Package 'jmdem'

October 13, 2022

Type Package

Title Fitting Joint Mean and Dispersion Effects Models

Version 1.0.1

Date 2020-03-03

Author Ka Yui Karl Wu

Maintainer Ka Yui Karl Wu <karlwuky@suss.edu.sg>

Description Joint mean and dispersion effects models fit the mean and dispersion parameters of a response variable by two separate linear models, the mean and dispersion submodels, simultaneously. It also allows the users to choose either the deviance or the Pearson residuals as the response variable of the dispersion submodel. Furthermore, the package provides the possibility to nest the submodels in one another, if one of the parameters has significant explanatory power on the other. Wu & Li (2016) <doi:10.1016/j.csda.2016.04.015>.

License GPL-2

Imports VGAM, statmod

NeedsCompilation no

Repository CRAN

Date/Publication 2020-03-04 06:50:02 UTC

R topics documented:

`anova.jmdem` *Analysis of Deviance for Joint Mean and Dispersion Effect Models Fits*

Description

Compute an analysis of deviance table for one or more double generalised linear model fits.

Usage

```
## S3 method for class 'jmdem'
anova(object, ..., test = NULL, type = c("1", "3"),
      print.results = TRUE)
```

Arguments

`object, ...`	one or several objects of class `jmdem`, typically the result of a call to `jmdem`.
`test`	a character string, (partially) matching one of "`Rao`" or "`Wald`". See `stat.anova.jmdem`.
`type`	a character string or integer, specifying whether a type "`1`" (*sequential*) analysis or a type "`3`" (*partial*) analysis should be conducted. It is only relevant if a single object is specified in `object`. Both numeric and character inputs are allowed. See details for type 1 and type 3 analysis.
`print.results`	logical, `TRUE` if the result table should be printed directly, `FALSE` if the results should be saved in an user-defined object.

Details

Specifying a single object gives a analysis of deviance table for that fit. If type `1` analysis is specified, a sequential analysis will be conducted. That is, the reductions in the residual deviance as each term of the formula is *added* in turn are given in as the rows of a table, plus the residual deviances themselves.

Type `3` analysis gives the reduction in the residual deviance of the fitted model after *removing* each term of the formula individually, that in turn are given as the rows of a table.

If more than one object is specified, the table has a row for the residual degrees of freedom and deviance for each model. For all but the first model, the change in degrees of freedom and deviance is also given. (This only makes statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

The table will optionally contain "`Rao`" or "`Wald`" test statistics (and P values) comparing the model specified in the current row and the row above (type `1`) or the full model (type `3`). Both "`Rao`" and "`Wald`" test statistics are asymptotically chi-square distributed. "`LRT`" (Likelihood ratio test) and "`F`" ((F test) are not included in `anova.jmdem` because the comparison of the deviances of two joint mean and dispersion effects models is questionable, if not even invalid. One important argument is that the dependent variables of two different dispersion submodels given two different mean submodels are not the identical.

Value

An object of class "anova" inheriting from class "data.frame".

If print.results = TRUE,

table.x	the anova table constructed for the mean submodel.
table.z	the anova table constructed for the dispersion submodel.

Warning

The comparison between two or more models will only be valid if they are fitted to the same dataset. This may be a problem if there are missing values and R's default of na.action = na.omit is used, and anova will detect this with an error.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Hastie, T. J. and Pregibon, D. (1992). *Generalized linear models*. Chapter 6 of Statistical Models in S eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

McCullagh P. and Nelder, J.A. (1989) *Generalized Linear Models*. London: Chapman and Hall.

Smyth, G.K. (1989). *Generalized linear models with varying dispersion*. J. R. Statist. Soc. B, **51** (1), 47-60.

Smyth, G.K., Verbyla, A.P. (1996). *A conditional likelihood approach to residual maximum linear estimation in generalized linear models*. J. R. Statist. Soc. B, **58** (3), 565-572.

Smyth, G.K., Verbyla, A.P. (1999). *Adjusted likelihood methods for modelling dispersion in generalized linear models*. Environmetrics, **10**, 695-709.

Wu, K.Y.K., Li, W.K. (2016). *On a dispersion model with Pearson residual responses*. Comput. Statist. Data Anal., **103**, 17-27.

See Also

jmdem, anova

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")
```

```
## Run a partial analysis (type 3) with Wald test
anova(fit, test = "Wald", type = 3)
```

---

jmdem *Fitting Joint Mean and Dispersion Effects Models*

Description

jmdem is used to fit joint mean and dispersion effects models, specified by giving a symbolic description of the linear predictors for the mean and dispersion and a description of the error distribution

Usage

```
jmdem(mformula, dformula, data, mfamily = gaussian, dfamily = Gamma,
      weights, subset, dev.type = c("deviance", "pearson"),
      moffset = NULL, doffset = NULL, mustart = NULL, phistart = NULL,
      betastart = NULL, lambdastart = NULL, hessian = TRUE, na.action,
      grad.func = TRUE, fit.method = "jmdem.fit",
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
      df.adj = FALSE, disp.adj = FALSE, full.loglik = FALSE,
      beta.first = TRUE, prefit = TRUE, mcontrasts = NULL,
      dcontrasts = NULL, control = list(...),
      minv.method = c("solve", "chol2inv", "ginv"), ...)

