + \varepsilon_t. \qquad (2)$$

$k = 1, \ldots, \infty$. Hence, setting an acceptable order (via the `moreOrder` argument, 1 or 2 for instance), an AR(p + `moreOrd`) *inverted* model is internally fitted within `WN.InitARMA`. Consequently, the unobserved white noise, $\{\varepsilon_t\}$, is estimated by computing the *residuals* in (2), after regression. `whiteN = TRUE` enables this option.

Finally, initial values of the `MAXff`, and `ARMAXff` family functions can be computed by least squares from the estimated white noise above, $\{\varepsilon_t\}$ and the given data, $\{t_t\}$.

Initial values of `ARXff` are also internally computed using $\{t_t\}$ only.

Value

A list with the following components:

`Coeff`	The initial values of the VGLM/VGAM family function in turn: `ARXff`, `MAXff`, or `ARMAXff`.
`whiteN`	(Optional) Estimated white noise enabled only for `MAXff`, `ARMAXff` . That sequence is returned if `whiteN = TRUE`.

Warning

For some time series family functions, `MAXff` for instance, values of `moreOrder` > 3 do NOT improve the accuracy of estimates, and may lead the algorithm to failure to converge.

Author(s)

Victor Miranda and T. W. Yee.

References

Brockwell, P. and Davis, R. (2002) *Introduction to Time Series and Forecasting*. Springer, New York, USA.

Durbin, J. (1959) Efficient Estimation of Parameters in Moving-Average Models. *Biometrika*, **46**, pp 306–316.

See Also

`MAXff`, `ARMAXff`.

Examples

```
# Generating some data -> an MA(3)
set.seed(1004)
mydata <- arima.sim( n = 200, list(ma = c(0.3, 0.56 , 0.11)) )

# Computing initial values to be passed to MAXff()
WN.InitARMA(tsData = data.frame(y = mydata),
            order = c(0, 0, 3),
            moreOrder = 1)

# Returning initial values and white noise.
initMA <- WN.InitARMA(tsData = data.frame(y = mydata),
                      order = c(0, 0, 3),
                      moreOrder = 1,
                      whiteN = TRUE)

# Initial values passed to MAXff()
initMA$Coeff
```

```
# Estimated white noise
head(initMA$WhiteNoise)
```

`yulesimonMlink` *Link functions for the mean of 1–parameter discrete distributions: The Yule–Simon Distribution.*

Description

Computes the `yulesimonMlink` transformation, its inverse and the first two derivatives.

Usage

```
yulesimonMlink(theta, bvalue = NULL, inverse = FALSE,
               deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ by default, or η depending upon other arguments. See `Links`.
`bvalue, inverse, deriv, short, tag`	Details at `Links`

Details

Assume $Y \sim \text{Yule} - \text{Simon}(\rho)$, where ρ is a shape parameter as in `yulesimon`. Then, the mean of Y is given by

$$\mu = \frac{\rho}{\rho - 1} = (1 - \rho^{-1})^{-1},$$

provided $\rho > 1$.

This link function may be conceived as a natural link function for the mean of the Yule–Simon distribution which comes up by taking the logarithm on both sides of this equation. More precisely, the `yulesimonMlink` tranformation for $\rho > 1$ is given by

$$\texttt{yulesimonMlink}(\rho) = -\log(1 - \rho^{-1}).$$

While this link function can be used to model any parameter lying in $(1, \infty)$, it is particularly useful for event-rate data where the mean, μ, can be written in terms of some rate of events, say λ, and the timeframe observed t. Specifically,

$$\mu = \lambda t.$$

Assuming that additional covariates might be available to linearly model λ (or $\log \lambda$), this model can be treated as a VGLM with one parameter where the time t (as $\log t$) can be easily incorporated in the analysis as an offset.

Under this link function the domain set for ρ is $(1, \infty)$. Hence, values of ρ too close to 1 from the right, or out of range will result in `Inf`, `-Inf`, `NA` or `NaN`. Use argument `bvalue` to adequately replace them before computing the link function.

Unlike `logffMlink` or `zetaffMlink`, the inverse of this link function can be written in close form.

If `theta` is a character, arguments `inverse` and `deriv` are disregarded.

Value

For `deriv = 0`, the `yulesimonMlink` transformation of `theta` when `inverse = FALSE`, and if `inverse = TRUE` then `exp(theta) / (exp(theta) - 1)`.

For `deriv = 1`, d `eta` / d `theta` as a function of `theta` if `inverse = FALSE`, else the reciprocal d `theta` / d `eta`.

For `deriv = 2` the second order derivatives are correspondingly returned.

Warning

Conforming with `yulesimon`, the domain set for rho is $(0, \infty)$. However, in order for `yulesimonMlink` to be a real number, rho must be greater then 1.0. Then, when a VGLM is fitted via `yulesimon` using this link function, numerical instability will occur if the estimated or the true value of rho lies between 0 and 1, or if the initial values for rho generated by `yulesimon` fail to meet $rho > 1$. Alternatively, try `posPoiMlink` or `loglink` if this happens.

Note

If the underlying assumption $\rho > 1$ is not met, then this function returns `NaN`. This is equivalent to claim that the mean is infinite or negative and, consequently, its logarithm will not be real.

The vertical line `theta = 1` is an asymptote for this link function. As a result, it may return `Inf` for values of ρ too close to 1 from the right.

Author(s)

V. Miranda and T. W. Yee

See Also

`yulesimon`, `Links`, `posPoiMlink`, `loglink`.

Examples

