Package ‘mixreg’

October 14, 2021

Version 2.0-10

Date 2021-10-14

Title Functions to Fit Mixtures of Regressions

Author Rolf Turner <`r.turner@auckland.ac.nz`>

Maintainer Rolf Turner <`r.turner@auckland.ac.nz`>

Depends R (>= 3.5.0)

Description Fits mixtures of (possibly multivariate) regressions
(which has been described as doing ANCOVA when you don't
know the levels). Turner (2000) <doi:10.1111/1467-9876.00198>.

LazyData true

License GPL (>= 2)

URL `http://www.stat.auckland.ac.nz/~rolf/`

NeedsCompilation no

Repository CRAN

Date/Publication 2021-10-14 04:10:02 UTC

R topics documented:

`aphids` *Data on the rate of infection of tobacco plants by a virus spread by aphids.*

Description

The data set `aphids` is a data frame with 51 rows and 2 columns. The rows correspond to 51 independent experiments in which varying numbers of aphids were released in a flight chamber containing 12 infected and 69 healthy tobacco plants. The resulting number of infected plants (amongst those previously healthy) was recorded.

Usage

```
aphids
```

Format

The data frame `aphids` contains the following columns:

`aphRel` The number of aphids released in the flight chamber in each instance.

`plntsInf` The resulting number (out of a possible 69) of infected plants

Determination of Infection Count

After 24 hours from the time that the aphids were released, the flight chamber was fumigated to kill the aphids, and the previously healthy plants were moved to a greenhouse and monitored to detect symptoms of infection. The number of plants displaying such symptoms was recorded.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

Source

These data appear courtesy of Gilles Boiteau and George Tai of the Potato Research Centre, Agriculture and Agri-Food Canada, Fredericton, New Brunswick. Any published work using these data should cite the paper given in the **References**.

References

Boiteau, G., M. Singh, R. P. Singh, G. C. C. Tai, and T. R. Turner (1998). Rate of spread of PVY-n by alate Myzus persicae (Sulzer) from infected to healthy plants under laboratory conditions. Potato Research, vol. 41, pp. 335 – 344.

cband	*Calculate confidence and prediction bands for mixtures of one-variable regressions.*

Description

Produces confidence and prediction bands, two-sided or upper or lower, for the lines fitted in a model consisting of a mixture of one-variable regressions.

Usage

```
cband(object, alpha=0.05, MC=FALSE, xlen=100, plot=FALSE,...)
```

Arguments

`object`	Object describing the fitted mixture of regressions, as returned by `mixreg()`.
`alpha`	One minus the confidence level for the confidence and prediction bands; e.g. alpha = 0.05 for 95% confidence.
`MC`	Logical scalar; should the covariance matrix of the parameter estimates be calculated by a Monte Carlo procedure? If the covariance matrix is extracted from `object`, the `"MC"` attribute of this matrix is checked. If argument `MC` disagrees with this attribute then the covariance matrix is re-calculated using the appropriate procedure.
`xlen`	The number of points to be plotted in the band envelopes. The x-values of the points will be equispaced from the minimum to the maximum of the predictor variable.
`plot`	Logical scalar; should a plot of the fitted model and confidence and prediction bands be produced immediately?
`...`	Extra arguments to be passed to `covMixMC()` if the covariance matrix needs to be (re-) calculated and `MC` is `TRUE`.

Details

The prediction bands are conditional in that the associated probability is conditional upon the associated observation being generated by the relevant component of the mixture.

The covariance matrix need to construct the confidence and prediction bands is extracted from `object`, given that `object` has an entry named `"covMat"` (i.e. if the call to `mixreg` that produced `object` was made with `covMat=TRUE`). If `object` has no such entry then `covMix()` is called to produce the covariance matrix. (Such a call may take a bit of time.)

Value

An object of class `"cband"`, consisting of a list with entries:

`theta`	The parameter list from object (as returned by mixreg).
`intercept`	The logical value from object indicating whether intercepts were fitted.

`x`	The predictor for the model (extracted from the `data` entry of `object`).
`y`	The response for the model (extracted from the `data` entry of `object`).
`xf`	The equispaced sequence of values, extending from `min(x)` to `max(x)`, at which the values of the band envelopes were calculated.
`bnds`	A list with one entry for each component of the mixture. Each entry is a matrix with 8 columns (lower and upper confidence and prediction bounds for one-sided intervals and lower and upper confidence and prediction bounds for two-sided interval).
`alpha`	Numeric scalar; the `alpha` argument.
`varnms`	Character vector of length two providing the names of the predictor (first entry) and of the response (second entry).

If `plot` is `TRUE` the value is returned invisibly.

Side Effects

If `plot` is `TRUE` a plot of the fit and the confidence and prediction bands is produced in whatever device is currently open or on screen if no device is open.

Warning

If `MC` is `FALSE` then an error may be thrown if the observed Fisher information (the inverse of which is use as the estimated covariance matrix) is singular. Note that if `MC` is FALSE then any call to `covMix()` is made with `useMC="no"`.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

See Also

`ncMcTest()`, `covMix()`, `mixreg()`, `plot.cband()`, `residuals.mixreg()`, `plot.mixresid()`, `qqMix()`,

Examples

```
# Aphids.
    thStrt <- list(list(beta=c(3.0,0.1),sigsq=16,lambda=0.5),
                   list(beta=c(0.0,0.0),sigsq=16,lambda=0.5))
    fit    <- mixreg(plntsInf~aphRel,ncomp=2,thetaStart=thStrt,
                     covMat=TRUE,data=aphids)
    cbds   <- cband(fit,plot=TRUE)
    plot(cbds) # Same plot as was produced by call to cband().
# Kilns.
   thStrt <- list(
                    list(beta=c(26.07,48808),sigsq=1.1573,lambda=0.33333333),
                    list(beta=c(23.48,32387),sigsq=1.8730,lambda=0.33333333),
                    list(beta=c(-0.0597,20760),sigsq=0.2478,lambda=0.33333333)
                  )
    fit    <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
    ## Not run:  # Takes too long.
        res    <- residuals(fit,std=TRUE)
        qqMix(res) # No way are these residuals Gaussian!
        cbdsG  <- cband(fit)
        cbdsMC <- cband(fit,MC=TRUE)
        plot(cbdsG)
        plot(cbdsMC)
    # Same-same, despite the lack of Gaussianity!

## End(Not run)
```

`covMix` *Calculate the covariance matrix of the parameter estimates for a mixture of regressions.*

Description

Produces an estimate of the covariance matrix of the parameter estimates for a fitted mixture of linear regressions, by inverting the observed Fisher information matrix.