jmdem.fit(x, y, z = NULL, weights, mfamily = gaussian, dfamily = Gamma,
          mu, phi, beta, lambda, moffset = NULL, doffset = NULL,
          dev.type = c("deviance", "pearson"), hessian = TRUE,
          method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
          disp.adj = FALSE, df.adj = FALSE, full.loglik = FALSE,
          control = list(), mintercept = TRUE, dintercept = TRUE,
          grad.func = TRUE, lower = -Inf, upper = Inf, ...)
```

Arguments

mformula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the *mean* submodel to be fitted. The details of model specification are given under 'Details'.
dformula	a symbolic description of the *dispersion* submodel to be fitted. The details are also given under 'Details'.
data	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which jmdem is called.
mfamily	a description of the error distribution and link function to be used in the *mean* submodel. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)

`dfamily` a description of the error distribution and link function to be used in the *dispersion* submodel. (Also see `family` for details of family functions.)

`weights` an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`dev.type` a specification of the type of residuals to be used as the response of the *dispersion* submodel. The ML estimates of the jmdem are the optima of either the quasi-likelihood function for *deviance residuals*, or the pseudo-likelihood function for *Pearson* residuals.

`moffset` an a priori known component to be included in the linear predictor of the *mean* submodel during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.

`doffset` an a priori known component to be included in the linear predictor of the *dispersion* submodel during fitting. See `model.offset`.

`mustart, mu` a vector of starting values of individual means.

`phistart, phi` a vector of starting values of individual dispersion.

`betastart, beta`
a vector of starting values for the regression parameters of the *mean* submodel.

`lambdastart, lambda`
a vector of starting values for the regression parameters of the *dispersion* submodel.

`hessian` the method used to compute the information matrix. Hessian matrix will be calculated for `"TRUE"`, Fisher matrix for `"FALSE"`.

`na.action` a function which indicates what should happen when the data contain `NA`s. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The 'factory-fresh' default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful.

`grad.func` the gradient function will be included in the optimisation for the `"BFGS"`, `"CG"` and `"L-BFGS-B"` methods for `"TRUE"`. If it is `NULL`, a finite-difference approximation will be used.

For the `"SANN"` method it specifies a function to generate a new candidate point. If it is `NULL` a default Gaussian Markov kernel is used.

`fit.method` the method to be used in fitting the model. The default method `"jmdem.fit"` uses the general-purpose optimisation (`optim`): the alternative "model.frame" returns the model frame and does no fitting.

User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as `jmdem.fit`. If specified as a character string it is looked up from within the *stats* namespace.

`method` the method to be used for the optimisation. See `optim` for details.

df.adj	an adjustment factor for the degrees of freedom (n-p)/n, where n is the number of observations and p is the number of parameters to be estimated in jmdem, will be multiplied to the likelihood function before the optimisation for "TRUE".
disp.adj	an adjustment factor for the dispersion weight will be multiplied to the estimated dispersion parameter during the optimisation for "TRUE". For details, please see McCullagh and Nelder (1989, Ch. 10, P. 362).
full.loglik	the full likelihood function instead of the quasi- or pseudo-likelihood function will be used for the optimisation for TRUE.
beta.first	the mean effects will be estimated (assuming constant sample dispersion) at the initial stage for TRUE. For FALSE, the dispersion effects will be estimated first (assuming constantly zero mean for the whole sample).
prefit	a specfication whether jmdem uses glm to prefit the starting values of the mean and dispersion parameters. For FALSE, the initial parameter values of all the regressors are set to zero and the sample mean and sample dispersion will be used as the starting values of the corresponding submodel intercepts instead. If the submodels have no intercept, all parameters will also be set to zero. The sample mean and sample dispersion will then be used as mustart and phistart in the internal computation (they will not be officially recorded in mustart and phistart in the output object). Defaule value is TRUE.
mcontrasts	an optional list for the mean effect constrasts. See the contrasts.arg of model.matrix.default.
dcontrasts	an optional list for the dispersion effect constrasts. See the contrasts.arg of model.matrix.default.
control	a list of parameters for controlling the fitting process. For jmdem.fit this is passed to jmdem.control.
minv.method	the method used to invert matrices during the estimation process. "solve" gives the solutions of a system of equations, "chol2inv" gives the inverse from Choleski or QR decomposition and "ginv" gives the generalized inverse of a matrix. If none of the methods is specified or if they are specified in a vector such as c("solve", "chol2inv", "ginv"), the matrices will be inverted by the methods in the sequence as given in the vector until it is found.
x, y, z	x is a *mean* submodel's design matrix of dimension n * p, z is a *dispersion* submodel's design matrix of dimension n * k, and y is a vector of observations of length n. If z is NULL, the *dispersion* submodel only contains an intercept.
mintercept	a specification whether the intercept term is included in the *mean* submodel.
dintercept	a specification whether the intercept term is included in the *dispersion* submodel.
lower, upper	bounds on the variables for the "L-BFGS-B" optimisation method.
...	For control: arguments to be used to form the default control argument if it is not supplied directly. For jmdem and jmdem.fit: further arguments passed to or from other methods.

Details

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response.

A terms specification of the form `first + second` indicates all the terms in first together with all the terms in second with any duplicates removed. A specification of the form `first:second` indicates the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification `first * second` indicates the cross of first and second. This is the same as `first + second + first:second`.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

An additional term `response ~ terms + eta` can be added to `dformula` if the *mean* submodel is nested in the *dispersion* submodel in the form such that

$$g(E(y_i)) = \boldsymbol{x}_i\boldsymbol{\beta} = \eta_i, h(\phi) = \boldsymbol{z}_i\boldsymbol{\lambda} + \eta_i\gamma.$$

In the contrary, if the *dispersion* submodel is nested in the *mean* submodel such that

$$g(E(y_i)) = \boldsymbol{x}_i\boldsymbol{\beta} + \delta_i\kappa, h(\phi_i) = \boldsymbol{z}_i\boldsymbol{\lambda} = \delta_i,$$

`mformula` can be specified as `response ~ terms + delta`.

Non-`NULL` weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations. For a binomial GLM prior weights are used to give the number of trials when the response is the proportion of successes: they would rarely be used for a Poisson GLM.

If more than one of `etastart` and `mustart` is specified, the first in the list will be used. It is often advisable to supply starting values for a quasi family, and also for families with unusual links such as gaussian("log").