```
## Example 1  ##
Shapes <- 1:10
yulesimonMlink(theta = Shapes, deriv = 1)  ## d eta/d theta, as function of theta

yulesl.inv <-
```

```
    # The inverse  minus actual values
    yulesimonMlink(theta = yulesimonMlink(theta = Shapes),  inverse = TRUE) - Shapes

summary(yulesl.inv)     ## zero


## Example 2. Special values of theta (rho) ##
rhos <- c(-Inf, -2, -1, 0.0, 0.5, 1, 5, 10, 100, Inf, NaN, NA)
rbind(rho = rhos,
      yuleslink  = yulesimonMlink(theta = rhos),
      inv.yulesl =yulesimonMlink(theta = rhos, inverse = TRUE))


## Example 3 The yulesimonMlink transformation and the first two derivatives ##


rhos <- seq(1, 20, by = 0.01)[-1]
y.rhos <- yulesimonMlink(theta = rhos, deriv = 0)
der.1  <- yulesimonMlink(theta = rhos, deriv = 1)
der.2  <- yulesimonMlink(theta = rhos, deriv = 2)

plot(y.rhos ~ rhos, col = "black",
     main = "log(mu), mu = E[Y], Y ~ Yule-Simon(rho).",
     ylim = c(-5, 10), xlim = c(-1, 5), lty = 1, type = "l", lwd = 3)
abline(v = 1.0, col = "orange", lty = 2, lwd = 3)
abline(v = 0, h = 0, col = "gray50", lty = "dashed")

lines(rhos, der.1, col = "blue", lty = 5)
lines(rhos, der.2, col = "chocolate", lty = 4)
legend(2, 7, legend = c("yulesimonMlink", "deriv = 1", "deriv = 2"),
       col = c("black", "blue", "chocolate"), lty = c(1, 5, 4))
```

`zetaffMlink` *Link functions for the mean of 1–parameter discrete distributions: The Zeta Distribution.*

Description

Computes the `zetaffMlink` transformation, including its inverse and the first two derivatives.

Usage