Usage

```
covMix(object,useMC=c("ifNec","no"),...)
covMixMC(object,nsim=100,progRep=TRUE,seed=NULL,...)
```

Arguments

`object`	Object describing the fitted mixture of regressions, as returned by `mixreg()`.
`useMC`	Text string specifying whether to call upon a Monte Carlo procedure if there are problems with the "analytic" procedure (i.e. if the calculated observed Fisher information is singular) or to simply give up and throw an error.
`nsim`	Positive integer scalar specifying how many simulated samples to generate for the purpose of calculating the Monte Carlo estimate of the covariance matrix.
`progRep`	Logical scalar; should nominal "progress reports" be issued during the simulation?
`seed`	Integer scalar. The seed for the random number generator used to produce random samples from which to calculate the Monte Carlo based estimate of the covariance matrix. If this argument is not supplied, then it is randomly sampled from `1:1e5`.
`...`	Optional arguments `semiPar` and `conditional` to be passed on to `rmixreg()` by `covMixMC()`.

Details

If different variances are allowed amongst the components (i.e. if `object$eqVar` is `FALSE`) then the parameters are taken in the order beta.1, sigsq.1, lambda.1, . . . , beta.K, sigsq.K for a K component model — lambda.K is redundant and hence omitted. If equal variances are assumed, the parameters are taken in the order beta.1, lambda.1, . . . , beta.K, sigsq.

In the foregoing beta refers to the linear coefficients, sigsq to the variance or variances, and lambda to the mixing probabilities.

Value

The estimated covariance matrix. If the Monte Carlo method was applied then this matrix has an attribute `"seed"`. This attribute will be the value of the `seed` argument if this was supplied, otherwise it is the randomly generated replacement for this argument.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

T. A. Louis (1982). Finding the observed information matrix when using the EM algorithm. *Journal of the Royal Statistical Society*, series B **44** pp. 226 – 233.

See Also

`ncMcTest()`, `cband()` `mixreg()`, `plot.cband()`, `plot.mixresid()`, `qqMix()`, `residuals.mixreg()`

Examples

```
# Aphids.
    fita   <- mixreg(plntsInf~aphRel,ncomp=2,seed=42,data=aphids)
    cMafi  <- covMix(fita)
    ## Not run:
        cMaMC  <- covMixMC(fita)

## End(Not run)
# Kilns.
   thStrt <- list(
                   list(beta=c(26.07,48808),sigsq=1.1573,lambda=0.33333333),
                   list(beta=c(23.48,32387),sigsq=1.8730,lambda=0.33333333),
                   list(beta=c(-0.0597,20760),sigsq=0.2478,lambda=0.33333333)
                 )
    fitk   <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
    ## Not run:
        cMkfi  <- covMix(fitk)
        cMkMC  <- covMixMC(fitk)
        cMkMCs <- covMixMC(fitk,semiPar=TRUE)
```

```
## End(Not run)
```

`kilns` *The* `kilns` *data sets.*

Description

Data on the rotation times of kilns.

Usage

```
kilnAfull
kilnAsubset
kilnAoneOut
kilnB
```

Format

Each data set is a data frame with columns consisting of observations on variables `x` and `y`:

`x` a numeric vector of *reciprocals* of percentages

`y` a numeric vector of times of a single revolution of a kiln, in seconds

- The data set `kilnAfull` has 3793 observations.
- The data set `kilnAoneOut` has 3792 observations.
- The data set `kilnAsubset` has 92 observations.
- The data set `kilnB` has 3740 observations.

Details

These data consist of observation relating to the rotation times of two kilns, “A” and “B”. They are daily averages observed over 11 years, or 4017 days, from 1 January 2005 to 31 December 2015. The `kilnAsubset` data consist of a small subset of the `kilnAfull` data. The `kilnAoneOut` data set is the same as the `kilnAfull` data set but with one row/observation, number 1171 (which appears to be an outlier in some sense), removed.

The reason that `kilnAfull` and `kilnB` do not contain 4017 observation is that there were a number of missing values in both `x` and `y`. Rows in which either or both `x` and `y` were missing (there were 224 such) were deleted. Likewise 277 rows were deleted in the process of forming `kilnB`.

Plots of the percentages versus times displayed patterns of points with curved structure. Transforming the percentages to their reciprocals changed these patterns to ones that are very close to being straight lines.

The kiln “A” data clearly involve three components. The kiln “B” data involve only two components (likewise clearly discernible).

Source

The data were kindly provided by Petr Pikal (Prerov, Czech Republic).

Examples

```
    fit1 <- mixreg(y~x,data=kilnAfull,ncomp=3,seed=173)
    plot(fit1) # Components 1 and 2 seem to have got swapped and
               # the component 1 (???) line is a bit skew-wiff.
# There's a point that looks to be a bit of an outlier.
# It has been identified to be point 1171.
    with(kilnAfull,text(x[1171],y[1171],labels="1171",
                        adj=-0.3,col="red"))
# Removing this point gives kilnAoneOut.
    fit2 <- mixreg(y~x,data=kilnAoneOut,ncomp=3,seed=173)
    plot(fit2) # Still no good; same as fit1, although the "outlier" is gone.
## Not run:
    vfit <- visualFit(y~x,data=kilnAoneOut,ncomp=3)
    fit3 <- mixreg(y~x,data=kilnAoneOut,ncomp=3,thetaStart=vfit$theta)
    plot(fit3) # Much better.
    chk <- mixreg(y~x,data=kilnAfull,ncomp=3,thetaStart=vfit$theta)
    plot(chk) # No good; same as fit1 and fit2 but without the swapping
              # of components 1 and 2.  It was the outlier that caused the
              # problem, not the random starting values.

## End(Not run)
   thStrt <- list(
                  list(beta=c(26.07,48808),sigsq=1.1573,lambda=0.33333333),
                  list(beta=c(23.48,32387),sigsq=1.8730,lambda=0.33333333),
                  list(beta=c(-0.0597,20760),sigsq=0.2478,lambda=0.33333333)
                )
# Roughly vfit$theta.
    fit3.a <- mixreg(y~x,data=kilnAoneOut,ncomp=3,thetaStart=thStrt)
    plot(fit3.a) # Sames as fit3.
    chk.a <- mixreg(y~x,data=kilnAfull,ncomp=3,thetaStart=thStrt)
    plot(chk.a) # Same as chk.
```

---

mixreg *Fit a mixture of linear regressions.*

Description

Estimates the parameters for a mixture of linear regressions, assuming Gaussian errors, using the EM algorithm.

