Package ‘statmod’

June 18, 2017

Version 1.4.30
Date 2017-06-16
Title Statistical Modeling
Author Gordon Smyth [cre, aut], Yifang Hu [ctb], Peter Dunn [ctb], Belinda Phipson [ctb], Yun-shun Chen [ctb]
Maintainer Gordon Smyth <smyth@wehi.edu.au>
Depends R (>= 1.6.1)
Imports stats, graphics
Suggests MASS, tweedie
Description
A collection of algorithms and functions to aid statistical modeling. Includes growth curve comparisons, limiting dilution analysis (aka ELDA), mixed linear models, heteroscedastic regression, inverse-Gaussian probability calculations, Gauss quadrature and a secure convergence algorithm for nonlinear models. Includes advanced generalized linear model functions that implement secure convergence, dispersion modeling and Tweedie power-law families.
License GPL-2 | GPL-3
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-06-18 14:56:34 UTC

R topics documented:

statmod-package ......................................................... 2
Digamma ................................................................. 3
elda ................................................................. 5
fitNBP ............................................................... 8
forward .............................................................. 9
gauss.quad ........................................................... 10
gauss.quad.prob .................................................. 12
glm.scoretest ....................................................... 13
glmgam.fit .......................................................... 15
growthcurve ......................................................... 16
Introduction to the StatMod Package

Description

This package includes a variety of functions for numerical analysis and statistical modelling. The functions are briefly summarized by type of application below.

Generalized Linear Models

The function \texttt{tweedie} defines a large class of generalized linear model families with power variance functions. It used in conjunction with the \texttt{glm} function, and widens the class of families that can be fitted.

\texttt{qresiduals} implements randomized quantile residuals for generalized linear models.

The functions \texttt{canonic.digamma}, \texttt{unitdeviance.digamma}, \texttt{varfun.digamma}, \texttt{cumulant.digamma}, \texttt{d2cumulant.digamma}, \texttt{meanval.digamma} and \texttt{logmdigamma} are used to fit double-generalized models, in which a link-linear model is fitted to the dispersion as well as to the mean. Specificaly they are used to fit the dispersion submodel associated with a gamma glm.

Growth Curves

\texttt{compareGrowthCurves}, \texttt{compareTwoGrowthCurves} and \texttt{meanT} are functions to test for differences between growth curves with repeated measurements on subjects.

Limiting Dilution Analysis

The \texttt{limdil} function is used in the analysis of stem cell frequencies. It implements limiting dilution analysis using complementary log-log binomial generalized linear model regression, with some improvements on previous programs.
**Probability Distributions**

The functions `qinvgauss`, `dinvgauss`, `pinvgauss` and `rinvgauss` provide probability calculations for the inverse Gaussian distribution. `gauss.quad` and `gauss.quad.prob` compute Gaussian Quadrature with probability distributions.

**Tests**

- `hommel.test` performs Hommel’s multiple comparison tests.
- `power.fisher.test` computes the power of Fisher’s exact test for comparing proportions.
- `sage.test` is a fast approximation to Fisher’s exact test for each tag for comparing two Serial Analysis of Gene Expression (SAGE) libraries.
- `permp` computes p-values for permutation tests when the permutations are randomly drawn.

**Variance Models**

- `mixedModel2`, `mixedModel2Fit` and `glmgam.fit` fit mixed linear models.
- `remlscore` and `remlscoregamma` fit heteroscedastic and varying dispersion models by REML.
- `welding` is an example data set.

**Matrix Computations**

- `matvec` and `vecmat` facilitate multiplying matrices by vectors.

**Author(s)**

Gordon Smyth

---

**Digamma**

**Digamma generalized linear model family**

**Description**

Produces a Digamma generalized linear model family object. The Digamma distribution is the distribution of the unit deviance for a gamma response.

**Usage**

```
Digamma(link = "log")
unitdeviance.digamma(y, mu)
cumulant.digamma(theta)
meanval.digamma(theta)
d2cumulant.digamma(theta)
varfun.digamma(mu)
canoninc.digamma(mu)
```
Digamma

Arguments

- **link**: character string, number or expression specifying the link function. See *quasi* for specification of this argument.
- **y**: numeric vector of (positive) response values
- **mu**: numeric vector of (positive) fitted values
- **theta**: numeric vector of values of the canonical variable, equal to $-1/\phi$ where $\phi$ is the dispersion parameter of the gamma distribution

Details

This family is useful for dispersion modelling with gamma generalized linear models. The Digamma distribution describes the distribution of the unit deviances for a gamma family, in the same way that the gamma distribution itself describes the distribution of the unit deviances for Gaussian or inverse Gaussian families. The Digamma distribution is so named because it is dual to the gamma distribution in the above sense, and because the digamma function appears in its mean function.

Suppose that $y$ follows a gamma distribution with mean $\mu$ and dispersion parameter $\phi$, so the variance of $y$ is $\phi \mu^2$. Write $d(y, \mu)$ for the gamma distribution unit deviance. Then $\text{meanval. digamma}(-1/\phi)$ gives the mean of $d(y, \mu)$ and $2 \times \text{d2cumulant. digamma}(-1/\phi)$ gives the variance.

Value

Digamma produces a glm family object, which is a list of functions and expressions used by glm in its iteratively reweighted least-squares algorithm. See family for details.

The other functions take vector arguments and produce vector values of the same length and called by Digamma. unitdeviance.digamma gives the unit deviances of the family, equal to the squared deviance residuals. cumulant.digamma is the cumulant function. If the dispersion is unity, then successive derivatives of the cumulant function give successive cumulants of the Digamma distribution. meanvalue.digamma gives the first derivative, which is the expected value. d2cumulant.digamma gives the second derivative, which is the variance. canonic.digamma is the inverse of meanvalue.digamma and gives the canonical parameter as a function of the mean parameter. varfun.digamma is the variance function of the Digamma family, the variance as a function of the mean.

Author(s)

Gordon Smyth

References


See Also

quasi.make.link
Examples

# Test for log-linear dispersion trend in gamma regression
y <- rchisq(20, df=1)
x <- 1:20
out.gam <- glm(y~x, family=Gamma(link="log"))
d <- residuals(out.gam)^2
out.dig <- glm(d~x, family=Digamma(link="log"))
summary(out.dig, dispersion=2)

elda

Extreme Limiting Dilution Analysis

Description

Fit single-hit model to a dilution series using complementary log-log binomial regression.

Usage

elda(response, dose, tested=rep(1,length(response)), group=rep(1,length(response)), observed=FALSE, confidence=0.95, test.unit.slope=FALSE)
limdil(response, dose, tested=rep(1,length(response)), group=rep(1,length(response)), observed=FALSE, confidence=0.95, test.unit.slope=FALSE)
eldaOneGroup(response, dose, tested, observed=FALSE, confidence=0.95, tol=1e-8, maxit=100, trace=FALSE)

Arguments

response numeric vector giving number of positive cases out of tested trials. Should take non-negative integer values.
dose numeric vector of expected number of cells in assay. Values must be positive.
tested numeric vector giving number of trials at each dose. Should take integer values.
group vector or factor giving group to which the response belongs.
observed logical, is the actual number of cells observed?
confidence numeric level for confidence interval. Should be strictly between 0 and 1.
test.unit.slope logical, should the adequacy of the single-hit model be tested?
tol convergence tolerance.
maxit maximum number of Newton iterations to perform.
trace logical, if TRUE then interim results are output at each iteration.
Details

elda and limdil are alternative names for the same function. (limdil was the older name before
the 2009 paper by Hu and Smyth.) eldaOneGroup is a lower-level function that does the computa-
tions when there is just one group, using a globally convergent Newton iteration. It is called by the
other functions.