`glm.fit` is the workhorse function: it is not normally called directly but can be more efficient where the response vector, design matrix and family have already been calculated.

Value

`coefficients`	a named vector of estimated coefficients of both the mean and *dispersion* submodel
`beta`	estimated coefficients of the *mean* submodel
`lambda`	estimated coefficients of the *dispersion* submodel
`residuals`	the *working* residuals, that is the residuals in the final iteration of the `optim` fit. Depending on the type of deviance specified by `dev.type`, `residuals` corresponds to `deviance.residuals` or `pearson.residuals`. Since cases with zero weights are omitted, their working residuals are `NA`.
`deviance.residuals`	the *deviance* residuals resulting from the final iteration of the `optim` fit.
`pearson.residuals`	the *pearson* residuals resulting from the final iteration of the `optim` fit.
`fitted.values`	the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
`dispersion`	the fitted individual dispersion values, obtained by transforming the linear predictors of the *dispersion* submodel by the corresponding inverse of the link function.

`mean.rank` the numeric rank of the fitted *mean* submodel.

`dispersion.rank`
the numeric rank of the fitted *dispersion* submodel.

`rank` the total numeric rank of the fitted model. `mean.rank` and `dispersion.rank` are the corresponding ranks of the fitted *mean* and *dispersion* submodels.

`mean.family` the `family` object used for the *mean* submodel.

`dispersion.family`
the `family` object used for the *dispersion* submodel.

`mean.linear.predictors`
the linear fit on link scale of the *mean* submodel.

`dispersion.linear.predictors`
the linear fit on link scale of the *dispersion* submodel.

`deviance` the residual sum of squares of the complete fitted model.

`individual.loglik`
individual value of the log-likelihood function given the estimated mean and dispersion.

`aic` the *Akaike Information Criterion*, minus twice the maximised log-likelihood plus twice the number of parameters.

`iter` number of iterations needed for the fit.

`weights` the working weights, that is the weights in the final iteration of the `optim` fit.

`prior.weights` the weights initially supplied, a vector of `1`s if none were.

`info.matrix` the information matrix given the estimated model coefficients. The diagonal elements of its inverse are the standard errors of the model parameters.

`df.residual` the residual degrees of freedom of the complete fitted model.

`y` the `y` vector used.

`x` the *mean* submodel design matrix.

`z` the *dispersion* submodel design matrix.

`log.llh` the maximised log-likelihood of the entire sample.

`converged` logical. Was the `optim` algorithm judged to have converged?

`gradient` logical. Was the gradient function included in the `optim` algorithm?

`deviance.type` the type of redidual deviance specified, it is either "`deviance`" or "`pearson`".

`information.type`
the type of information matrix specified, it is either "`Hessian`" or "`Fisher`".

`dispersion.adjustment`
logical. Was the dispersion parameter adjusted by an adjustment factor during the optimisation?

`df.adjustment` logical. Was the likelihood function adjusted by the degrees of freedom adjustment factor?

`optim.method` the name of the method used in `optim`.

`control` the value of the control argument used.

`data` the evaluated dataset specified in the `data` argument.

`mean.model` the model frame of the *mean* submodel.

`dispersion.model`
the model frame of the *dispersion* submodel.

`call` the matched call.

`mean.formula` the formula of the *mean* submodel supplied.

`dispersion.formula`
the formula of the *dispersion* submodel supplied.

`fit.method` the name of the fit function used, currently always "`jmdem.fit`".

`mean.offset` the offset vector used in the *mean* submodel.

`dispersion.offset`
the offset vector used in the *dispersion* submodel.

`dispersion.deviance`
the deviance sum of squares of the *dispersion* submodel.

`dispersion.df.residual`
the residual degrees of freedom of the *dispersion* submodel.

`null.deviance` the residual sum of squares of the complete null model.

`df.null` the residual degrees of freedom for the complete null model.

`dispersion.null.deviance`
the residual sum of squares of the dispersion null submodel.

`dispersion.df.null`
the residual degrees of freedom for the dispersion null submodel.

`beta.null` the estimated coefficients of the mean null submodel.

`lambda.null` the estimated coefficients of the dispersion null submodel.

`dispersion.null`
the estimated dispersion of the complete null model.

`residuals.null` the residuals of the complete null model.

`mustart` the vector of starting values for individual means used.

`phistart` the vector of starting values for individual dispersion used.

`betastart` the vector of starting values for the *mean* submodel parameters used.

`lambdastart` the vector of starting values for the *dispersion* submodel parameters used.

`mean.terms` the terms object used for the *mean* submodel.

`dispersion.terms`
the terms object used for the *dispersion* submodel.

`xlevels` a record of the levels of the factors used in fitting the *mean* submodel.

`zlevels` a record of the levels of the factors used in fitting the *dispersion* submodel.

`mean.contrasts` the contrasts used for the *mean* submodel.

`dispersion.contrasts`
the contrasts used for the *dispersion* submodel.

`na.action` information returned by model.frame on the special handling of `NA`s.

`init.mean.fit` the initial values of the *mean* submodel coefficients, linear predictors and fitted values.

`init.dispersion.fit`
: the initial values of the *dispersion* submodel coefficients, linear predictors and fitted values.

`matrix.inverse.method`
: information returned on the method used for inverting matrices during optimisation.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Carroll, R.J., Ruppert, D. (1988). *Transformation and Weighting in Regression*. London: Chapman and Hall.

Cordeiro, M.G., Simas, A.B. (2009). *The distribution of pearson residuals in generalized linear models*. Comput. Statist. Data Anal., **53**, 3397-3411.

McCullagh, P. (1983). *Quasi-likelihood functions*. Annals of Statistics **11** (1), 59-67.

McCullagh P. and Nelder, J.A. (1989) *Generalized Linear Models*. London: Chapman and Hall.

Nash, J.C. (1990). *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation*. Adam Hilger.

Nelder, J.A., Lee, Y., Bergman, B., Hynen, A., Huele, A.F., Engel, J. (1998). *Joint modelling of mean and dispersion*. Technometrics, **40** (2), 168-175.

Nelder, J.A., Pregibon, D. (1987). *An extended quasi-likelihood function*. Biometrika, **74** (2), 221-232.

Nocedal, J., Wright, S.J. (1999). *Numerical Optimization*. Springer.

Smyth, G.K. (1989). *Generalized linear models with varying dispersion*. J. R. Statist. Soc. B, **51** (1), 47-60.

Smyth, G.K., Verbyla, A.P. (1996). *A conditional likelihood approach to residual maximum linear estimation in generalized linear models*. J. R. Statist. Soc. B, **58** (3), 565-572.

Smyth, G.K., Verbyla, A.P. (1999). *Adjusted likelihood methods for modelling dispersion in generalized linear models*. Environmetrics, **10**, 695-709.

Wedderburn, R. (1974). *Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method*. Biometrika, **61** (3), 439-447.

Wu, K.Y.K., Li, W.K. (2016). *On a dispersion model with Pearson residual responses*. Comput. Statist. Data Anal., **103**, 17-27.

See Also

`anova.jmdem`, `summary.jmdem`, etc. for `jmdem` methods, and the generic functions `effects`, `fitted.values`, and `residuals`.

Examples