Usage

```
mixreg(x, y, ncomp = NULL, intercept = TRUE, eqVar = FALSE,
       thetaStart = NULL, itmax = 1000,eps = 1e-06, verb = TRUE,
       digits = 7, maxTry = 5, seed = NULL, data = NULL,
       covMat=FALSE,MC=FALSE,warn=TRUE,...)
```

Arguments

`x`	Either a formula specifying the regression model in question or a matrix of predictors for that regression model. In the latter case `x` should *NOT* include an initial column of 1s. (If an intercept is required, leave `intercept` as `TRUE`!) If there is only one predictor, `x` may be a vector rather than a one-column matrix.
`y`	The vector of responses for the regression models. Not used and **should not be specified** if `x` is a formula.
`ncomp`	The number of components in the mixture. If `thetaStart` (see below) is supplied, then the number of components is determined from this argument and `ncomp` is ignored. If neither `ncomp` nor `thetaStart` is supplied then `ncomp` defaults to 2.
`intercept`	Logical scalar; should the linear regressions should have intercepts fitted? Ignored if `x` is a formula.
`eqVar`	Logical scalar; should the error variance should be the same for all component? (The alternative is that each component should be allowed a different error variance.)
`thetaStart`	Either a list or a matrix providing starting values for the estimation procedure. If it is a list each of its entries is in turn a list with entries `beta` (vector of linear coefficients), `sigsq` (variance) and `lambda` (mixing probability). If it is a matrix it must have `ncomp` rows and `ncoef + 2` columns, where `ncoef` is the number of regression coefficients in the model. The latter quantity is `ncol(x)` if `intercept` is `FALSE` and is `1 + ncol(x)` if `intercept` is `TRUE`. If `eqVar` is `TRUE` then it is sensible to have all the starting values of `sigsq` equal, but this is not strictly necessary. If `thetaStart` is not specified, starting values are produced "randomly". This sometimes works, but sometimes doesn't. The function `visualFit()` provides a convenient means of determining starting values by a "visual" procedure. Note that if `object` is the value returned by `visualFit()` then the appropriate value for `thetaStart` is `object$theta`.
`itmax`	The maximum number of EM steps to be undertaken. If this maximum number of steps is exceeded then a warning is issued, but a fit (probably unreliable) is returned anyway. The returned value will, in this case, have the `converged` component equal to `FALSE`.
`eps`	A value specifying the convergence criterion for the EM algorithm. If the maximum absolute value of the change in the parameters is less than `eps` the algorithm is considered to have converged.
`verb`	Logical argument; if `verb` is `TRUE` then details of the progress of the algorithm are printed out at each EM step.
`digits`	The number of digits to which the details are printed out, when `verb` is `TRUE`.‘
`maxTry`	If the algorithm encounters a singularity in the likelihood (as may possibly occur when eqVar is FALSE) the algorithm is restarted using new (randomly generated) starting values. The restart is attempted a maximum of maxTry times.
`seed`	A numeric scalar used to seed the random number generators if `thetaStart` is not specified (whence starting values are produced randomly). If `seed` is not an integer it gets rounded to the nearest integer (so `seed=pi` has the same impact

	as `seed=3`. If not supplied, `seed` is selected randomly from `1,2,...,1e5`. The seed for random number generation is set to `seed` in `initRand()` before any random number generation is done. The argument `seed` is ignored if `thetaStart` is supplied.
`data`	A list or data frame in which the values of the data `x` and `y` may be stored. If `data` is not `NULL` then `mixreg()` looks for `x` and `y` first in `data` and then in the global environment.
`covMat`	Logical scalar; should the covariance matrix of the parameter estimates be calculated? (This can take some time.)
`MC`	Logical scalar; should the covariance matrix of the parameter estimates be calculated by a Monte Carlo procedure? Ignored if `covMat` is `FALSE`.
`warn`	Logical scalar. Should a warning be issued if the EM algorithm fails to converge in `itmax` iterations? Note that this failure is discernible anyway, from the `converged` component of the returned value, but it's probably best to be alerted to the failure. This argument is present mainly so that it can be set to `FALSE` internally in `ncMcTest()`, where such warnings are redundant.
`...`	Optional arguments which are passed to `covMixMC()` (if `covMat` and `MC` are both `TRUE`. These optional arguments may be `semiPar`, `conditional` or `cMseed`. (Any other argument being given to `mixreg()` will trigger an error.)

Value

A list, of class mixreg, with components

`parmat`	The parameters of the fitted model arranged as a matrix, each row corresponding to one component of the mixture.
`theta`	The parameters of the fitted model as a list, each entry of the list being itself a list (like those in thetaStart) corresponding to one component of the mixture.
`log.like`	The log likelihood of the fitted model, based on Gaussian errors.
`aic`	The Akaike Information Criterion value for the fitted model; aic is equal to -2 * log.like + 2*M where M is the number of parameters in the model.
`intercept`	The value of the `intercept` argument.
`eqVar`	The value of the `eqVar` argument.
`nsteps`	The number of steps the EM algorithm took to converge.
`converged`	Logical scalar indicating whether the algorithm did indeed 'converge or stopped because it reach the itmax EM step.
`data`	A list or data frame providing the data `x` and `y` to which the model was fitted. It may be equal to the value of the `data` argument, or it may have been constructed, in whole or in part, from the `x` and `y` arguments.
`formula`	The formula used by `lm()` in fitting the regression models. Depending on circumstances it may be equal to the argument `fmla`, or it may have been constructed internally by `mixreg()`, in which case it is of the form `y ~ x` or, if `intercept` is `FALSE`, `y ~ x -1` (with, of course, `x` and `y` replaced by the actual names of the objects provided as the corresponding arguments).

The returned value has an attribute `"seed"` specifying the seed for the random number generators that was used in producing random starting values. If `thetaStart` was specified then the attribute `"seed"` is `NA`.

Warning

If the `x` argument is a formula, then specifying `y` is not only meaningless, but will cause a (possibly mystifying) error to be thrown.

Notes

This function (and indeed the entire `mixreg` package) is structured so as to be able to deal with mixtures of regressions involving any number of predictors. However I know of no practical examples in which more than one predictor is involved, and it seems likely that models involving more than one predictor would present substantial difficulties. Some of the functions in this package (`cband()`, `visualFit()`, `plot.mixreg()`, `...`) can cope only with single-predictor models.

The entries `parmat` and `theta` of the returned value contain the same information, presented in a different format.

Even if `eqVar` is `TRUE`, each entry of `theta` still has its own `sigsq` entry. The values of these will all be equal, however, if `eqVar` is `TRUE`.

For *scalar* mixtures, allowing the components to have different variances can have the effect of introducing singularities in the likelihood function. It is not clear to me what the impact of allowing different variances is in the mixtures-of-regressions setting. In respect of the scalar setting, Aitkin and Tunnicliffe Wilson (see **References**) assert that the singularities that may arise "do not cause any computational difficulty in practice". However in the mixtures of regressions setting, I have observed strange anomalies which appear to be alleviated by setting `eqVar=TRUE`. See **Examples**. If results seem to be unsatisfactory, you may be well-advised to try setting `eqVar=TRUE`, to see if that makes an improvement.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

A. P. Dempster, N. M. Laird and D. B. Rubin (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society* series B, **39**, pp. 1 – 22.

M. Aitkin and G. Tunnicliffe Wilson (1980). Mixture models, outliers, and the EM algorithm. *Technometrics* **22**, pp. 325 – 331.