These functions implement maximum likelihood analysis of limiting dilution data using methods
proposed by Hu and Smyth (2009). The functions gracefully accommodate situations where 0% or
100% of the assays give positive results, which is why we call it "extreme" limiting dilution
analysis. The functions provide the ability to test for differences in stem cell frequencies between
groups, and to test goodness of fit in a number of ways. The methodology has been applied to the
analysis of stem cell assays (Shackleton et al, 2006).

The statistical method is explained by Hu and Smyth (2009). A binomial generalized linear model
is fitted for each group with cloglog link and offset log(dose). If observed=FALSE, a classic
Poisson single-hit model is assumed, and the Poisson frequency of the stem cells is the exp of
the intercept. If observed=TRUE, the values of dose are treated as actual cell numbers rather than
expected values. This doesn’t change the generalized linear model fit, but it does change how the
frequencies are extracted from the estimated model coefficient (Hu and Smyth, 2009).

The confidence interval is a Wald confidence interval, unless the responses are all negative or all
positive, in which case Clopper-Pearson intervals are computed.

If group takes several values, then separate confidence intervals are computed for each group. In
this case a likelihood ratio test is conducted for differences in active cell frequencies between the
groups.

These functions compute a number of different tests of goodness of fit. One test is based on the
coefficient for log(dose) in the generalized linear model. The nominal slope is 1. A slope greater
than one suggests a multi-hit model in which two or more cells are synergistically required to
produce a positive response. A slope less than 1 suggests some sort of cell interference. Slopes less
than 1 can also be due to heterogeneity of the stem cell frequency between assays. elda conducts
likelihood ratio and score tests that the slope is equal to one.

Another test is based on the coefficient for dose. This idea is motivated by a suggestion of Gart
and Weiss (1967), who suggest that heterogeneity effects are more likely to be linear in dose than
log(dose). These functions conducts score tests that the coefficient for dose is non-zero. Negative
values for this test suggest heterogeneity.

These functions produce objects of class "limdil". There are print and plot methods for "limdil"
objects.

Value

elda and limdil produce an object of class "limdil". This is a list with the following components:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CI</td>
<td>numeric matrix giving estimated stem cell frequency and lower and upper limits of Wald confidence interval for each group</td>
</tr>
<tr>
<td>test.difference</td>
<td>numeric vector giving chisquare likelihood ratio test statistic and p-value for testing the difference between groups</td>
</tr>
<tr>
<td>test.slope.wald</td>
<td>numeric vector giving wald test statistics and p-value for testing the slope of the offset equal to one</td>
</tr>
</tbody>
</table>
test.slope.lr numeric vector giving chisquare likelihood ratio test statistics and p-value for testing the slope of the offset equal to one
test.slope.score.logdose numeric vector giving score test statistics and p-value for testing multi-hit alternatives
test.slope.score.dose numeric vector giving score test statistics and p-value for testing heterogeneity
response numeric of integer counts of positive cases, out of tested trials
tested numeric vector giving number of trials at each dose
dose numeric vector of expected number of cells in assay
group vector or factor giving group to which the response belongs
num.group number of groups

Author(s)
Yifang Hu and Gordon Smyth

References


See Also
`plot.limdil` and `print.limdil` are methods for `limdil` class objects.

A webpage interface to this function is available at http://bioinf.wehi.edu.au/software/elda.

Examples

# When there is one group
Dose <- c(50,100,200,400,800)
Responses <- c(2,6,9,15,21)
Tested <- c(24,24,24,24,24)
out <- elda(Responses,Dose,Tested,test.unit.slope=TRUE)
out
plot(out)

# When there are four groups
Dose <- c(30000, 20000, 4000, 500, 30000, 20000, 4000, 500, 30000, 20000, 4000, 500, 30000, 20000, 4000, 500)
Responses <- c(2, 3, 2, 1, 6, 5, 6, 1, 2, 3, 4, 2, 6, 6, 6, 1)
Tested <- c(6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6)
Group <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 4)
elda(Responses, Dose, Tested, Group, test.unit.slope=TRUE)

## fitNBP

### Negative Binomial Model for SAGE Libraries with Pearson Estimation of Dispersion

#### Description

Fit a multi-group negative-binomial model to SAGE data, with Pearson estimation of the common overdispersion parameter.

#### Usage

```r
fitNBP(y, group=NULL, lib.size=colSums(y), tol=1e-5, maxit=40, verbose=FALSE)
```

#### Arguments

- **y**: numeric matrix giving counts. Rows correspond to tags (genes) and columns to SAGE libraries.
- **group**: factor indicating which library belongs to each group. If NULL then one group is assumed.
- **lib.size**: vector giving total number of tags in each library.
- **tol**: small positive numeric tolerance to judge convergence
- **maxit**: maximum number of iterations permitted
- **verbose**: logical, if TRUE then iteration progress information is output.

#### Details

The overdispersion parameter is estimated equating the Pearson goodness of fit to its expectation. The variance is assumed to be of the form $\text{Var}(y) = \mu(1 + \phi \mu)$ where $E(y) = \mu$ and $\phi$ is the dispersion parameter. All tags are assumed to share the same dispersion.

For given dispersion, the model for each tag is a negative-binomial generalized linear model with log-link and $\log(\text{lib.size})$ as offset. The coefficient parametrization used is that corresponding to the formula `$\sim 0 + \text{group} + \text{offset}(\log(\text{lib.size}))$.

Except for the dispersion being common rather than gene-wise, the model fitted by this function is equivalent to that proposed by Lu et al (2005). The numeric algorithm used is that of alternating iterations (Smyth, 1996) using Newton’s method as the outer iteration for the dispersion parameter starting at $\phi=0$. This iteration is monotonically convergent for the dispersion.
**forward**

**Description**

Fit a multi-group negative-binomial model to SAGE data, with Pearson estimation of the common overdispersion parameter.

**Usage**

```r
forward(y, x, xkept=NULL, intercept=TRUE, nvar=ncol(x))
```

**Value**

List with components

- `coefficients` numeric matrix of rates for each tag (gene) and each group
- `fitted.values` numeric matrix of fitted values
- `dispersion` estimated dispersion parameter

**Author(s)**

Gordon Smyth

**References**


**See Also**

`sage.test`

**Examples**

```r
# True value for dispersion is 1/size=2/3
# Note the Pearson method tends to under-estimate the dispersion
y <- matrix(rnbinom(10*4,mu=4,size=1.5),10,4)
lib.size <- rep(50000,4)
group <- c(1,1,2,2)
fit <- fitNBP(y,group=group,lib.size=lib.size)
logratio <- fit$coef # not c(-1,1)
```
Arguments

- **y**: numeric response vector.
- **x**: numeric matrix of covariates, candidates to be added to the regression.
- **xkept**: numeric matrix of covariates to be included in the starting regression.
- **intercept**: logical, should an intercept be added to **xkept**?
- **nvar**: integer, number of covariates from **x** to add to the regression.

Details

This function has the advantage that **x** can have many more columns than the length of **y**.

Value

Integer vector of length **nvar**, giving the order in which columns of **x** are added to the regression.