```
## Fit poisson counts by unnested mean and dispersion submodels.
## Use log-links for both submodels. Set dispersion fitting based
## on deviance residuals. Use conjugate gradient (CG) as
## optimisation method.
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

## Fit Gaussian responses by nesting dispersion submodel in the mean
## submodel. Use default link for both submodels. Set dispersion fitting
## based on pearson residuals. Use quasi-Newton (BFGS) as optimisation
## method. Adjust degrees of freedom for the likelihood function.
MyData <- simdata.jmdem.sim(mformula = y ~ x + delta, dformula = ~ z,
                            mfamily = gaussian(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4, 1),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x + delta, dformula = ~ z, data = MyData,
             mfamily = gaussian, dfamily = Gamma, dev.type = "pearson",
             method = "BFGS", df.adj = TRUE)
```

`jmdem.control` *Auxiliary for Controlling JMDEM Fitting*

Description

Auxiliary function for `jmdem` fitting. Typically only used internally by `jmdem.fit`, but may be used to construct a `control` argument to either function.

Usage

```
jmdem.control(maxit = 100, epsilon = 1e-8, prefit.trace = FALSE,
              fit.trace = FALSE, null.approx = 1e-8, trace = 0,
              fnscale = -1, parscale = 1, ndeps = 0.001,
              abstol = -Inf, reltol = sqrt(.Machine$double.eps),
              alpha = 1, beta = 0.5, gamma = 2, REPORT = 10,
              type = 1, lmm = 5, factr = 1e+07, pgtol = 0,
              temp = 10, tmax = 10)
```

Arguments

maxit	integer giving the maximal number of optimisation iterations.
epsilon	positive convergence tolerance ϵ; the iterations converge when $\|dev - dev_{old}\|/(\|dev\| + 0.1) < \epsilon$.
prefit.trace	logical indicating if output should be produced for each iteration in the prefit process.
fit.trace	logical indicating if output should be produced for each iteration in the jmdem.fit process.
null.approx	approximisation of zeros to avoid estimation abortion in the case of $log(0)$ or $1/0$. The following control arguments are used by optim. Please refer to optim for details
trace	non-negative integer. If positive, tracing information on the progress of the optimisation is produced. Higher values may produce more tracing information: for method "L-BFGS-B" there are six levels of tracing.
fnscale	An overall scaling to be applied to the value of fn and gr during optimisation. If negative, turns the problem into a maximisation problem. Optimisation is performed on fn(par)/fnscale.
parscale	A vector of scaling values for the parameters. Optimisation is performed on par/parscale and these should be comparable in the sense that a unit change in any element produces about a unit change in the scaled value. Not used (nor needed) for method = "Brent".
ndeps	A vector of step sizes for the finite-difference approximation to the gradient, on par/parscale scale. Defaults to 1e-3.
abstol	The absolute convergence tolerance. Only useful for non-negative functions, as a tolerance for reaching zero.
reltol	Relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of reltol * (abs(val) + reltol) at a step. Defaults to sqrt(.Machine$double.eps), typically about 1e-8.
alpha, beta, gamma	Scaling parameters for the "Nelder-Mead" method. alpha is the reflection factor (default 1.0), beta the contraction factor (0.5) and gamma the expansion factor (2.0).
REPORT	The frequency of reports for the "BFGS", "L-BFGS-B" and "SANN" methods if control$trace is positive. Defaults to every 10 iterations for "BFGS" and "L-BFGS-B", or every 100 temperatures for "SANN".
type	for the conjugate-gradients ("CG") method. Takes value 1 for the Fletcher-Reeves update, 2 for Polak-Ribiere and 3 for Beale-Sorenson.
lmm	is an integer giving the number of BFGS updates retained in the "L-BFGS-B" method, It defaults to 5.
factr	controls the convergence of the "L-BFGS-B" method. Convergence occurs when the reduction in the objective is within this factor of the machine tolerance. Default is 1e7, that is a tolerance of about 1e-8.

pgtol	helps control the convergence of the "L-BFGS-B" method. It is a tolerance on the projected gradient in the current search direction. This defaults to zero, when the check is suppressed.
tmax	controls the "SANN" method. It is the starting temperature for the cooling schedule. Defaults to 10.
temp	is the number of function evaluations at each temperature for the "SANN" method. Defaults to 10.

Details

The control argument of jmdem is by default passed to the control argument of jmdem.fit, which uses its elements as arguments to jmdem.control: the latter provides defaults and sanity checking.

When trace is true, calls to cat produce the output for each iteration. Hence, options(digits = *) can be used to increase the precision, see the example.

Value

A list with components named as the arguments.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Belisle, C.J.P. (1992). *Convergence theorems for a class of simulated annealing algorithms on Rd*. Journal of Applied Probability, **29**, 885-895.

Byrd, R. H., Lu, P., Nocedal, J. and Zhu, C. (1995). *A limited memory algorithm for bound constrained optimisation*. SIAM Journal on Scientific Computing, **16**, 1190-1208.

Fletcher, R. and Reeves, C.M. (1964). *Function minimization by conjugate gradients*. Computer Journal, **7**, 148-154.

Hastie, T. J. and Pregibon, D. (1992). *Generalized linear models*. Chapter 6 of Statistical Models in S eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

Nash, J.C. (1990). *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation*. Adam Hilger.

Nelder, J.A., Mead, R. (1965). *A simplex algorithm for function minimization*. Computer Journal, **7**, 308-313.

Nocedal, J., Wright, S.J. (1999). *Numerical Optimisation*. Springer.

Smyth, G.K. (1989). *Generalised linear models with varying dispersion*. J. R. Statist. Soc. B, **51** (1), 47-60.

Smyth, G.K., Verbyla, A.P. (1999). *Adjusted likelihood methods for modelling dispersion in generalised linear models*. Environmetrics, **10**, 695-709.

Wu, K.Y.K., Li, W.K. (2016). *On a dispersion model with Pearson residual responses*. Comput. Statist. Data Anal., **103**, 17-27.

See Also

jmdem.fit, the fitting procedure used by jmdem.

Examples

```
## Example in jmdem(...). Limit maximum iteration number to 20 and
## trace the deviance development in the fitting process
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ s,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ s, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG",
             control = list(maxit = 20, fit.trace = TRUE))

## Change to a small convergence tolerance and trace the optimisation
## process in optim
jmdem.control(list(epsilon = 1e-14, trace = 1))
```

---

jmdem.sim *Simulate joint mean and dispersion effects models fits*

Description

Simulate iterative jmdem fits on user-defined model settings

Usage

```
jmdem.sim(mformula = "y ~ 1 + x", dformula = "~ 1 + z", data = NULL,
          beta.true, lambda.true, mfamily = gaussian,
          dfamily = Gamma, dev.type = c("deviance", "pearson"),
          x.str = list(type = "numeric", random.func = "runif", param = list()),
          z.str = list(type = "numeric", random.func = "runif", param = list()),
          n = NULL, simnum = NULL, trace = FALSE, asymp.test = FALSE,
          weights = NULL, moffset = NULL, doffset = NULL,
          mustart = NULL, phistart = NULL, betastart = NULL,
          lambdastart = NULL, hessian = TRUE, na.action,
          grad.func = TRUE, fit.method = "jmdem.fit",
          method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
          df.adj = FALSE, disp.adj = FALSE, full.loglik = FALSE,
          mcontrasts = NULL, dcontrasts = NULL, beta.first = TRUE,
          prefit = TRUE, control = list(...),
          minv.method = c("solve", "chol2inv", "ginv"), ...)

simdata.jmdem.sim(mformula = "y ~ 1 + x", dformula = "~ 1 + z", beta.true, lambda.true,
```

```
              x.str = list(type = "numeric", random.func = "runif", param = list()),
              z.str = list(type = "numeric", random.func = "runif", param = list()),
                mfamily = gaussian, dfamily = Gamma, weights = NULL, n, simnum = 1,
                    moffset = NULL, doffset = NULL)