See Also

`ncMcTest()`, `cband()`, `qqMix()`, `residuals.mixreg()`

Examples

```
# Aphids.
    fit1   <- mixreg(aphRel,plntsInf,ncomp=2,seed=42,data=aphids)
    plot(fit1)
    thStrt <- list(list(beta=c(3.0,0.1),sigsq=16,lambda=0.5),
                   list(beta=c(0.0,0.0),sigsq=16,lambda=0.5))
    fit2   <- mixreg(aphRel,plntsInf,ncomp=2,thetaStart=thStrt,data=aphids)
    fit3   <- mixreg(plntsInf ~ aphRel,ncomp=2,thetaStart=thStrt,data=aphids)
# Kilns.
## Not run:
    vfit <- visualFit(y ~ x,ncomp=3, data=kilnAoneOut)
    fit  <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=vfit$theta)

## End(Not run)
    thStrt <- list(
                   list(beta=c(26.07,48808),sigsq=1.1573,lambda=0.33333333),
                   list(beta=c(23.48,32387),sigsq=1.8730,lambda=0.33333333),
                   list(beta=c(-0.0597,20760),sigsq=0.2478,lambda=0.33333333)
                )
    # Roughly the value of vfit$theta.
    fit  <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
    plot(fit)
# Kilns, zero intercept model.
## Not run:
    vfit <- visualFit(y ~ x - 1,ncomp=3, data=kilnAoneOut)
    fit  <- mixreg(y ~ x - 1,ncomp=3,data=kilnAoneOut,thetaStart=vfit$theta)

## End(Not run)
    thStrt <- list(list(beta=50900,sigsq=3.297,lambda=0.33333333),
                  list(beta=33800,sigsq=25.52,lambda=0.33333333),
                  list(beta=20755,sigsq=0.2477,lambda=0.33333333))
    # Roughly the value of vfit$theta.
    fit  <- mixreg(y ~ x - 1,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
    plot(fit) # Yikes!!!  (But a plot of vfit looks practically perfect.)
    fit  <- mixreg(y ~ x - 1,ncomp=3,data=kilnAoneOut,thetaStart=thStrt,eqVar=TRUE)
    plot(fit) # Looks fine.
```

---

ncMcTest *Perform a Monte Carlo test for the number of components in a mixture of regressions.*

Description

Produces `nsim` simulated realizations of the likelihood ratio statistic, either parametrically or semi-parametrically, and calculates the corresponding p-value of the test.

Usage

```
ncMcTest(x, y, data=NULL, ncomp=2, ncincr=1, intercept=TRUE, nsim=99,
         seed=NULL, ts1=NULL, ts2=NULL, semiPar=FALSE,
         conditional=semiPar, verb=FALSE, progRep=TRUE, ...)
```

Arguments

`x`	Either a formula specifying the regression model to be fitted or a predictor or matrix of predictors for the regression model. If `x` is a matrix it should *NOT* include an initial column of `1`s. If an intercept is desired then the `intercept` argument should be left equal to `TRUE`.
`y`	The vector of responses for the regression models. Ignored if `x` is a formula.
`data`	A list or data frame containing the variables in the regression model. Such variables will be looked for first in `data` and then in the global environment.
`ncomp`	The null-hypothesized number of components in the mixture.
`ncincr`	The increment from the null-hypothesized number of components in the mixture to the number under the alternative hypothesis; i.e. the number of components under the alternative hypothesis is `ncomp + ncincr`.
`intercept`	Logical argument indicating whether the regression models in the mixture should have intercept terms. Ignored if `x` is a formula.
`nsim`	The number of simulated replicates of the log likelihood ratio statistic to be produced.
`seed`	Positive integer scalar. The seed for random number generation. If left `NULL` it is obtained by sampling from `1:1e5`.
`ts1`	Starting values for fitting the `ncomp` component model. If ts1 is null, random starting values are used. (This is not recommended.)
`ts2`	Starting values for fitting the `ncomp+nincr` component model. If ts2 is null, random starting values are used. (This is not recommended.)
`semiPar`	Logical scalar; should semi-parametric bootstrapping should be used?
`conditional`	Logical scalar; should the component-selection probabilities be determined conditionally upon the observations?
`verb`	Logical argument indicating whether the fitting processes should be verbose (i.e. whether details should be printed out at each step of the EM algorithm). If TRUE a huge amount of screen output is produced.
`progRep`	Logical argument indicating whether the index, of the simulated statistic just constructed, should be printed out, to give an idea of how the process is progressing.
`...`	Further arguments to be passed to `mixreg()` to control the fitting procedure.

Details

In this context the "parametric" procedure is to simulate data sets by generating data from the fitted `ncomp` model parameters, using Gaussian errors. In contrast, under the semiparametric bootstrapping procedure, the errors are generated by resampling from the residuals. Since at each predictor value there are `ncomp` residuals, one for each component of the model, the errors are selected at random from these `ncomp` possibilities. If the argument `conditional` is `TRUE` then the selection probabilities at this step are the conditional probabilities, of the observation being generated by each component of the model, given that observation. If `conditional` is `FALSE` then these probabilities are the corresponding entries of `lambda` (see **Value**. The residuals are sampled independently in either case. The procedure is termed *semi*-parametric since the sampling probabilities depend on the parameters of the model. Note that it makes no sense to specify `conditional=TRUE` if `semiPar` is `FALSE`. Doing so will generate an error.

It is important to be aware that the test conducted by this function is a *Monte Carlo* test and that the p-value produced is a Monte Carlo p-value. It is consequently an *exact* p-value in a sense which must be carefully understood. See for example Baddeley et al. 2015 (section 10.6) and Turner and Jeffs 2017 for explanation of the interpretation of Monte Carlo p-values and for some general discussion of Monte Carlo tests and of their advantages. Such tests effect substantial savings on computational costs with only marginal diminishment of power.

Value

A list with components:

`lrs`	the likelihood ratio statistic for the test
`pval`	the (Monte Carlo) p-value of the test
`simStats`	a vector of the values of the likelihood ratio statistics of the simulated data sets
`aic.ncomp`	a vector of the aic values for the `ncomp` models fitted to the simulated data sets
`aic.ncomp+nincr`	a vector of the aic values for the `ncomp+nincr` models fitted to the simulated data sets. (Note that in any given instance `ncomp` and `nincr` are replaced by the actual numeric values that are used in that instance.)
`df`	the degrees of freedom that would be appropriate if the test statistic actually had a chi-squared distribution. Explicitly `df` is equal to the number of parameters in the alternative model minus the number of parameters in the null model
`screwUps`	a data frame with columns `seed`, `i` and `type`, containing respectively the random number generator seed, the index and the type of each screw-up. Note that if there were in fact no screw-ups then the `screwUps` entry of the returned value will not be present. Note also that it is possible for values in the `i` column of `screwUps` to be repeated. The entries of the `type` column take values in the set `{1,2,3,4,5}`. These values have the interpretation: • 1: singularity in the likelihood surface for the `ncomp` model • 2: the algorithm failed to converge when fitting the `ncomp` component model • 3: singularity in the likelihood surface for the `ncomp+nincr` model • 4: the algorithm converged for the `ncomp` component model but failed to converge for the `ncomp + nincr` model • 5: the likelihood ratio statistic for the `ncomp` model was *greater* than that for the `ncomp+nincr` model (which is *theoretically* impossible) Note that if a screw-up does occur, the replicate is redone completely so that the returned value contains results for a full `nsim` simulations.