Author(s)

Gordon Smyth

See Also

**step**

Examples

```r
y <- rnorm(10)
x <- matrix(rnorm(10*5),10,5)
forward(y,x)
```

---

**gauss.quad**

Gaussian Quadrature

Description

Calculate nodes and weights for Gaussian quadrature.

Usage

```r
gauss.quad(n,kind="legendre",alpha=0,beta=0)
```

Arguments

- **n**: number of nodes and weights
- **kind**: kind of Gaussian quadrature, one of "legendre", "chebyshev1", "chebyshev2", "hermite", "jacobi" or "laguerre"
- **alpha**: parameter for Jacobi or Laguerre quadrature, must be greater than -1
- **beta**: parameter for Jacobi quadrature, must be greater than -1
Details

The integral from \(a\) to \(b\) of \(w(x)f(x)\) is approximated by \(\sum w_f(x)\) where \(x\) is the vector of nodes and \(w\) is the vector of weights. The approximation is exact if \(f(x)\) is a polynomial of order no more than \(2n-1\). The possible choices for \(w(x)\), \(a\) and \(b\) are as follows:

Legendre quadrature: \(w(x)=1\) on \((-1,1)\).

Chebyshev quadrature of the 1st kind: \(w(x)=1/\sqrt{1-x^2}\) on \((-1,1)\).

Chebyshev quadrature of the 2nd kind: \(w(x)=\sqrt{1-x^2}\) on \((-1,1)\).

Hermite quadrature: \(w(x)=\exp(-x^2)\) on \((-\infty, \infty)\).

Jacobi quadrature: \(w(x)=(1-x)\alpha (1+x)\beta\) on \((-1,1)\). Note that Chebyshev quadrature is a special case of this.

Laguerre quadrature: \(w(x)=x^\alpha \exp(-x)\) on \((0, \infty)\).

The algorithm used to generated the nodes and weights is explained in Golub and Welsch (1969).

Value

A list containing the components

- `nodes`: vector of values at which to evaluate the function
- `weights`: vector of weights to give the function values

Author(s)

Gordon Smyth, using Netlib Fortran code [http://www.netlib.org/go/gaussq.f](http://www.netlib.org/go/gaussq.f), and including a suggestion from Stephane Laurent

References


See Also

gauss.quad.prob, integrate
**Examples**

```r
# mean of gamma distribution with alpha=6
out <- gauss.quad(10,"laguerre",alpha=5)
sum(out$weights * out$nodes) / gamma(6)
```

**Description**

Calculate nodes and weights for Gaussian quadrature in terms of probability distributions.

**Usage**

```r
gauss.quad.prob(n,dist="uniform",l=0,u=1,mu=0,sigma=1,alpha=1,beta=1)
```

**Arguments**

- `n` number of nodes and weights
- `dist` distribution that Gaussian quadrature is based on, one of "uniform", "normal", "beta" or "gamma"
- `l` lower limit of uniform distribution
- `u` upper limit of uniform distribution
- `mu` mean of normal distribution
- `sigma` standard deviation of normal distribution
- `alpha` positive shape parameter for gamma distribution or first shape parameter for beta distribution
- `beta` positive scale parameter for gamma distribution or second shape parameter for beta distribution

**Details**

This is a rewriting and simplification of `gauss.quad` in terms of probability distributions. The probability interpretation is explained by Smyth (1998). For details on the underlying quadrature rules, see `gauss.quad`.

The expected value of $f(X)$ is approximated by \( \sum(w*f(x)) \) where $x$ is the vector of nodes and $w$ is the vector of weights. The approximation is exact if $f(x)$ is a polynomial of order no more than $2n-1$. The possible choices for the distribution of $X$ are as follows:

- Uniform on $(l,u)$.
- Normal with mean $mu$ and standard deviation $sigma$.
- Beta with density $x^{(alpha-1)}(1-x)^{(beta-1)}/B(alpha,beta)$ on $(0,1)$.
- Gamma with density $x^{(alpha-1)}\exp(-x/beta)/beta*alpha/gamma(alpha)$. 
Value

A list containing the components

nodes vector of values at which to evaluate the function
weights vector of weights to give the function values

Author(s)

Gordon Smyth, using Netlib Fortran code http://www.netlib.org/go/gaussq.f, and including corrections suggested by Spencer Graves

References


See Also

gauss.quad, integrate

Examples

# the 4th moment of the standard normal is 3
out <- gauss.quad.prob(10, "normal")
sum(out$weights * out$nodes^4)

# the expected value of log(X) where X is gamma is digamma(alpha)
out <- gauss.quad.prob(32, "gamma", alpha=5)
sum(out$weights * log(out$nodes))

glm.scoretest  Score Test for Adding a Covariate to a GLM

Description

Computes score test statistics (z-statistics) for adding covariates to a generalized linear model.

Usage

glm.scoretest(fit, x2, dispersion=NULL)

Arguments

fit generalized linear model fit object, of class glm.
x2 vector or matrix with each column a covariate to be added.
dispersion the dispersion for the generalized linear model family.
Details

Rao’s score statistic. Is the locally most powerful test for testing vs a one-sided alternative. Asymptotically equivalent to likelihood ratio tests, but convenient for one-sided tests.

This function computes a score test statistics for adding each covariate individually.

The dispersion parameter is treated as for summary.glm. If NULL, the Pearson estimator is used, except for the binomial and Poisson families, for which the dispersion is one.

Value

numeric vector containing the z-statistics, one for each covariate.

Author(s)

Gordon Smyth

References


See Also

glm, add1

Examples

# Pearson's chisquare test for independence
# in a contingency table is a score test.

# First the usual test

y <- c(20,40,40,30)
chisq.test(matrix(y,2,2),correct=FALSE)

# Now same test using glm.scoretest

a <- gl(2,1,4)
b <- gl(2,2,4)
fit <- glm(y~a+b,family=poisson)
x2 <- c(0,0,0,1)
z <- glm.scoretest(fit,x2)
z^2
**glmgam.fit**

*Fit Generalized Linear Model by Fisher Scoring with Levenberg Damping*

**Description**

Fit a generalized linear model with secure convergence. Provided for gamma glm with identity links or negative binomial glm with log-links.

**Usage**

```r
glmgam.fit(x, y, coef.start=NULL, tol=1e-6, maxit=50, trace=FALSE)
glmnb.fit(X, y, dispersion, weights=NULL, offset=0, coef.start=NULL, start.method="mean", tol=1e-6, maxit=50, trace=FALSE)
```

**Arguments**

- `x`: design matrix, assumed to be of full column rank. Missing values not allowed.
- `y`: numeric vector of responses. Negative or missing values not allowed.
- `dispersion`: numeric vector of over-dispersion parameters for negative binomial. If of length 1, then same over-dispersion is assumed for all observations.
- `weights`: numeric vector of positive weights, defaults to all one.
- `offset`: offset vector for linear model
- `coef.start`: numeric vector of starting values for the regression coefficients
- `start.method`: method used to find starting values, possible values are "mean" and "log(y)"
- `tol`: small positive numeric value giving convergence tolerance
- `maxit`: maximum number of iterations allowed
- `trace`: logical value. If TRUE then output diagnostic information at each iteration.

**Details**

These functions implement a modified Fisher scoring algorithm for generalized linear models, similar to the Levenberg-Marquardt algorithm for nonlinear least squares. The Levenberg-Marquardt modification checks for a reduction in the deviance at each step, and avoids the possibility of divergence. The result is a very secure algorithm that converges for almost all datasets.

`glmgam.fit` is in principle similar to `glm.fit(X, y, family=Gamma(link="identity"))` but with much more secure convergence. This function is used by `mixedModel2Fit`.