```
zetaffMlink(theta, bvalue = NULL,
            alg.roots = c("Newton-Raphson", "bisection")[1],
            inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE)
```

Arguments

`theta`	Numeric or character. This is θ by default, although it can be η sometimes, depending on the other parameters. See below for further details.
`bvalue`	Details at `Links`.
`alg.roots`	Character. The iterative method to find the inverse of this link function. Default is Newton–Raphson. Optionally, the bisection method is also available.
`inverse, deriv, short, tag`	Details at `Links`

Details

This is a link function for the zeta distribution, `zetaff`, which emerges by applying the logarithm transformation to its mean. Specifically, assume Y follows a zeta distribution with shape parameter s (this is `theta` in the VGLM/VGAM framework). Then, the mean of Y is

$$\mu = \frac{\zeta(s)}{\zeta(s+1)},$$

provided $s > 1$, where ζ is the Riemann's zeta function computed by `zeta`. The notation adopted here conforms with `zetaff` in terms of the density of the zeta distribution.

The `zetaffMlink` transformation is given by

$$\eta = \texttt{zetaffMlink}(\texttt{s}) = \log \frac{\zeta(\texttt{s})}{\zeta(\texttt{s}+1)}.$$

It is particularly useful when modelling event–rate data where the expected number of events, μ, can be modelled as

$$\mu = \lambda t.$$

Specifically, λ is a standardized mean per unit–time, and t is the observed timeframe.

The domain set for s, i.e. θ, is $(1, \infty)$. Hence, either large values of s, or those too close to 1 from the right, or out of range will result in `Inf`, `-Inf`, `NA` or `NaN`. Use argument `bvalue` to adequately replace them before computing the link function.

WARNING: While in `zetaff` the parameter s lies in $(1, \infty)$, `zetaffMlink` will be real when $s > 1$. Consequently, for any VGLM fitted via `zetaff` using this link function, numerical problems will take place if any s value lies between 0.0 and 1.0 at any iteration. Use optional link functions like `loglink`.

When `inverse = TRUE` and `deriv = 0`, s changes into η, and therefore the domain set (only in this case) turns into $(0, \infty)$. See below for further details.

If `theta` is a character, arguments `inverse` and `deriv` are disregarded.

Value

For `deriv = 0`, the `zetaffMlink` transformation of `theta`, if `inverse = FALSE`. When `inverse = TRUE`, `theta` becomes η, and then the inverse of `zetaffMlink` is required. However, it cannot be written in closed–form. Instead, the inverse image of η, say θ_η, is returned. That is, a unique vector θ_η such that

$$\texttt{zetaffMlink}(\theta_\eta) = \eta.$$

This process is equivalent to find the root, θ_η, of the function $\texttt{zetaffMlink}(\theta) - \eta$, which is internally carried out via the method entered at `alg.roots`. Options available are "`Newton-Raphson`" and "`bisection`".

For `deriv = 1`, d `eta` / d `theta` as a function of `theta` if `inverse = FALSE`, else the reciprocal d `theta` / d `eta`.

Similarly, when `deriv = 2` the second order derivatives are returned accordingly.

The first two derivatives of the Riemman's zeta function are computed by `zeta`.

Besides, the `zetaffMlink` function as well as its derivatives are graphically delimited for specific asymptotes. Consequently, the mathematical limit of this link function is returned for special values of `theta`, e.g. for `theta` = ∞. See example 2 below.

Warning

Where the inverse image of η, θ_η, is required, values entered at `theta` (becoming η) must be non-negative. The reason is that the `zetaffMlink` transformation is decreasing but strictly positive in $(1, \infty)$ asymptotically approaching to the horizontal axis. In this way, the *shifted–down* `zetaffMlink` function

$$\texttt{zetaff.func}(\theta|\eta) = \texttt{zetaffMlink}(\theta) - \eta$$

uniquely intersects the horizontal axis and hence the inverse image computed by "`Newton-Raphson`" or "`bisection`" will be a real number.

Note

Overall, this link function is useful to model any parameter lying in $(1, \infty)$, specially if the theoretical mean can be written as $\mu = \lambda t$, as stated above. As a result, some problems may arise if there are covariates. Try another link function if any issue, such as `logloglink`.

Author(s)

V. Miranda and T. W. Yee

See Also

`zetaff`, `newtonRaphson.basic`, `bisection.basic`, `zeta`, `loglink`, `Links`.

Examples

```
## Example 1  ##
Shapes <- 1:10 + 0.1
zetaffMlink(theta = Shapes, deriv = 1)  ## d eta/d theta, as function of theta

zetafflk.inv <- zetaffMlink(theta = zetaffMlink(theta = Shapes), inverse = TRUE) - Shapes

summary(zetafflk.inv)                      ## Should be zero


## Example 2. Special values of theta, inverse = FALSE ##
Shapes <- c(-Inf, -1, 0.5, 1, 1.5, 10, 100, Inf, NaN, NA)
print(rbind(Shapes, zetaffMlink  = zetaffMlink(theta = Shapes),
            inv.zfflink = zetaffMlink(theta = Shapes, inverse = TRUE)), digits = 3)


## Example 3. Plot of 'zetaffMlink()' and its first two derivatives ##
## inverse = FALSE, deriv = 0, 1, 2 ##

Shapes <- seq(1, 20, by = 0.01)[-1]
y.shapes <- zetaffMlink(theta = Shapes, deriv = 0)
der.1 <- zetaffMlink(theta = Shapes, deriv = 1)
der.2 <- zetaffMlink(theta = Shapes, deriv = 2)

plot(y.shapes ~ Shapes,
     col = "black", main = "log(mu), mu = E[Y], Y ~ Zeta(s).",
     ylim = c(-5, 10), xlim = c(-0.1, 5), lty = 1, type = "l", lwd = 3)
abline(v = 1.0, col = "orange", lty = 2, lwd = 3)
abline(v = 0, h = 0, col = "gray50", lty = "dashed")

lines(Shapes, der.1, col = "blue", lty = 5)
lines(Shapes, der.2, col = "chocolate", lty = 4)
legend(2, 7, legend = c("zetaffMlink", "deriv = 1", "deriv = 2"),
       col = c("black", "blue", "chocolate"), lty = c(1, 5, 4), lwd = c(3, 1, 1))
```

Index