The returned value has an attribute `"seed"` which is the (initial) value of the random number generation seed that was used. This is either the value of the argument `seed`, or, if this was `NULL`, a randomly generated value.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

Adrian Baddeley, Ege Rubak and Rolf Turner (2015). Spatial Point Patterns: Methodology and Applications with R. London: Chapman and Hall/CRC Press.

Rolf Turner and Celeste Jeffs (2017). A note on exact Monte Carlo hypothesis tests. *Communications in Statistics: Simulation and Computation* **46**, pp. 6545 – 6558.

See Also

cband(), covMix(), mixreg(), plot.cband(), plot.mixresid(), qqMix(), residuals.mixreg()

Examples

```
    ## Not run: 
       tst12 <- ncMcTest(plntsInf ~ aphRel,ncomp=1,data=aphids,seed=42)

## End(Not run) # Monte Carlo p-value is 0.01; mixture model is called for.
    TS1 <- list(list(beta=c(3.0,0.1),sigsq=16,lambda=0.5),
                list(beta=c(0.0,0.0),sigsq=16,lambda=0.5))
    TS2 <- list(list(beta=c(3.0,0.1),sigsq=9,lambda=1/3),
                list(beta=c(1.5,0.05),sigsq=9,lambda=1/3),
                list(beta=c(0.0,0.0),sigsq=9,lambda=1/3))
    x <- aphids$aphRel
    y <- aphids$plntsInf
    ## Not run: 
      nsim <- 999

## End(Not run)

    tst23 <- ncMcTest(x,y,nsim=nsim,ts1=TS1,ts2=TS2)
```

---

plot.cband — *Plot confidence bands for a mixture of regressions.*

Description

Plots the fitted lines and confidence and prediction bands as calculated by cband, for a mixture of regressions on one variable.

Usage

```
## S3 method for class 'cband'
plot(x, cbands=TRUE, pbands=TRUE,
                       type=c("both","upper","lower"),
                       legPos="topleft", sepFac=0.8, ...)
```

Arguments

`x`	An object of class `"cband"` specifying the fitted lines and confidence and prediction bands to be plotted, as produced by cband.
`cbands`	Logical scalar; should the confidence bands be plotted?
`pbands`	Logical scalar; should the prediction bands be plotted?
`type`	Character string specifying whether the bands plotted should be two-sided (`"both"`) or if only the upper (`"upper"` or lower (`"lower"`) envelopes should be plotted. May be abbreviated (e.g. to `"b"`, `"u"` or `"l"`).
`legPos`	A list with entries `x` and `y`, or a text string, specifying the placement of the legend. See `legend()` for details. The plotting of a legend may be suppressed `legPos=NULL`. The legend consists of two parts; the upper part, which specifies line types, will have an entry for confidence bands only if `cbands` is `TRUE` and likewise it will have an entry type for prediction bands only if `pbands` is `TRUE`. The lower part specifies the colours of the fitted lines corresponding to the different components. It will be present only if there is more than one colour for the components.
`sepFac`	"Separation factor". A numeric scalar determining the amount of separation between the two parts of the legend (see above). Has an effect only if both parts of the legend are present. Making `sepFac` larger increases the amount of separation between the two parts; making it smaller decreases the amount. Making it much smaller than the default value will cause over-printing.
`...`	Optional extra arguments for `plot()`, `points()` and `lines()`. These may include `xlim`, `ylim`, `xlab`, `ylab`, `lty`, `col`, `pch`, and `main`. If `col` is supplied, its first entry determines the colour in which the points are plotted, and the remaining entries of `col` determine the colours in which the fitted lines and envelopes (corresponding to the different components) are plotted. Note that `col` is replicated to have length `K+1` where `K` is the number of components in the model. The argument `lty` determines the line types for the fitted lines, the confidence envelopes and the prediction envelopes. These line types remain constant across components. Note that if `main` is not supplied then a "sensible" default main title is created. If no main title is desired, specify `main=""`.

Details

This function is a method for plot. Note that a simple plot of the fit may be produced by calling `plot(object)` where `object` is an object of class `"mixreg"` and the `x` argument of this function was produced by applying `cband()` to this object.

Value

None. This function is called for its side effect of producing a plot.

Author(s)

Rolf Turner `<r.turner@auckland.ac.nz>`

See Also

`cband()`, `plot.cint()`, `plot.mixreg()`, `plot.mixresid()`, `qqMix()`, `residuals.mixreg()`

Examples

```
thStrt <- list(list(beta=c(3.0,0.1),sigsq=16,lambda=0.5),
               list(beta=c(0.0,0.0),sigsq=16,lambda=0.5))
fit    <- mixreg(plntsInf~aphRel,ncomp=2,thetaStart=thStrt,
                 covMat=TRUE,data=aphids)
cbds   <- cband(fit)
plot(cbds)
plot(cbds,pbands=FALSE)
plot(cbds,pbands=FALSE,type="u")
```

---

plot.cint — *Plot confidence intervals for a Gaussian scalar mixture model.*

Description

Plots confidence and prediction intervals, corresponding to each component of a Gaussian scalar mixture model.

Usage

```
## S3 method for class 'cint'
plot(x, cints=TRUE, pints=TRUE,rugged=TRUE,
                      type=c("both","upper","lower"),...)
```

Arguments

`x`	An object of class `"cint"`, specifying the confidence and prediction intervals to be plotted, as produced by the undocumented function `cint()`. Such an object would usually be produced (indirectly) from a call to `cband()`. If the class `"mixreg"` object, to which `cband()` is applied, specifies a fit to a model involving no predictors, then confidence and prediction *bands* make no sense, and the object is handed over to `cint()` to produce confidence and prediction *intervals*.
`cints`	Logical scalar; should the confidence intervals be plotted?
`pints`	Logical scalar; should the prediction intervals be plotted?
`rugged`	Logical scalar. Should a “rug” be added to the plot, displaying the values of the variable to which the model was fitted? See `rug()`. If `rugged` is `TRUE` then a rug is added at the right hand margin of the plot.
`type`	Character string specifying whether the intervals plotted should be two-sided (`"both"`) or if only the upper (`"upper"` or lower (`"lower"`) bounds should be plotted. May be abbreviated (e.g. to `"b"`, `"u"` or `"l"`).