`glmnb.fit` is in principle similar to `glm.fit(X, y, family=negative.binomial(link="log", theta=1/dispersion))` but with more secure convergence.
Value

List with the following components:

- **coefficients**: numeric vector of regression coefficients
- **fitted**: numeric vector of fitted values
- **deviance**: residual deviance
- **iter**: number of iterations used to convergence. If convergence was not achieved then `iter` is set to `maxit+1`.

Author(s)

Gordon Smyth and Yunshun Chen

Examples

```r
y <- rgamma(10, shape=5)
X <- cbind(1:10)
fit <- glmgam.fit(X, y, trace=TRUE)
```

——

growthcurve  

**Compare Groups of Growth Curves**

Description

Do all pairwise comparisons between groups of growth curves using a permutation test.

Usage

```r
compareGrowthCurves(group, y, levels=NULL, nsim=100, fun=meanT, times=NULL, verbose=TRUE, adjust="holm")
compareTwoGrowthCurves(group, y, nsim=100, fun=meanT)
plotGrowthCurves(group, y, levels=sort(unique(group)), times=NULL, col=NULL,...)
```

Arguments

- **group**: vector or factor indicating group membership. Missing values are allowed in `compareGrowthCurves` but not in `compareTwoGrowthCurves`.
- **y**: matrix of response values with rows for individuals and columns for times. The number of rows must agree with the length of `group`. Missing values are allowed.
- **levels**: a character vector containing the identifiers of the groups to be compared. By default all groups with two or more members will be compared.
- **nsim**: number of permutations to estimate p-values.
- **fun**: a function defining the statistic used to measure the distance between two groups of growth curves. Defaults to `meanT`.  

——
times a numeric vector containing the column numbers on which the groups should be compared. By default all the columns are used.

verbose should progress results be printed?

adjust method used to adjust for multiple testing, see p.adjust.

col vector of colors corresponding to distinct groups

Details

compareTwoGrowthCurves performs a permutation test of the difference between two groups of growth curves. compareGrowthCurves does all pairwise comparisons between two or more groups of growth curves. Accurate p-values can be obtained by setting nsim to some large value, nsim=10000 say.

Value

compareTwoGrowthCurves returns a list with two components, stat and p.value, containing the observed statistics and the estimated p-value. compareGrowthCurves returns a data frame with components

Group1 name of first group in a comparison
Group2 name of second group in a comparison
Stat observed value of the statistic
P.Value estimated p-value
adj.P.Value p-value adjusted for multiple testing

Author(s)

Gordon Smyth

References


See Also

meanT.compareGrowthCurves, compareTwoGrowthCurves

Examples

# A example with only one time
data(PlantGrowth)
compareGrowthCurves(PlantGrowth$group, as.matrix(PlantGrowth$weight))

# Can make p-values more accurate by nsim=10000
hommel.test  

Test Multiple Comparisons Using Hommel’s Method

Description

Given a set of p-values and a test level, returns vector of test results for each hypothesis.

Usage

hommel.test(p, alpha=0.05)

Arguments

p  numeric vector of p-values
alpha numeric value, desired significance level

Details

This function implements the multiple testing procedure of Hommel (1988). Hommel’s method is also implemented as an adjusted p-value method in the function \texttt{p.adjust} but the accept/reject approach used here is faster.

Value

logical vector indicating whether each hypothesis is accepted

Author(s)

Gordon Smyth

References


See Also

\texttt{p.adjust}

Examples

\begin{verbatim}
p <- sort(runif(100))[1:10]
cbind(p,p.adjust(p,"hommel"),hommel.test(p))
\end{verbatim}
**invgauss**

**Inverse Gaussian Distribution**

**Description**

Density, cumulative probability, quantiles and random generation for the inverse Gaussian distribution.

**Usage**

\[
dinvgauss(x, \text{mean}=1, \text{shape}=\text{NULL}, \text{dispersion}=1, \log=\text{FALSE})\\
\text{pinvgauss}(q, \text{mean}=1, \text{shape}=\text{NULL}, \text{dispersion}=1, \text{lower.tail}=\text{TRUE}, \log.p=\text{FALSE})\\
\text{qinvgauss}(p, \text{mean}=1, \text{shape}=\text{NULL}, \text{dispersion}=1, \text{lower.tail}=\text{TRUE}, \log.p=\text{FALSE}, \\
\quad \text{maxit}=200L, \text{tol}=1e-14, \text{trace}=\text{FALSE})\\
\text{rinvgauss}(n, \text{mean}=1, \text{shape}=\text{NULL}, \text{dispersion}=1)\\
\]

**Arguments**

- **x, q** vector of quantiles.
- **p** vector of probabilities.
- **n** sample size. If \(\text{length}(n)\) is larger than 1, then \(\text{length}(n)\) random values are returned.
- **mean** vector of (positive) means.
- **shape** vector of (positive) shape parameters.
- **dispersion** vector of (positive) dispersion parameters. Ignored if shape is not \text{NULL}, in which case \text{dispersion}=1/shape.
- **lower.tail** logical; if \text{TRUE}, probabilities are \(P(X<x)\) otherwise \(P(X>x)\).
- **log** logical; if \text{TRUE}, the log-density is returned.
- **log.p** logical; if \text{TRUE}, probabilities are on the log-scale.
- **maxit** maximum number of Newton iterations used to find \(q\).
- **tol** small positive numeric value giving the convergence tolerance for the quantile.
- **trace** logical, if \text{TRUE} then the working estimate for \(q\) from each iteration will be output.

**Details**

The inverse Gaussian distribution takes values on the positive real line. It is somewhat more right skew than the gamma distribution, with variance given by \text{dispersion}*\text{mean}^3. The distribution has applications in reliability and survival analysis and is one of the response distributions used in generalized linear models.

Giner and Smyth (2016) show that the inverse Gaussian distribution is asymptotically equal to an inverse chisquare distribution as the mean becomes large.
The functions provided here implement numeric algorithms developed by Giner and Smyth (2016) that achieve close to full machine accuracy for all possible parameter values. Giner and Smyth (2016) show that the probability calculations provided by these functions are considerably more accurate, and is most cases faster, than previous implementations of inverse Gaussian functions. The improvement in accuracy is most noticeable for extreme probability values and for large parameter values.

The shape and dispersion parameters are alternative parametrizations for the variability, with \( \text{disp} = 1/\text{shape} \). Only one of these two arguments needs to be specified. If both are set, then shape takes precedence.

**Value**

Output values give density \( \text{dinvgauss} \), probability \( \text{pinvgauss} \), quantile \( \text{qinvgauss} \) or random sample \( \text{rinvgauss} \) for the inverse Gaussian distribution with mean \( \text{mean} \) and dispersion \( \text{disp} \). Output is a vector of length equal to the maximum length of any of the arguments \( x, q, \text{mean}, \text{shape} \) or \( \text{disp} \). If the first argument is the longest, then all the attributes of the input argument are preserved on output, for example, a matrix \( x \) will give a matrix on output. Elements of input vectors that are missing will cause the corresponding elements of the result to be missing, as will non-positive values for \( \text{mean} \) or \( \text{disp} \).

**Author(s)**

Gordon Smyth.

Paul Bagshaw (Centre National d’Etudes des Telecommunications, France) and Trevor Park (Department of Statistics, University of Florida) made suggestions on a much earlier versions of the code.