getdata.jmdem.sim(object)
```

Arguments

mformula	the user-defined true mean submodel, expressed in form of an object of class "formula". The number of regressors and their interactions can be specified here, but not their true parameter values.
dformula	the user-defined true dispersion submodel. See mformula.
data	an optional data frame or list of several data frames. If no data are provided, jmdem.sim will generate its own data for simulation by simdata.jmdem.sim.
beta.true	a vector of the true parameter values of the mean submodel. The number of elements in beta.true must be identical with the number of parameters to be estimated in mformula, including the intercept if there exists one in the model.
lambda.true	a vector of the true parameter values of the dispersion submodel. The number of elements in lambda.true must be identical with the number of parameters to be estimated in dformula, including the intercept if there exists one in the model.
mfamily	a description of the error distribution and link function to be used in the *mean* submodel. This can be a character string naming a family function, a family function or the result of a call to a family function. (See family for details of family functions.)
dfamily	a description of the error distribution and link function to be used in the *dispersion* submodel. (Also see family for details of family functions.)
dev.type	a specification of the type of residuals to be used as the response of the *dispersion* submodel. The ML estimates of the jmdem are the optima of either the quasi-likelihood function for *deviance residuals*, or the pseudo-likelihood function for *Pearson* residuals.
x.str	a list of user-specified structure for the generation of the mean submodel design matrix, including the type (numeric, character, logical etc.), an r function (random.func) to generate the values of the regressors and the corresponding parameters (param) to be passed on to (random.func). Note that all parameters that belong to the same random.func must be put in a list(...). See details.
z.str	a list of user-specified structure for the generation of the dispersion submodel design matrix, including the type (numeric, character, logical etc.), an r function (random.func) to generate the values of the regressors and the corresponding parameters (param) to be passed on to (random.func). Note that all parameters that belong to the same random.func must be put in a list(...). See details.
n	a numeric value specifying the sample size in each simulation.
simnum	a numeric value specifying the number of simulations.
trace	a specification whether the estimated coefficients should be printed to screen after each simulation.

asymp.test	a specification whether the Rao's score and Wald tests should be conducted for each simulation.
...	for control: arguments to be used to form the default control argument if it is not supplied directly. For jmdem.sim: further arguments passed to or from other methods. The following arguments are used for JMDEM fitting. See jmdem for details.
weights	an optional vector of 'prior weights' to be used in the fitting process. Should be NULL or -a numeric vector.
moffset	an a priori known component to be included in the linear predictor of the *mean* submodel during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
doffset	an a priori known component to be included in the linear predictor of the *dispersion* submodel during fitting. See model.offset.
mustart	a vector of starting values of individual means.
phistart	a vector of starting values of individual dispersion.
betastart	a vector of starting values for the regression parameters of the *mean* submodel.
lambdastart	a vector of starting values for the regression parameters of the *dispersion* submodel.
hessian	the method used to compute the information matrix. Hessian matrix will be calculated for "TRUE", Fisher matrix for "FALSE".
na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
grad.func	the gradient function will be included in the optimisation for the "BFGS", "CG" and "L-BFGS-B" methods for "TRUE". If it is NULL, a finite-difference approximation will be used. For the "SANN" method it specifies a function to generate a new candidate point. If it is NULL a default Gaussian Markov kernel is used.
fit.method	the method to be used in fitting the model. The default method "jmdem.fit" uses the general-purpose optimisation (optim): the alternative "model.frame" returns the model frame and does no fitting. User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as jmdem.fit. If specified as a character string it is looked up from within the *stats* namespace.
method	the method to be used for the optimisation. See optim for details.
df.adj	an adjustment factor for the degrees of freedom (n-p)/n, where n is the number of observations and p is the number of parameters to be estimated in jmdem, will be multiplied to the likelihood function before the optimisation for "TRUE".
disp.adj	an adjustment factor for the dispersion weight will be multiplied to the estimated dispersion parameter during the optimisation for "TRUE". For details, please see McCullagh and Nelder (1989, Ch. 10, P. 362).

full.loglik	the full likelihood function instead of the quasi- or pseudo-likelihood function will be used for the optimisation for TRUE.
mcontrasts	an optional list for the mean effect constrasts. See the contrasts.arg of model.matrix.default.
dcontrasts	an optional list for the dispersion effect constrasts. See the contrasts.arg of model.matrix.default.
beta.first	the mean effects will be estimated (assuming constant sample dispersion) at the initial stage for TRUE. For FALSE, the dispersion effects will be estimated first (assuming constantly zero mean for the whole sample).
prefit	a specfication whether jmdem uses glm to prefit the starting values of the mean and dispersion parameters. For FALSE, the initial parameter values of all the regressors are set to zero and the sample mean and sample dispersion will be used as the starting values of the corresponding submodel intercepts instead. If the submodels have no intercept, all parameters will also be set to zero. The sample mean and sample dispersion will then be used as mustart and phistart in the internal computation (they will not be officially recorded in mustart and phistart in the output object). Defaule value is TRUE.
control	a list of parameters for controlling the fitting process. For jmdem.fit this is passed to jmdem.control.
minv.method	the method used to invert matrices during the estimation process. "solve" gives the solutions of a system of equations, "chol2inv" gives the inverse from Choleski or QR decomposition and "ginv" gives the generalised inverse of a matrix. If none of the methods is specified or if they are specified in a vector such as c("solve", "chol2inv", "ginv"), the matrices will be inverted by the methods in the sequence as given in the vector until it is found.
object	one or several objects of class jmdem.sim, typically the result of a call to jmdem.sim.

Details

jmdem.sim simulates the fitting of datasets in which the regressors of the mean and dispersion submodels are generated according to the specification given in x.str and z.str. The response variable will be then generated according to the distribution specified in mfamily with linear predictor of the mean given by mformula and the linear predictor of the dispersion given by dformula.

The specifications in x.str and z.str are rather flexible if more than one independent variables are included in any of the submodels. For instance, if one of the two independent variables of the mean submodel is numeric generated from the normal distribution of mean 0 and standard deviation 1, and the other one is a 4-level factor $0, 1, 2, 3$ generated from the uniform distribution, then they can be specified in a vector using c(...), such as: x.str = list(type = c("numeric", "factor"), random.func = c("rnorm", "runif"), param = c(list(mean = 0, sd = 1), list(min = 0, max = 3))).

Note that the higher the number of simulations specified in simnum, the more stabilised are the aggregated simulation results. The larger the sample size in each simulation, the less fluctuated are the estimated results among the simulations.

Users gain simdata.jmdem.sim higher control on the simulation by generating a number of datasets upon their own settings first, and not running jmdem.sim at the same time. By taking these steps, users also have the flexiblility to edit the datasets according their own individual requirements, before calling them in jmdem.sim.

Users can also extract the datasets used in jmdem.sim by getdata.jmdem.sim. This function is useful if the datasets are generated in jmdem.sim where users do not have access prior to the simulations.

getdata.jmdem.sim and simdata.jmdem.sim can also be useful if the users would like to conduct various simulations with different jmdem settings on the same data.

Value

An object of class jmdem.sim contains of a list of jmdem fits with full model information. That means, each element of the jmdem.sim object contains the same attributes as a jmdem object. See *values* of jmdem for details.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

See Also

jmdem, summary.jmdem.sim

Examples

```
## Run 10 JMDEM simulations with samples of size 50. The response
## variable is Gaussian with mean beta_0 + beta_1 * x and variance
## log(sigma^2) = lambda_0 + lambda_1 * z. The observations of
## the predictor x should be random numbers generated from the normal
## distribution with mean 0 and standard deviation 2. The observations
## of z are factors with three levels between 0 and 2, generated from
## the uniform distribution. The true values of the mean submodel's
## intercept and slope are 1.5 and 4, as well as 2.5, 3 and -0.2 for
## the dispersion submodel's intercept and slope.
sim <- jmdem.sim(mformula = y ~ x, dformula = ~ z, beta.first = TRUE,
                 mfamily = gaussian, dfamily = Gamma(link = "log"),
                 x.str = list(type = "numeric", random.func = "rnorm",
                              param = list(mean = 0, sd = 2)),
                 z.str = list(type = "factor", random.func = "runif",
                              param = list(min = 0, max = 2)),
                 beta.true = c(1.5, 4), lambda.true = c(2.5, 3, -0.2),
                 grad.func = TRUE, method = "BFGS", n = 50,
                 simnum = 10)
```

---

jmdem.summaries *Accessing Joint Mean and Dispersion Effect Model Fits*

Description

These functions are all methods for class jmdem or summary.jmdem objects.

Usage

```
## S3 method for class 'jmdem'
formula(x, submodel = c("both", "mean", "dispersion"), ...)