...	Optional extra arguments for plot() and title() These may include xlim, ylim, xlab, ylab, col, pch, and main. Note that col is replicated to have length K where K is the number of components in the model. If main is not supplied then a “sensible” default main title is created. If no main title is desired, specify main="".

Details

This function is a method for plot. Note that a simple plot of the fit may be produced by calling `plot(object)` where `object` is an object of class `"mixreg"` and the `x` argument of this function was produced by applying `cband()` to this object.

Value

None. This function is called for its side effect of producing a plot.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

cband(), plot.cband(), plot.mixreg(), plot.mixresid(), qqMix(), residuals.mixreg()

Examples

```
fit  <- mixreg(plntsInf~1,ncomp=2,seed=42,covMat=TRUE,data=aphids)
cis  <- cband(fit) # *Could* call cint(fit) directly
plot(cis)
plot(cis,pintss=FALSE)
plot(cis,pintss=FALSE,type="u")
```

`plot.mixreg` *Plot a fitted mixture of regressions.*

Description

Plots the fitted regression lines for a mixture of regression models as fitted by `mixreg()` (or possibly `visualFit()`).

Usage

```
   ## S3 method for class 'mixreg'
plot(x, y, cMeth=c("none","distance","prob"),
           legPos = "topleft", ...)
```

Arguments

x	An object of class "mixreg" as returned by mixreg() or possibly by visualFit().
y	Not used.
cMeth	Text string specifying the "classification method". If cMeth is "none" no classification is done. If cMeth is "distance" then points are classified as "belonging" to a component of the mixture according to which of the regression lines they are nearest to, in terms of Euclidean distance. If cMeth is "prob" then points are classified as "belonging" to a component of the mixture according to which of the probabilities γ_{ij} of their being associated with that component is largest. If the points are classified then they are plotted in colours corresponding to that classification.
legPos	A list with entries x and y, or a text string, specifying the placement of the legend. See legend() for details. The plotting of a legend may be suppressed by setting legPos=NULL.
...	Optional extra arguments for plot(), points() and lines(). These may include xlim, ylim, xlab, ylab, lty, col, pch, and main. If col is supplied, then when cMeth is "none" its first entry determines the colour in which the points are plotted and the remaining entries of col determine the colours in which the fitted lines (corresponding to the different components) are plotted. If the points actually get classified then the entries of col determine the colours for both points and lines. Note that lty is replicated to have length K where K is the number of components in the model. If cMeth is "none" then col is replicated to have length K+1, otherwise it is replicated to have length K.

Value

If cMeth is "none" then no value is returned. Otherwise the value is the data component of the x argument of this function, augmented by an extra column groups. This column is a factor that specifies the component to which each point has been assigned. This data frame also has an attribute "cMeth", the value of the cMeth argument.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

plot.cband(), plot.mixresid(), qqMix(),

Examples

```
thStrt <- list(
                list(beta=c(26.07,48808),sigsq=1.1573,lambda=0.33333333),
                list(beta=c(23.48,32387),sigsq=1.8730,lambda=0.33333333),
                list(beta=c(-0.0597,20760),sigsq=0.2478,lambda=0.33333333)
              )
kfit  <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
plot(kfit,pch=8,col=c("red","green","blue","black"))
plot(kfit,pch=8,col=c("green","blue","black"),cMeth="d")
plot(kfit,pch=8,col=c("green","blue","black"),cMeth="p")
afit <- mixreg(plntsInf ~ aphRel,data=aphids,ncomp=2)
plot(afit,cMeth="p",col=c("blue","red"),pch=20)
# Separates the points into two groups incredibly clearly!
npfit <- mixreg(plntsInf ~ 1, data=aphids,ncomp=2)
plot(npfit,cMeth="p",col=c("blue","red"),pch=20)
```

---

plot.mixresid — *Plot residuals for a fitted mixture of linear regressions.*

Description

Plots the residuals against predictors or fitted values using symbols whose size is proportional to the probability that the associated observation was generated by the associated component of the model.

Usage

```
## S3 method for class 'mixresid'
plot(x, vsFit=FALSE, whichx=1,digits=2,
                         shape=c("disc","lozenge","square","none"),
                         ngon=20, size=1, gexp=1, polycol=NULL,
                         xlab=NULL, ...)
```

Arguments

`x`	An object of class `"mixresid"`, which consists of a list with entries providing the residuals, the relevant probabilities, the predictors, and the observations, as returned by `residuals.mixreg()`. When plotting against fitted values, it is probably better to used the standardized residuals, i.e. `residuals.mixreg()` should be called with `std=TRUE`.
`vsFit`	Logical scalar; should the residuals be plotted against the fitted values?
`whichx`	Integer scalar that indicates which predictor to plot against if there is more than one predictor. I.e. `whichx` indicates which column of the `x` matrix to use. If `vsFit` is `TRUE`, then `whichx` is ignored.
`digits`	Integer scalar giving the number of digits to which the fitted values should be rounded when these values are used as axis labels. Such use occurs only when there are no predictors in the model (i.e. when the formula is of the form `y ~ 1` — so that there is a single fitted value for each component) and when `vsFit` is `TRUE`. In all other circumstances `digits` is ignored.
`shape`	Character string indicating the shape of the plotting symbol; may be abbreviated, e.g. to `"d"`, `"l"`, `"s"` or `"n"`. If `shape` is `"none"` then "ordinary" plotting is done, using the usual plotting symbols etc. This can be much faster, but somewhat defeats the purpose of this plotting function in that the resulting plots can be misleading. Using `shape="n"` allows the user to compare the misleading result with plots in which "improbable" residuals are downweighted.

`ngon`	The `"disc"` shape is actually a regular polygon; `ngon` specifies how many sides it should have; ignored if `shape` is not equal to `"disc"`.
`size`	Positive numeric scalar. A scale factor to change the absolute sizes of the plotting symbols; values larger than 1 make the symbols larger; values less than 1 make them smaller.
`gexp`	Non-negative numeric scalar ("gamma exponent"). The power to which the conditional "component probabilities" γ_{ij} should be raised. The default value 1 causes the area of the plotting symbol to be proportional to the probability. Setting `gexp=0.5` effectively causes the diameter of the plotting symbol to be proportional to the probability. Setting `gexp=3` causes the area of the plotting symbol to be proportional to the square of the probability. Increasing `gexp` decreases the visual impact of plotted points with low probability, and vice versa. Setting `gexp=0` has effectively the same impact as setting `shape="none"`.
`polycol`	Character string specifying the colour in which polygons are plotted. This encompasses both the colour of the border of the polygon (specified as `border` argument in `polygon()`) and and the "fill" colour (specified as `col` argument in `polygon()`). Note that the border colour and fill colour are hard-wired to be the same in this function — there is no option to make them different. The "exception" to this rule results from using the default value of `polycol`, i.e. `NULL`. This causes the border colour to be `par("fg")` (usually black) and the polygons not to be filled (so that the fill colour is "transparent", i.e. in effect (usually) white.
`xlab`	The x label for the plot; defaults to `"x"` unless `vsFit` is `TRUE`, in which case it defaults to `"fitted values"`.
`...`	Additional arguments (e.g. `pch`, `col`, `cex`, `...`) to be passed to the `points()` function which actually plots the points when `shape="none"`.