**References**


**Examples**

```r
q <- rinvgauss(10, mean=1, disp=0.5)  # generate vector of 10 random numbers
p <- pinvgauss(q, mean=1, disp=0.5)  # p should be uniformly distributed

# Quantile for small right tail probability:
qinvgauss(1e-20, mean=1.5, disp=0.7, lower.tail=FALSE)

# Same quantile, but represented in terms of left tail probability on log-scale
qinvgauss(-1e-20, mean=1.5, disp=0.7, lower.tail=TRUE, log.p=TRUE)
```
Description

The difference between the log and digamma functions.

Usage

logmdigamma(x)

Arguments

x numeric vector or array of positive values. Negative or zero values will return NA.

Details

digamma(x) is asymptotically equivalent to log(x). logmdigamma(x) computes log(x) - digamma(x) without subtractive cancellation for large x.

Author(s)

Gordon Smyth

References


See Also
digamma

Examples

log(10^15) - digamma(10^15) # returns 0
logmdigamma(10^15) # returns value correct to 15 figures
Multiply a Matrix by a Vector

Description

Multiply the rows or columns of a matrix by the elements of a vector.

Usage

matvec(M, v)
vecmat(v, M)

Arguments

M numeric matrix, or object which can be coerced to a matrix.
v numeric vector, or object which can be coerced to a vector. Length should match the number of columns of M (for matvec) or the number of rows of M (for vecmat)

Details

matvec(M,v) is equivalent to M %*% diag(v) but is faster to execute. Similarly vecmat(v,M) is equivalent to diag(v) %*% M but is faster to execute.

Value

A matrix of the same dimensions as M.

Author(s)

Gordon Smyth

Examples

A <- matrix(1:12,3,4)
A
matvec(A,c(1,2,3,4))
vecmat(c(1,2,3),A)
Description

The mean-t statistic of the distance between two groups of growth curves.

Usage

`meanT(y1, y2)`

Arguments

- `y1`: matrix of response values for the first group, with a row for each individual and a column for each time. Missing values are allowed.
- `y2`: matrix of response values for the second group. Must have the same number of columns as `y1`. Missing values are allowed.

Details

This function computes the pooled two-sample t-statistic between the response values at each time, and returns the mean of these values weighted by the degrees of freedom. This function is used by `compareGrowthCurves`.

Value

numeric vector of length one containing the mean t-statistic.

Author(s)

Gordon Smyth

See Also

`compareGrowthCurves`, `compareTwoGrowthCurves`

Examples

```r
y1 <- matrix(rnorm(4*3),4,3)
y2 <- matrix(rnorm(4*3),4,3)
meanT(y1,y2)

data(PlantGrowth)
compareGrowthCurves(PlantGrowth$group,as.matrix(PlantGrowth$weight))
# Can make p-values more accurate by nsim=10000
```
mixedModel2  

Fit Mixed Linear Model with 2 Error Components

Description

Fits a mixed linear model by REML. The linear model contains one random factor apart from the unit errors.

Usage

mixedModel2(formula, random, weights=NULL, only.varcomp=FALSE, data=list(), subset=NULL, contrasts=NULL, tol=1e-6, maxit=50, trace=FALSE)  
mixedModel2Fit(y, X, Z, w=NULL, only.varcomp=FALSE, tol=1e-6, maxit=50, trace=FALSE)  
randomizedBlock(formula, random, weights=NULL, only.varcomp=FALSE, data=list(), subset=NULL, contrasts=NULL, tol=1e-6, maxit=50, trace=FALSE)  
randomizedBlockFit(y, X, Z, w=NULL, only.varcomp=FALSE, tol=1e-6, maxit=50, trace=FALSE)

Arguments

The arguments formula, weights, data, subset and contrasts have the same meaning as in lm. The arguments y, X and w have the same meaning as in lm.wfit.

- **formula**: vector specifying the fixed model.
- **weights**: vector or factor specifying the blocks corresponding to random effects.
- **only.varcomp**: logical value, if true computation of standard errors and fixed effect coefficients will be skipped.
- **data**: optional vector of prior weights.
- **subset**: optional vector of prior weights.
- **contrasts**: an optional list. See the contrasts.arg argument of model.matrix.default.
- **tol**: small positive numeric tolerance, passed to glmgam.fit.
- **maxit**: maximum number of iterations permitted, passed to glmgam.fit.
- **trace**: logical value, passed to glmgam.fit. If TRUE then working estimates will be printed at each iteration.
- **y**: numeric response vector.
- **X**: numeric design matrix for fixed model.
- **Z**: numeric design matrix for random effects.
- **w**: optional vector of prior weights.
Details

Note that randomizedBlock and mixedModel2 are alternative names for the same function.

This function fits the model \( y = Xb + Zu + e \) where \( b \) is a vector of fixed coefficients and \( u \) is a vector of random effects. Write \( n \) for the length of \( y \) and \( q \) for the length of \( u \). The random effect vector \( u \) is assumed to be normal, mean zero, with covariance matrix \( \sigma^2_u I_q \) while \( e \) is normal, mean zero, with covariance matrix \( \sigma^2 I_n \). If \( Z \) is an indicator matrix, then this model corresponds to a randomized block experiment. The model is fitted using an eigenvalue decomposition which transforms the problem into a Gamma generalized linear model.

Note that the block variance component \( \text{varcomp}[2] \) is not constrained to be non-negative. It may take negative values corresponding to negative intra-block correlations. However the correlation \( \text{varcomp}[2]/\text{sum(\text{varcomp})} \) must lie between -1 and 1.

Missing values in the data are not allowed.

This function is equivalent to \( \text{lme(formula,random=\sim|random)} \), except that the block variance component is not constrained to be non-negative, but is faster and more accurate for small to moderate size data sets. It is slower than \( \text{lme} \) when the number of observations is large.

This function tends to be fast and reliable, compared to competitor functions which fit randomized block models, when then number of observations is small, say no more than 200. However it becomes quadratically slow as the number of observations increases because of the need to do two eigenvalue decompositions of order nearly equal to the number of observations. So it is a good choice when fitting large numbers of small data sets, but not a good choice for fitting large data sets.

Value

A list with the components:

- \( \text{varcomp} \) vector of length two containing the residual and block components of variance.
- \( \text{se.varcomp} \) standard errors for the components of variance.
- \( \text{reml.residuals} \) standardized residuals in the null space of the design matrix.

If \( \text{fixed.estimates=} \text{TRUE} \) then the components from the diagonalized weighted least squares fit are also returned.

Author(s)

Gordon Smyth

References


See Also

- \( \text{gllmgam.fit, lme, lm, lm.fit} \)
Examples

# Compare with first data example from Venable and Ripley (2002),
# Chapter 10, "Linear Models"
library(MASS)
data(petrol)
out <- mixedModel2(Y~SG+VP+V10+EP, random=No, data=petrol)
cbind(varcomp=out$varcomp, se=out$se.varcomp)

mscale  M Scale Estimation

Description

Robust estimation of a scale parameter using Hampel's redescending psi function.

Usage

mscale(u, na.rm=FALSE)

Arguments

u       numeric vector of residuals.
na.rm   logical. Should missing values be removed?

Details

Estimates a scale parameter or standard deviation using an M-estimator with 50% breakdown. This means the estimator is highly robust to outliers. If the input residuals u are a normal sample, then mscale(u) should be equal to the standard deviation.

Value

numeric constant giving the estimated scale.