## S3 method for class 'jmdem'
family(object, submodel = c("both", "mean", "dispersion"), ...)

## S3 method for class 'jmdem'
residuals(object, type = c("deviance", "pearson", "working",
                           "response", "partial"), ...)
```

Arguments

x, object	the function family accesses the family objects which are stored within objects created by jmdem.
submodel	character. The family of the specified submodel. For both, the families of the mean and dispersion submodels will be return in a list of 2 elements.
type	character. For residuals, the type of residuals which should be returned. The alternatives are: "deviance" (default), "pearson", "working", "response", and "partial".
...	further arguments passed to methods.

Details

family is a generic function with methods for class "jmdem". See family for details.

Here formula is referred to the case that it is called on a fitted jmdem model object. The default first, depending on the specified submodel argument, looks for a "mean.formula" and/or "dispersion.formula" component of the jmdem object (and evaluates it), then a "mean.terms" and/or "dispersion.terms" component, then a mformula and/or dformula parameter of the call (and evaluates its value) and finally a "formula" attribute.

The references define the types of residuals: Davison & Snell is a good reference for the usages of each.

The partial residuals are a matrix of working residuals, with each column formed by omitting a term from the model.

How residuals treats cases with missing values in the original fit is determined by the na.action argument of that fit. If na.action = na.omit omitted cases will not appear in the residuals, whereas if na.action = na.exclude they will appear, with residual value NA. See also naresid.

For fits done with y = FALSE the response values are computed from other components.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Cox, D. R. and Snell, E. J. (1981). *Applied Statistics; Principles and Examples*. London: Chapman and Hall.

Chambers, J. M. and Hastie, T. J. (1992) *Statistical Models in S*. Wadsworth & Brooks/Cole.

Davison, A. C. and Snell, E. J. (1991). *Residuals and diagnostics*. In: Statistical Theory and Modelling. In Honour of Sir David Cox, FRS, eds. Hinkley, D. V., Reid, N. and Snell, E. J., Chapman & Hall.

Dobson, A. J. (1983). *An Introduction to Statistical Modelling*. London: Chapman and Hall.

Hastie, T. J. and Pregibon, D. (1992). *Generalized linear models*. Chapter 6 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

McCullagh P. and Nelder, J. A. (1989). *Generalized Linear Models*. London: Chapman and Hall.

See Also

jmdem, anova.jmdem, coef, deviance, df.residual, effects, fitted, weighted.residuals, residuals, residuals.jmdem, summary.jmdem, weights.

Examples

```
## The jmdem(...) example
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

coef(fit)
plot(resid(fit), fitted(fit))
abline(h = 0, lty = 2, col = "gray")
```

---

model.matrix.jmdem *Construct Design Matrices*

Description

model.matrix creates a design (or model) matrix, e.g., by expanding factors to a set of dummy variables (depending on the contrasts) and expanding interactions similarly.

Usage

```
## S3 method for class 'jmdem'
model.matrix(object, submodel = c("both", "mean", "dispersion"), ...)
```

Arguments

`object`	the function family accesses the family objects which are stored within objects created by `jmdem`.
`submodel`	character. The family of the specified submodel. For `both`, the families of the mean and dispersion submodels will be return in a list of 2 elements.
`...`	further arguments passed to or from other methods.

Details

`model.matrix` creates a design matrix from the description given in `terms(object)`, using the data in `data` which must supply variables with the same names as would be created by a call to `model.frame(object)` or, more precisely, by evaluating `attr(terms(object), "variables")`.

Value

The design matrix for a regression-like model with the specified formula and data.

There is an attribute "`assign`", an integer vector with an entry for each column in the matrix giving the term in the formula which gave rise to the column. Value `0` corresponds to the intercept (if any), and positive values to terms in the order given by the `term.labels` attribute of the `terms` structure corresponding to object.

If there are any factors in terms in the model, there is an attribute "`contrasts`", a named list with an entry for each factor. This specifies the contrasts that would be used in terms in which the factor is coded by contrasts (in some terms dummy coding may be used), either as a character vector naming a function or as a numeric matrix.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Chambers, J. M. (1992). *Data for models*. Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

`model.frame`, `model.extract`, `terms`

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
```

```
            dev.type = "deviance", method = "CG")

## Extract the design matrix of the mean submodel
model.matrix(fit, submodel = "mean")
```

`predict.jmdem` *Predict Method for JMDEM Fits*

Description

Obtains predictions and optionally estimates standard errors of those predictions from a fitted joint mean and dispersion effect model object.

Usage

```
## S3 method for class 'jmdem'
predict(object, newdata = NULL, type = c("link", "response"),
        se.fit = FALSE, na.action = na.pass, ...)
```

Arguments

`object`	a fitted object of class inheriting from "`jmdem`".
`newdata`	optionally, a data frame in which to look for variables with which to predict. If omitted, the fitted linear predictors are used.
`type`	the type of prediction required. The default is on the scale of the linear predictors; the alternative "`response`" is on the scale of the response variable. Thus for a default binomial model the default predictions are of log-odds (probabilities on logit scale) and `type` = "`response`" gives the predicted probabilities.
`se.fit`	logical switch indicating if standard errors are required.
`na.action`	function determining what should be done with missing values in `newdata`. The default is to predict `NA`.
`...`	further arguments passed to or from other methods.

Details

If `newdata` is omitted the predictions are based on the data used for the fit. In that case how cases with missing values in the original fit is determined by the `na.action` argument of that fit. If `na.action` = `na.omit` omitted cases will not appear in the residuals, whereas if `na.action` = `na.exclude` they will appear (in predictions and standard errors), with residual value `NA`. See also `napredict`.

Value

If `se.fit` = `FALSE`, a vector or matrix of predictions.

If `se.fit` = `TRUE`, a list with components

`fit`	Predictions, as for se.fit = FALSE.
`se.fit`	Estimated standard errors.