Details

This function is a "method" for plot. The plot produced is visually assessed by ignoring or discounting small symbols.

The label for the x-axis ("`xlab`") is by default taken from the `vnms` component of the object being plotted. If you find this label to be unsatisfactory, supply the argument `xlab`.

Value

None. This function is called for its side effect of drawing a residual plot.

Side Effects

A residual plot is produced in whatever device is currently open.

ACKNOWLEDGEMENT

The idea of creating residual plots for regression mixtures by making the symbol size proportional to the associated probability is due to Prof. Adrian Baddeley who was, at the time, at the University of Western Australia. He is now (2021) at Curtin University.

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

See Also

mixreg(), residuals.mixreg() cband(), plot.cband(), qqMix()

Examples

```
    thStrt <- list(list(beta=c(3.0,0.1),sigsq=16,lambda=0.5),
                   list(beta=c(0.0,0.0),sigsq=16,lambda=0.5))
    fit    <- mixreg(aphRel,plntsInf,ncomp=2,thetaStart=thStrt,data=aphids)
    rrr    <- residuals(fit)
    plot(rrr)
    plot(rrr,shape="n")
# The plot with shape="n" gives an impression that variability
# increases with aphRel; the plot with default shape ("disc")
# does not give that impression
    rrs    <- residuals(fit,std=TRUE)
    plot(rrs,vsFit=TRUE)
    plot(rrs,vsFit=TRUE,shape="n")
    fit <- mixreg(plntsInf ~ 1,data=aphids,ncomp=2)
    rrr <- residuals(fit,std=TRUE)
    plot(rrr,vsFit=TRUE,digits=4,polycol="blue")
```

---

qqMix — *Draw a normal quantile-quantile plot of the residuals from a fitted mixture of linear regressions.*

Description

Draws a normal quantile-quantile plot using symbols whose sizes are proportional to the probabilities that the associated observations were generated by the associated components of the model.

Usage

```
qqMix(object, xlim=NULL, ylim=NULL,
      shape=c("disc","lozenge","square","none"), ngon=20, size=1, ...)
```

Arguments

`object`	An object of class `"mixresid"`, which consists of a list with entries providing the residuals, the relevant probabilities, the predictors, and the observations, as returned by `residuals.mixreg()`. When plotting against fitted values, it is probably better to used the standardized residuals, i.e. `residuals.mixreg()` should be called with `std=TRUE`.
`xlim`	A numeric vector of length two, giving the limits on the x-axis. Has the usual default.
`ylim`	A numeric vector of length two, giving the limits on the y-axis. Has the usual default.
`shape`	The shape of the plotting symbol. May be abbreviated to the first letter, i.e. `"d"`, `"l"`, `"s"`, or `"n"`. If `shape` is `"none"` then “ordinary” plotting is done, using the usual plotting symbols etc. This can be much faster, but somewhat defeats the purpose of this plotting function in that the resulting plots can be misleading. Using `shape="n"` allows the user to compare the misleading result with plots in which “improbable” residuals are downweighted,
`ngon`	The "disc" shape is actually a regular polygon; ngon specifies how many sides it should have; ignored if shape is not equal to "disc".
`size`	A scale factor to change the absolute sizes of the plotting symbols; values larger than 1 make the symbols larger than the default; values less than 1 make them smaller.
`...`	Additional arguments to be passed to the polygon function which actually draws the plotting symbols.

Details

The plot produced is visually assessed by ignoring or discounting small symbols.

Value

None. This function is called for its side effect of drawing a normal quantile-quantile plot.

ACKNOWLEDGEMENT

The idea of creating residual plots for regression mixtures by making the symbol size proportional to the associated probability is due to Prof. Adrian Baddeley who was, at the time (2000) at the University of Western Australia. He is now (2021) at Curtin University.

Author(s)

Rolf Turner `<r.turner@auckland.ac.nz>`

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

See Also

ncMcTest(),cband(), covMix(), mixreg(), plot.cband(), plot.mixresid(), qqMix(), residuals.mixreg()

Examples

```
# Aphids.
    fita <- mixreg(plntsInf~aphRel,ncomp=2,seed=42,data=aphids)
    resa <- residuals(fita,std=TRUE)
    qqMix(resa,size=2,shape="l")
    qqMix(resa,shape="n",pch=7,cex=1.5)
# Kilns.
    thStrt <- list(list(beta=c(26.1,48800),sigsq=0.58,lambda=0.33333333),
                   list(beta=c(23.5,32400),sigsq=0.58,lambda=0.33333333),
                   list(beta=c(-0.06,20760),sigsq=0.58,lambda=0.33333333))
    fitk   <- mixreg(y ~ x,ncomp=3,data=kilnAoneOut,thetaStart=thStrt)
    ## Not run:  # Takes too long
        resk   <- residuals(fitk,std=TRUE)
        qqMix(resk,shape="s")
        qqMix(resk,shape="s",xlim=c(-0.25,0.9),ylim=c(-3,3))
        qqMix(resk,shape="n")

## End(Not run)
```

---

residuals.mixreg *Calculate the residuals of a mixture of linear regressions.*

Description

Calculates the residuals from each component of the mixture and the matrix of probabilities that each observation was generated by each component.

Usage

```
## S3 method for class 'mixreg'
residuals(object, std=FALSE,...)
```

Arguments

object	An object of class "mixreg" as returned by mixreg().
std	Logical argument; if TRUE then the residuals are standardized (by dividing them by their estimated standard deviation).
...	Not used.

Details

The calculation of the estimated standard deviations of the residuals is a little bit complicated since each component of the model is fitted using weighted regression in a setting in which the weights are NOT the reciprocals of error variances. See the reference below for more detail.

Value

A list (of class "mixresid") with entries

`resid`	The residuals of the model, bundled together in a $n \times K$ matrix, where n is the number of observations and K is the number of components in the model. The kth column of this matrix is the vector of residuals from the kth component of the model.
`fvals`	Matrix of the fitted values of the model, structured like `resid` (above).
`gamma`	An $n \times K$ matrix of probabilities. The entry `gamma[i,j]` of this matrix is the (fitted) probability that observation i was generated by component j.
`x`	The matrix of predictors in the regression model (or if there is only one predictor, this predictor as a vector).
`y`	The vector of response values.
`vnms`	Character vector; the first entry is the name of the response. The remaining entries are “reasonable” names for the individual (vector) predictors. Note that if there is no predictor then `vnms` is of length two with second entry `"index"`.
`noPred`	Logical scalar; set to `TRUE` if there are no predictors in the model.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

T. Rolf Turner (2000). Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49** Part 3, pp. 371 – 384.

See Also

`ncMcTest()`,`cband()`, `covMix()`, `mixreg()`, `plot.cband()`, `plot.mixresid()`, `qqMix()`, `residuals.mixreg()`

Examples

```
fit    <- mixreg(aphRel,plntsInf,ncomp=2,seed=42,data=aphids)
r      <- residuals(fit)
plot(r)
fit    <- mixreg(plntsInf ~ 1,ncomp=2,data=aphids)
r      <- residuals(fit)
plot(r,shape="l",polycol="green")
```

rmixreg *Simulate data from a mixture of regressions model.*

Description

Simulate data from a mixture of regressions model, as specified by the user or as fitted to a data set. The simulation may be done either in a parametric or "semiparametric" manner.