Author(s)

Gordon Smyth

References


**Examples**

```r
u <- rnorm(100)
sd(u)
mscale(u)
```

**permp**  
*Exact permutation p-values*

**Description**

Calculates exact p-values for permutation tests when permutations are randomly drawn with replacement.

**Usage**

```r
permp(x, nperm, n1, n2, total.nperm=NULL, method="auto", twosided=TRUE)
```

**Arguments**

- `x`: number of permutations that yielded test statistics at least as extreme as the observed data. May be a vector or an array of values.
- `nperm`: total number of permutations performed.
- `n1`: sample size of group 1. Not required if `total.nperm` is supplied.
- `n2`: sample size of group 2. Not required if `total.nperm` is supplied.
- `total.nperm`: total number of permutations allowable from the design of the experiment.
- `method`: character string indicating computation method. Possible values are "exact", "approximate" or "auto".
- `twosided`: logical, is the test two-sided and symmetric between the two groups?

**Details**

This function can be used for calculating exact p-values for permutation tests where permutations are sampled with replacement, using theory and methods developed by Phipson and Smyth (2010). The input values are the total number of permutations done (`nperm`) and the number of these that were considered at least as extreme as the observed data (`x`).

`total.nperm` is the total number of distinct values of the test statistic that are possible. This is generally equal to the number of possible permutations, unless a two-sided test is conducted with equal sample sizes, in which case `total.nperm` is half the number of permutations, because the test statistic must then be symmetric in the two groups. Usually `total.nperm` is computed automatically from `n1` and `n2`, but can also be supplied directly by the user.

When `method="exact"`, the p-values are computed to full machine precision by summing a series terms. When `method="approximate"`, an approximation is used that is faster and uses less memory. If `method="auto"`, the exact calculation is used when `total.nperm` is less than or equal to 10,000 and the approximation is used otherwise.
plot.limdil

Value

vector or array of p-values, of same dimensions as x

Author(s)

Belinda Phipson and Gordon Smyth

References


Examples

```r
x <- 0:5
# Both calls give same results
permp(x=x, nperm=99, n1=6, n2=6)
permp(x=x, nperm=99, total.nperm=462)
```

Description

Plot or print the results of a limiting dilution analysis.

Usage

```r
## S3 method for class 'limdil'
print(x, ...)
## S3 method for class 'limdil'
plot(x, col.group=NULL, cex=1, lwd=1, legend.pos="bottomleft", ...)
```

Arguments

- `x`: object of class `limdil`.
- `col.group`: vector of colors for the groups of the same length as `levels(x$group)`.
- `cex`: relative symbol size
- `lwd`: relative line width
- `legend.pos`: positioning on plot of legend when there are multiple groups
- `...`: other arguments to be passed to `plot` or `print`. Note that `pch` and `lty` are reserved arguments for the plot method.
Details

The print method formats results similarly to a regression or anova summary in R. The plot method produces a plot of a limiting dilution experiment similar to that in Bonnefoix et al (2001). The basic design of the plot was made popular by Lefkovits and Waldmann (1979). The plot shows frequencies and confidence intervals for the multiple groups. A novel feature is that assays with 100% successes are included in the plot and are represented by down-pointing triangles.

Author(s)

Yifang Hu and Gordon Smyth

References


See Also

limdil describes the limdil class.

power.fisher.test

Power of Fisher’s Exact Test for Comparing Proportions

Description

Calculate by simulation the power of Fisher’s exact test for comparing two proportions given two margin counts.

Usage

power.fisher.test(p1, p2, n1, n2, alpha=0.05, nsim=100, alternative="two.sided")

Arguments

p1 first proportion to be compared.
p2 second proportion to be compared.
n1 first sample size.
n2 second sample size.
alpha significance level.
nsim number of data sets to simulate.
alternative indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less".
Details
Computes the power of Fisher’s exact test for testing the null hypothesis that \( p_1 \) equals \( p_2 \) against the alternative that they are not equal.

Value
Estimated power of the test.

Author(s)
Gordon Smyth

See Also
fisher.test, power.t.test

Examples

```r
power.fisher.test(0.5,0.9,20,20) # 70% chance of detecting difference
```

### Description

Compute randomized quantile residuals for generalized linear models.

### Usage

```r
qresiduals(glm.obj, dispersion=NULL) qresid(glm.obj, dispersion=NULL) qres.binom(glm.obj) qres.poi(glm.obj) qres.nbinom(glm.obj) qres.gamma(glm.obj, dispersion=NULL) qres.invgauss(glm.obj, dispersion=NULL) qres.tweedie(glm.obj, dispersion=NULL) qres.default(glm.obj, dispersion=NULL)
```

### Arguments

- **glm.obj**: Object of class glm. The generalized linear model family is assumed to be binomial for qres.binom, poisson for qres.poi, negative binomial for qres.nbinom, Gamma for qres.gamma, inverse Gaussian for qres.invgauss or tweedie for qres.tweedie.
- **dispersion**: a positive real number. Specifies the value of the dispersion parameter for a Gamma or inverse Gaussian generalized linear model if known. If NULL, the dispersion will be estimated by its Pearson estimator.
Details

Quantile residuals are based on the idea of inverting the estimated distribution function for each observation to obtain exactly standard normal residuals. In the case of discrete distributions, such as the binomial and Poisson, some randomization is introduced to produce continuous normal residuals. Quantile residuals are the residuals of choice for generalized linear models in large dispersion situations when the deviance and Pearson residuals can be grossly non-normal. Quantile residuals are the only useful residuals for binomial or Poisson data when the response takes on only a small number of distinct values.

Value

Numeric vector of standard normal quantile residuals.

Author(s)

Gordon Smyth

References


See Also

residuals.glm

Examples

```r
# Poisson example: quantile residuals show no granularity
y <- rpois(20, lambda=4)
x <- 1:20
fit <- glm(y ~ x, family = poisson)
qr <- qresiduals(fit)
qqnorm(qr)
abline(0,1)

# Gamma example:
# Quantile residuals are nearly normal while usual resid are not
y <- rchisq(20, df=1)
fit <- glm(y ~ x, family = Gamma)
qr <- qresiduals(fit, dispersion=2)
qqnorm(qr)
abline(0,1)

# Negative binomial example:
if(require("MASS")) {
  fit <- glm(Days ~ Age, family = negative.binomial(2), data = quine)
summary(qresiduals(fit))
  fit <- glm.nb(Days ~ Age, link = log, data = quine)
summary(qresiduals(fit))
}
```
**Description**

Fits a heteroscedastic regression model using residual maximum likelihood (REML).

**Usage**

```r
corelm(y, X, Z, trace=FALSE, tol=1e-5, maxit=40)
```

**Arguments**

- `y`: numeric vector of responses
- `X`: design matrix for predicting the mean
- `Z`: design matrix for predicting the variance
- `trace`: Logical variable. If true then output diagnostic information at each iteration.
- `tol`: Convergence tolerance
- `maxit`: Maximum number of iterations allowed

**Details**

Write $\mu_i = E(y_i)$ for the expectation of the $i$th response and $s_i = (y_i)$. We assume the heteroscedastic regression model

$$
\mu_i = x_i^T \beta \\
\log(\sigma_i^2) = z_i^T \gamma,
$$

where $x_i$ and $z_i$ are vectors of covariates, and $\beta$ and $\gamma$ are vectors of regression coefficients affecting the mean and variance respectively.

Parameters are estimated by maximizing the REML likelihood using REML scoring as described in Smyth (2002).