Note

Variables are first looked for in newdata and then searched for in the usual way (which will include the environment of the formula used in the fit). A warning will be given if the variables found are not of the same length as those in newdata if it was supplied.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

See Also

jmdem

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

## Predict on the scale of the response variable with standard errors.
predict(fit, type = "response", se.fit = TRUE)

## Predict based on a new observation on the scale of the linear
## predictors with standard errors.
predict(fit, newdata = data.frame(x = -1.5, z = 100), se.fit = TRUE)
```

---

score.jmdem, wald.jmdem

Asymptotic tests for fits of joint mean and dispersion effects models

---

Description

Computes a score (Rao) or Wald chi-squared test statistics for comparing two jmdem models.

Usage

```
score.jmdem(object, ...)

wald.jmdem(object, ...)
```

Arguments

`object`	a model or `list` of two or more models fitted by `jmdem` to be tested. Pairwise tests will be conducted.
`...`	a `list` of two or more fitted models to be tested.

Details

Given a vector of model coefficients of length p, $\Theta = (\theta_1, \ldots, \theta_q, \theta_{q+1}, \ldots, \theta_p)^T$, the score and Wald tests are usually used to test the null hypothesis against an alternative

$$H_0 : \theta_{q+1} = \ldots = \theta_p = 0 vs. H_0 not true$$

Thus, they are asymptotic tests on the explanatory power of one or more regressors. And the result of the score and Wald tests only makes sense if the models involved are nested, i.e. all coefficients of a "smaller" (null, restricted) model are included in a "bigger" (alternative, unrestricted) model.

The main difference between the score and Wald tests is that the score test only requires the knowledge of the fitted coefficients of the "small" model. The Wald test, on the other hand, only need the estimates of the "bigger" model. Nevertheless, these tests are asymptotically equivalent, i.e. for large samples, the test statistics of these tests on the same set of models should be very close.

The key assumption is that the coefficient estimates asymptotically follow a (multivariate) normal distribution with mean and variance equal to the model parameters and their variance-covariance matrix.

`score.jmdem` and `wald.jmdem` extract the fitted coefficients and their variance-covariance matrix from the model objects, and evaluate the test statistics subsequently. So it is not necessary to specify the coefficients and variance-covariance matrix in the function arguments.

`score.jmdem` and `wald.jmdem` only return the test statistics. They are asymptotically chi-square distributed with $p - q$ degrees of freedom.

Value

`score.jmdem` and `wald.jmdem` return a column matrix containing the test statistics of the pairwise comparisons of all models given by the user in `object` and `...`.

Note

The score test is sometimes also called the Rao's score test or Lagrange multiplier (LM) test in different literatures.

Normally, asymptotic tests include likelihood ratio (LR), Rao's score and Wald tests. The likelihood ratio test is omitted here because the comparison of the deviances of two joint mean and dispersion effects models is questionable, if not even invalid. One important argument is that the dependent variables of two different dispersion submodels given two different mean submodels are not the identical.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Engle, R.F. (1983). *Wald, Likelihood Ratio, and Lagrange Multiplier Tests in Econometrics*. In Intriligator, M. D.; Griliches, Z. *Handbook of Econometrics*. **II**. Elsevier. pp. 796-801.

McCullagh P. and Nelder, J.A. (1989) *Generalized Linear Models*. London: Chapman and Hall.

Wu, K.Y.K., Li, W.K. (2016). *On a dispersion model with Pearson residual responses*. Comput. Statist. Data Anal., **103**, 17-27.

See Also

anova.jmdem, anova, jmdem

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x + delta, dformula = ~ z,
                            mfamily = gaussian(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4, 1),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x + delta, dformula = ~ z, data = MyData,
             mfamily = gaussian, dfamily = Gamma, dev.type = "pearson",
             method = "BFGS", df.adj = TRUE)

fit.1 <- update(fit, mformula = . ~ . - delta)

fit.2 <- update(fit.1, mformula = . ~ . - x)

## conduct a Wald tests
wald.jmdem(fit, fit.1, fit.2)

## should deliver the same results as above
wald.jmdem(object = list(fit, fit.1, fit.2))

## conduct the score test and compute the p-value directly.
raotest <- score.jmdem(fit, fit.2)
pchisq(raotest, df = abs(df.residual(fit) - df.residual(fit.2)),
       lower.tail = FALSE)
```

---

stat.anova.jmdem *JMDEM Anova Statistics*

Description

This is a utility function, used in jmdem method for anova(..., test != NULL) and should not be used by the average user.

Usage

```
stat.anova.jmdem(table, test = c("Rao", "Wald"))
```

Arguments

table	numeric matrix as results from anova.jmdem(..., test = NULL, print.results = FALSE) saved as the attributes table.x or table.z.
test	a character string, partially matching one of "Rao" or "Wald".

Value

A matrix which is the original table, augmented by a column of test statistics, depending on the test argument.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Hastie, T. J. and Pregibon, D. (1992). *Generalized linear models*. Chapter 6 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

anova.jmdem

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

stat.anova.jmdem(anova(fit, test = "Rao", print.results = FALSE)$table.x)
```

summary.jmdem *Summarising Joint Mean and Dispersion Effects Model Fits*

Description

These functions are all methods for class jmdem or summary.jmdem objects.

Usage

```
## S3 method for class 'jmdem'
summary(object, correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'summary.jmdem'
print(x, digits = max(3L, getOption("digits") - 3L),
      scientific = FALSE, symbolic.cor = x$symbolic.cor,
      signif.stars = getOption("show.signif.stars"), ...)
```

Arguments

object	an object of class "jmdem", usually, a result of a call to jmdem.
x	an object of class "summary.jmdem", usually, a result of a call to summary.jmdem.
correlation	logical; if TRUE, the correlation matrix of the estimated parameters is returned and printed.
digits	the number of significant digits to use when printing.
scientific	logical; if TRUE, scientific notation is used when printing.
symbolic.cor	logical. If TRUE, print the correlations in a symbolic form (see symnum) rather than as numbers.
signif.stars	logical. If TRUE, 'significance stars' are printed for each coefficient.
...	further arguments passed to or from other methods.

Details

print.summary.jmdem tries to be smart about formatting the coefficients, standard errors, etc. and additionally gives 'significance stars' if signif.stars is TRUE. The coefficients, mean.coefficients and dispersion.coefficients components of the result give the estimated coefficients and their estimated standard errors, together with their ratio. This third column is labelled t-ratio and a fourth column gives the two-tailed p-value corresponding to the t-ratio based on a Student t distribution.

Aliased coefficients are omitted in the returned object but restored by the print method.

Correlations are printed to the same decimal places specified in digits (or symbolically): to see the actual correlations print summary(object)$correlation directly.

For more details, see summary.glm.

Value

call	the component from object.
mean.family	the component from object.
dispersion.family	the component from object.
deviance	the component from object.
mean.terms	the component from object.
dispersion.terms	the component from object.
aic	the component from object.
mean.contrasts	the component from object.
dispersion.contrasts	the component from object.
df.residual	the component from object.
null.deviance	the component from object.
df.null	the component from object.
information.type	the component from object.
iter	the component from object.
mean.na.action	the component from object.
dispersion.na.action	the component from object.
deviance.resid	the deviance residuals.
pearson.resid	the pearson residuals.
resid	the working residuals depends on the setting of deviance.type.
coefficients	the matrix of coefficients, standard errors, z-values and p-values. Aliased coefficients are omitted.
mean.coefficients	the matrix of coefficients, standard errors, z-values and p-values of the mean submodel.
dispersion.coefficients	the matrix of coefficients, standard errors, z-values and p-values of the dispersion submodel.
deviance.type	the type of redidual deviance specified, it is either "deviance" or "pearson".
aliased	named logical vector showing if the original coefficients are aliased.
df	a 3-vector of the rank of the model and the number of residual degrees of freedom, plus number of coefficients (including aliased ones).
covariance	the estimated covariance matrix of the estimated coefficients.
digits	the number of significant digits to use when printing.
scientific	logical value of using scientific notation when printing.
covmat.method	named method used to invert the covariance matrix.
correlation	(only if correlation is true.) The estimated correlations of the estimated coefficients.
symbolic.cor	(only if correlation is true.) The value of the argument symbolic.cor.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

See Also

jmdem, summary

Examples