**Value**

List with the following components:

- `beta`: vector of regression coefficients for predicting the mean
- `se.beta`: vector of standard errors for beta
- `gamma`: vector of regression coefficients for predicting the variance
- `se.gamma`: vector of standard errors for gamma
- `mu`: estimated means
- `phi`: estimated variances
- `deviance`: minus twice the REML log-likelihood
- `h`: numeric vector of leverages
remlscoregamma

- **cov.beta**: estimated covariance matrix for beta
- **cov.gam**: estimated covariate matrix for gamma
- **iter**: number of iterations used

**Author(s)**

Gordon Smyth

**References**


**Examples**

```r
data(welding)
attach(welding)
y <- Strength
# Reproduce results from Table 1 of Smyth (2002)
X <- cbind(1,(Drying+1)/2,(Material+1)/2)
colnames(X) <- c("1","B","C")
Z <- cbind(1,(Material+1)/2,(Method+1)/2,(Preheating+1)/2)
colnames(Z) <- c("1","C","H","I")
out <- remlscore(y,X,Z)
cbind(Estimate=out$gamma,SE=out$se.gam)
```

---

**remlscoregamma**

*Approximate REML for gamma regression with structured dispersion*

**Description**

Estimates structured dispersion effects using approximate REML with gamma responses.

**Usage**

```r
remlscoregamma(y,X,Z,mlink="log",dlink="log",trace=FALSE,tol=1e-5,maxit=40)
```

**Arguments**

- **y**: numeric vector of responses
- **X**: design matrix for predicting the mean
- **Z**: design matrix for predicting the variance
- **mlink**: character string or numeric value specifying link for mean model
- **dlink**: character string or numeric value specifying link for dispersion model
- **trace**: Logical variable. If true then output diagnostic information at each iteration.
- **tol**: Convergence tolerance
- **maxit**: Maximum number of iterations allowed
Details

Write $\mu_i = E(y_i)$ for the expectation of the $i$th response and $s_i = (y_i)$. We assume the heteroscedastic regression model

$$\mu_i = x_i^T \beta$$

$$\log(\sigma_i^2) = z_i^T \gamma,$$

where $x_i$ and $z_i$ are vectors of covariates, and $\beta$ and $\gamma$ are vectors of regression coefficients affecting the mean and variance respectively.

Parameters are estimated by maximizing the REML likelihood using REML scoring as described in Smyth and Verbyla (2001). See also Smyth and Verbyla (1999a,b).

Value

List with the following components:

- beta: Vector of regression coefficients for predicting the mean
- se.beta: Standard errors for beta
- gamma: Vector of regression coefficients for predicting the variance
- se.gamma: Standard errors for gamma
- mu: Estimated means
- phi: Estimated dispersions
- deviance: Minus twice the REML log-likelihood
- h: Leverages

References


Examples

```r
data(welding)
attach(welding)
y <- Strength
X <- cbind(1,(Drying+1)/2,(Material+1)/2)
colnames(X) <- c("1","B","C")
Z <- cbind(1,(Material+1)/2,(Method+1)/2,(Preheating+1)/2)
colnames(Z) <- c("1","C","H","I")
out <- remlscoregamma(y,X,Z)
```
Description

This function is kept here so as not to break code that depends on it, but has been replaced by `binomTest` in the edgeR Bioconductor package and is no longer updated. It may be removed in a later release of this package.

Computes p-values for differential abundance for each tag between two digital libraries, conditioning on the total count for each tag. The counts in each group as a proportion of the whole are assumed to follow a binomial distribution.

Usage

```r
sage.test(x, y, n1=sum(x), n2=sum(y))
```

Arguments

- `x`: integer vector giving counts in first library. Non-integer values are rounded to the nearest integer.
- `y`: integer vector giving counts in second library. Non-integer values are rounded to the nearest integer.
- `n1`: total number of tags in first library. Non-integer values are rounded to the nearest integer.
- `n2`: total number of tags in second library. Non-integer values are rounded to the nearest integer.

Details

This function was originally written for comparing SAGE libraries (a method for counting the frequency of sequence tags in samples of RNA). It can however be used for comparing any two digital libraries from RNA-Seq, ChIP-Seq or other technologies with respect to technical variation.

An exact two-sided binomial test is computed for each tag. This test is closely related to Fisher’s exact test for 2x2 contingency tables but, unlike Fisher’s test, it conditions on the total number of counts for each tag. The null hypothesis is that the expected counts are in the same proportions as the library sizes, i.e., that the binomial probability for the first library is `n1/(n1+n2)`.

The two-sided rejection region is chosen analogously to Fisher’s test. Specifically, the rejection region consists of those values with smallest probabilities under the null hypothesis.

When the counts are reasonably large, the binomial test, Fisher’s test and Pearson’s chisquare all give the same results. When the counts are smaller, the binomial test is usually to be preferred in this context.

This function is a later version of the earlier `sage.test` function in the sagenhaft Bioconductor package. This function has been made obsolete by `binomTest` in the edgeR package.
tweedie

Value

Numeric vector of p-values.

Author(s)

Gordon Smyth

References

http://en.wikipedia.org/wiki/Binomial_test
http://en.wikipedia.org/wiki/Fisher%27s_exact_test
http://en.wikipedia.org/wiki/RNA-Seq

See Also

binomTest (edgeR package), binom.test (stats package)

Examples

sage.test(c(0,5,10),c(0,30,50),n1=10000,n2=15000)
# Univariate equivalents:
binom.test(5,5+30,p=10000/(10000+15000))$p.value
binom.test(10,10+50,p=10000/(10000+15000))$p.value

---

tweedie  

Tweedie Generalized Linear Models

Description

Produces a generalized linear model family object with any power variance function and any power link. Includes the Gaussian, Poisson, gamma and inverse-Gaussian families as special cases.

Usage

tweedie(var.power=0, link.power=1-var.power)

Arguments

var.power  
index of power variance function

link.power  
index of power link function. link.power=0 produces a log-link. Defaults to the canonical link, which is 1-var.power.
tweedie

Details

This function provides access to a range of generalized linear model response distributions which are not otherwise provided by R, or any other package for that matter. It is also useful for accessing distribution/link combinations which are disallowed by the R glm function.

Let \( \mu_i = E(y_i) \) be the expectation of the \( i \)th response. We assume that

\[
\mu_i^q = x_i^T b, \quad \text{var}(y_i) = \phi \mu_i^p
\]

where \( x_i \) is a vector of covariates and \( b \) is a vector of regression coefficients, for some \( \phi, p \) and \( q \). This family is specified by \text{var.power} = p \text{ and } \text{link.power} = q \text{. A value of zero for } q \text{ is interpreted as } \log(\mu_i) = x_i^T b. \)

The variance power \( p \) characterizes the distribution of the responses \( y \). The following are some special cases:

<table>
<thead>
<tr>
<th>( p )</th>
<th>Response distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>1</td>
<td>Poisson</td>
</tr>
<tr>
<td>(1, 2)</td>
<td>Compound Poisson, non-negative with mass at zero</td>
</tr>
<tr>
<td>2</td>
<td>Gamma</td>
</tr>
<tr>
<td>3</td>
<td>Inverse-Gaussian</td>
</tr>
<tr>
<td>&gt; 2</td>
<td>Stable, with support on the positive reals</td>
</tr>
</tbody>
</table>

The name Tweedie has been associated with this family by Joergensen (1987) in honour of M. C. K. Tweedie.