```
## Example in jmdem(...)
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

## Summarise fit with correlation matrix
summary(fit, correlation = TRUE, digits = 4)
```

---

summary.jmdem.sim *Summarising JMDEM Simulations*

Description

These functions are all methods for class jmdem.sim or summary.jmdem.sim objects.

Usage

```
## S3 method for class 'jmdem.sim'
summary(object, digits = max(3L, getOption("digits") - 3L),
        scientific = FALSE, pvalue = 0.05,
        minv.method = c("solve", "chol2inv", "ginv"),
        other.call = FALSE, details = FALSE, ...)

## S3 method for class 'summary.jmdem.sim'
print(x, digits = max(3L, getOption("digits") - 3L), scientific = FALSE,
      pvalue = 0.05, signif.stars = getOption("show.signif.stars"),
      other.call = FALSE, details = FALSE, ...)
```

Arguments

object	an object of class "jmdem.sim", usually, a result of a call to jmdem.sim.
x	an object of class "summary.jmdem.sim", usually, a result of a call to summary.jmdem.sim.
digits	the number of significant digits to use when printing.

scientific	logical; if TRUE, scientific notation is used when printing.
pvalue	a value between 0 and 1. It is used to compute the coverage proportion of the true parameter values by the simulated fits.
minv.method	the method used to invert matrices during the estimation process. "solve" gives the solutions of a system of equations, "chol2inv" gives the inverse from Choleski or QR decomposition and "ginv" gives the generalised inverse of a matrix. If none of the methods is specified or if they are specified in a vector such as c("solve", "chol2inv", "ginv"), the matrices will be inverted by the methods in the sequence as given in the vector until it is found.
signif.stars	logical. If TRUE, 'significance stars' are printed for each coefficient.
other.call	logical. If true, the rest of simulation call (i.e. without the mean and dispersion submodel formulas, families, true values) will be shown.
details	logical. If true, coefficients, standard errors, true parameter coverage (TRUE/FALSE) and asymptotic test statistics of each simulation will be listed.
...	further arguments passed to or from other methods.

Details

The arithmetric mean of the coefficients, standard errors and coverage by the confidence intervals estimated in all simulations will be listed in a table. A detail listing of each simulation's results can be provided if required by details = TRUE. The summary also includes the averages of the Rao's score and Wald test statistics of all simulation fits.

print.summary.jmdem.sim tries to be smart about formatting the coefficients, standard errors, etc according the number of significant digits (default of user-specified) or the usage of scientific notation or not.

Value

digits	the number of significant digits to use when printing.
scientific	logical value of using scientific notation when printing.
details	logical value of printing details of each simulation.
other.call	logical value of printing other parameters of the simulation call.
pvalue	numeric value between 0 and 1 used for the computation of the true parameter coverage.
beta.true	user-defined vector containing the true parameter values of the mean submodel.
lambda.true	user-defined vector containing the true parameter values of the dispersion submodel.
simcall	the component from object.
mformula	the component from object.
dformula	the component from object.
mfamily	the component from object.
dfamily	the component from object.
coefficients	mean and dispersion submodel parameter coefficients fitted in each simulation saved in a data.frame.

`stderr` standard erros of all mean and dispersion submodel parameter coefficients estimated in each simulation saved in a `data.frame`.

`iterations` a vectror containing the running numbers of each simulation.

`confint` confidence intervals of all mean and dispersion submodel parameter coefficients estimated in each simulation saved in a `data.frame`.

`coverage` the coverage of all true submodel parameters by the confidence intervals estimated in each simulation saved in a `data.frame`.

`asymp.test` Rao's score and Wald test statistics of each simulation saved in a `data.frame`.

`average.summary`
Arithmetric means of the coefficients, standard errors, confidence interval coverage estimated in all simulations saved in a `data.frame`.

`average.asymp.test`
(Arithmetric means of the Rao's score and Wald test statistics estimated in all simulations saved in a `data.frame`.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

See Also

`jmdem.sim`, `jmdem`, `summary`

Examples

```
## Example in jmdem.sim(...)
sim <- jmdem.sim(mformula = y ~ x, dformula = ~ z, beta.first = TRUE,
                 mfamily = gaussian, dfamily = Gamma(link = "log"),
                 x.str = list(type = "numeric", random.func = "rnorm",
                              param = list(mean = 0, sd = 2)),
                 z.str = list(type = "factor", random.func = "runif",
                              param = list(min = 0, max = 2)),
                 beta.true = c(1.5, 4), lambda.true = c(2.5, 3, -0.2),
                 grad.func = TRUE, method = "BFGS", n = 50,
                 simnum = 10)

## Summarise simulation
summary(sim, details = FALSE, other.call = TRUE)
```

---

update.jmdem *Update and Re-fit a JMDEM Call*

Description

`update` will update and (by default) re-fit a model. It does this by extracting the call stored in the object, updating the call and (by default) evaluating that call. Sometimes it is useful to call update with only one argument, for example if the data frame has been corrected.

Usage

```
## S3 method for class 'jmdem'
update(object, mformula, dformula, ...)
```

Arguments

object	An existing fit from a jmdem model function
mformula	Changes to the formula of the mean submodel - see update.formula for details.
dformula	Changes to the formula of the dispersion submodel - see update.formula for details.
...	Additional arguments to the call, or arguments with changed values. Use name = NULL to remove the argument name.

Author(s)

Karl Wu Ka Yui (karlwuky@suss.edu.sg)

References

Chambers, J. M. (1992). *Linear models*. Chapter 4 of *Statistical Models* in S eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

update.formula

Examples

```
## Example in jmdem(...): Update the dispersion fitting based on Pearson
## residuals and change from Nelder-Mead to BFGS as optimisation method.
MyData <- simdata.jmdem.sim(mformula = y ~ x, dformula = ~ z,
                            mfamily = poisson(),
                            dfamily = Gamma(link = "log"),
                            beta.true = c(0.5, 4),
                            lambda.true = c(2.5, 3), n = 100)

fit <- jmdem(mformula = y ~ x, dformula = ~ z, data = MyData,
             mfamily = poisson, dfamily = Gamma(link = "log"),
             dev.type = "deviance", method = "CG")

update(fit, dev.type = "pearson", method = "BFGS")
```

Index