Value

A family object, which is a list of functions and expressions used by glm and gam in their iteratively reweighted least-squares algorithms. See \texttt{family} and \texttt{glm} in the R base help for details.

Author(s)

Gordon Smyth

References


See Also

glm, family, dtweedie

Examples

```r
y <- rgamma(20, shape=5)
x <- 1:20
# Fit a poisson generalized linear model with identity link
glm(y~x, family=tweedie(var.power=1, link.power=1))

# Fit an inverse-Gaussian glm with log-link
glm(y~x, family=tweedie(var.power=3, link.power=0))
```

welding

Data: Tensile Strength of Welds

Description

This is a highly fractionated two-level factorial design employed as a screening design in an off-line welding experiment performed by the National Railway Corporation of Japan. There were 16 runs and 9 experimental factors. The response variable is the observed tensile strength of the weld, one of several quality characteristics measured. All other variables are at plus and minus levels.

Usage

data(welding)

Format

A data frame containing the following variables. All the explanatory variables are numeric with two levels, -1 and 1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rods</td>
<td>Kind of welding rods</td>
</tr>
<tr>
<td>Drying</td>
<td>Period of drying</td>
</tr>
<tr>
<td>Material</td>
<td>Welded material</td>
</tr>
<tr>
<td>Thickness</td>
<td>Thickness</td>
</tr>
<tr>
<td>Angle</td>
<td>Angle</td>
</tr>
<tr>
<td>Opening</td>
<td>Opening</td>
</tr>
<tr>
<td>Current</td>
<td>Current</td>
</tr>
<tr>
<td>Method</td>
<td>Welding method</td>
</tr>
<tr>
<td>Preheating</td>
<td>Preheating</td>
</tr>
<tr>
<td>Strength</td>
<td>Tensile strength of the weld in kg/mm. The response variable.</td>
</tr>
</tbody>
</table>

Source

http://www.statsci.org/data/general/welding.html
References


Index

+ Topic algebra
  matvec, 22
+ Topic array
  matvec, 22
+ Topic datasets
  welding, 38
+ Topic distribution
  invgauss, 19
+ Topic documentation
  statmod-package, 2
+ Topic htest
  hommel.test, 18
  permp, 27
  power.fisher.test, 29
  sage.test, 35
+ Topic math
  gauss.quad, 10
  gauss.quad.prob, 12
  logmdigamma, 21
+ Topic models
  Digamma, 3
+ Topic regression
  elda, 5
  fitNBP, 8
  forward, 9
  glm.scoretest, 13
  glmgam.fit, 15
  growthcurve, 16
  meanT, 23
  mixedModel2, 24
  plot.limdil, 28
  qresiduals, 30
  remlscore, 32
  remlscoregamma, 33
  tweedie, 36

  add1, 14
  binom.test, 36

  canonic.digamma (Digamma), 3
  compareGrowthCurves, 2, 17, 23
  compareGrowthCurves (growthcurve), 16
  compareTwoGrowthCurves, 17, 23
  compareTwoGrowthCurves (growthcurve), 16
  cumulant.digamma (Digamma), 3
  d2cumulant.digamma (Digamma), 3
  Digamma, 3
dinvgauss, 3
    dinvgauss (invgauss), 19
  dtweedie, 38
  elda, 5
  eldaOneGroup (elda), 5
  family, 38
  fisher.test, 30
  fitNBP, 8
  forward, 9
  gauss.quad, 3, 10, 12, 13
  gauss.quad.prob, 11, 12
  glm, 13, 14, 38
  glm.scoretest, 13
  glmgam.fit, 3, 15, 25
  glmbn.fit (glmgam.fit), 15
  growthcurve, 16
  hommel.test, 3, 18
  integrate, 11, 13
  InverseGaussian (invgauss), 19
  invgauss, 19
  limdil, 2, 29
  limdil (elda), 5
  lm, 25
  lm.fit, 25
  lme, 25
<table>
<thead>
<tr>
<th>Function/Package</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>logmdigamma</td>
<td>21</td>
</tr>
<tr>
<td>meanT</td>
<td>16, 17, 23</td>
</tr>
<tr>
<td>meanval.digamma(Digamma)</td>
<td>3</td>
</tr>
<tr>
<td>mixedModel2</td>
<td>3, 24</td>
</tr>
<tr>
<td>mixedModel2Fit</td>
<td>3, 15</td>
</tr>
<tr>
<td>mixedModel2Fit(mixedModel2)</td>
<td>24</td>
</tr>
<tr>
<td>mscale</td>
<td>26</td>
</tr>
<tr>
<td>p.adjust</td>
<td>18</td>
</tr>
<tr>
<td>permP</td>
<td>3, 27</td>
</tr>
<tr>
<td>pinvgauss</td>
<td>3</td>
</tr>
<tr>
<td>pinvgauss(invgauss)</td>
<td>19</td>
</tr>
<tr>
<td>plot</td>
<td>6</td>
</tr>
<tr>
<td>plot.limdil</td>
<td>7, 28</td>
</tr>
<tr>
<td>plotGrowthCurves(growthcurve)</td>
<td>16</td>
</tr>
<tr>
<td>power.fisher.test</td>
<td>3, 29</td>
</tr>
<tr>
<td>power.t.test</td>
<td>30</td>
</tr>
<tr>
<td>print</td>
<td>6</td>
</tr>
<tr>
<td>print.limdil</td>
<td>7</td>
</tr>
<tr>
<td>print.limdil(plot.limdil)</td>
<td>28</td>
</tr>
<tr>
<td>qinvgauss</td>
<td>3</td>
</tr>
<tr>
<td>qinvgauss(invgauss)</td>
<td>19</td>
</tr>
<tr>
<td>qres.binom(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.default(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.gamma(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.invgauss(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.nbinom(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.pois(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qres.tweedie(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qresid(qresiduals)</td>
<td>30</td>
</tr>
<tr>
<td>qresiduals</td>
<td>2, 30</td>
</tr>
<tr>
<td>quasi</td>
<td>4</td>
</tr>
<tr>
<td>randomizedBlock(mixedModel2)</td>
<td>24</td>
</tr>
<tr>
<td>randomizedBlockFit(mixedModel2)</td>
<td>24</td>
</tr>
<tr>
<td>remlscore</td>
<td>3, 32</td>
</tr>
<tr>
<td>remlscoregamma</td>
<td>3, 33</td>
</tr>
<tr>
<td>residuals.glm</td>
<td>31</td>
</tr>
<tr>
<td>rinvgauss</td>
<td>3</td>
</tr>
<tr>
<td>rinvgauss(invgauss)</td>
<td>19</td>
</tr>
<tr>
<td>sage.test</td>
<td>3, 9, 35</td>
</tr>
<tr>
<td>statmod(statmod-package)</td>
<td>2</td>
</tr>
<tr>
<td>statmod-package</td>
<td>2</td>
</tr>
<tr>
<td>step</td>
<td>10</td>
</tr>
<tr>
<td>summary.glm</td>
<td>14</td>
</tr>
<tr>
<td>tweedie</td>
<td>2, 36</td>
</tr>
<tr>
<td>unitdeviance.digamma(Digamma)</td>
<td>3</td>
</tr>
<tr>
<td>varfun.digamma(Digamma)</td>
<td>3</td>
</tr>
<tr>
<td>vecmat</td>
<td>3</td>
</tr>
<tr>
<td>vecmat(matvec)</td>
<td>22</td>
</tr>
<tr>
<td>welding</td>
<td>3, 38</td>
</tr>
</tbody>
</table>