Usage

```
rmixreg(x, ...)
## Default S3 method:
rmixreg(x, nobs, theta, seed = NULL,
                          xNms=NULL, yNm = "y", ...)
## S3 method for class 'mixreg'
rmixreg(x, semiPar = FALSE, conditional=semiPar,
                         seed = NULL, ...)
```

Arguments

`x`	For the default method, this is a numeric vector constituting a predictor for a regression model, or a matrix whose columns form such predictors. The number of columns of `x` (or of `as.matrix(x)`) must be equal to the number of linear coefficients, or be one less than this number, otherwise an error is thrown. If the number of columns of `x` is one less than the number of linear coefficients then it is assumed that the missing predictor is the intercept and a column of 1s gets prepended (internally). Note that the number of linear coefficients is determined from argument `theta` (see below). Note that `x` may be set equal to `NULL` in which case data are generated from a model with no predictors, i.e. from a scalar mixture model. In this case `nobs` (see below) must be specified. Data from a scalar mixture model may also be generated by specifying `x` to be a vector of 1s. (In this case `nobs` is ignored.) For the `"mixreg"` method this is an object of class `"mixreg"` as returned by the function `mixreg()`.
`nobs`	Integer scalar, specifying the number of observations to be generated. Used only if argument `x` in `NULL`. Otherwise the number of observations is determined from the length or number of rows of `x`.
`theta`	Either a list or a matrix specifying the parameters of the model from which data are to be simulated. If it is a list it should have components `beta`, `sigsq` and `lambda`. Each of these components is in turn a list of length `K` where `K` is the number of components in the model. The kth entry of `beta` is a vector of regression coefficients. These vectors must all have the same length. The kth entry of `sigsq` is a positive scalar specifying the error variance for the kth component. The kth entry of `lambda` is a probability (positive scalar, less than 1) specifying the probability that an observation corresponds to the kth component. Note that the `lambda` values must sum to 1. If `theta` is a matrix it is of dimension

	$K \times np$ where `np` is the number of parameters. Note that `np` is equal to $p + 2$ where p is the number of linear regression coefficients (the other 2 parameters being `sigsq` and `lambda`). The rows of this matrix correspond to components of the mixture and the columns to the parameters of the model.
`seed`	A numeric scalar. If not an integer it gets rounded to the nearest integer (so `seed=pi` has the same impact as `seed=3`. If not supplied, `seed` is selected randomly from `1,2,...,1e5`. The seed for random number generation is set to `seed` in `rmixreg()` before any random number generation is done.
`xNms`	Character vector of names for the predictors *other than* the intercept (if there is one). This vector must be of length equal to the number of (non-intercept) predictors. This is equal to the number of columns of `x` *prior* to its having a column of 1s prepended (given that this augmentation of `x` does indeed take place). If not supplied the names used are `X1`, ..., `Xn` where `n` is the number of predictors. (Except, if there is only one predictor it is named by the name of the `x` variable.)
`yNm`	Character scalar; a name for the response.
`semiPar`	Logical scalar. Should the simulation be done "semiparametrically"? (See **Details**.)
`conditional`	Logical scalar; should the component-selection probabilities be determined conditionally upon the observations?
`...`	Not used.

Details

In this context "parametric" bootstrapping means that the bootstrap data sets are generated by simulating from the fitted `ncomp` model parameters, using Gaussian errors. In contrast semiparametric bootstrapping means that the errors are generated by resampling from the residuals. Since at each predictor value there are `ncomp` residuals, one for each component of the model, the errors are selected from these `ncomp` possibilities. If the argument `conditional` is `TRUE` then the selection probabilities at this step are the conditional probabilities, of the observation being generated by each component of the model, given that observation. If `conditional` is `FALSE` then these probabilities are the corresponding entries of `lambda` (see **Value**. The residuals are sampled independently in either case. The procedure is termed *semi*parametric (rather than non-parametric) since the sampling probabilities depend on the parameters of the model. Note that it makes no sense to specify `conditional=TRUE` if `semiPar` is `FALSE`. Doing so will generate an error.

Value

A data frame whose columns consist of the predictors and the simulated response. For the default method the predictor are the columns of the matrix specified by argument `x`. They have names given by argument `xNms` if this was provided and by `X1`, `X2`, ..., `Xn` (where `n` is the number of columns of `x`) or simply `x` if there is only a single predictor. For the `"mixreg"` method the columns are the same as those of `x$data`, with response column replaced by the simulated response.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

Turner, T. R. (2000) Estimating the rate of spread of a viral infection of potato plants via mixtures of regressions. *Applied Statistics* **49**, Part 3 pp. 371 – 384.

See Also

mixreg()

Examples

```
fit  <- mixreg(plntsInf ~ aphRel, ncomp=2, data=aphids)
sim1 <- rmixreg(fit)
with(sim1,plot(aphRel,plntsInf,,main="Parametric simulation"))
sim2 <- rmixreg(fit,semiPar=TRUE)
with(sim2,plot(aphRel,plntsInf,,main="Semiparametric simulation"))
x    <- cbind(1:50,rnorm(50))
pmat <- matrix(c(3,5,0.01,1600,0.7,1,2,-0.01,100,0.3),nrow=2,byrow=TRUE)
sim3 <- rmixreg(x,theta=pmat,seed=42)
with(sim3,plot(X1,y,main="Using rmixreg.default; predictor 1"))
with(sim3,plot(X2,y,main="Using rmixreg.default; predictor 2"))
pmat <- matrix(c(10,2,0.7,3,1,0.3),nrow=2,byrow=TRUE)
sim4 <- rmixreg(x=rep(1,50),theta=pmat,seed=17)
sim5 <- rmixreg(x=NULL,nobs=50,theta=pmat,seed=17) # Same as sim4 but
                                                   # with no columns of 1s.
chk4 <- mixreg(y~1,data=sim4,ncomp=2,seed=116)
chk5 <- mixreg(y~1,data=sim5,ncomp=2,seed=116) # Same-same.
```

---

stepPlot *Plot the steps of the fit of a mixture of regressions model.*

Description

Fit a mixture of regressions model, one EM step at a time, plotting the result after each step.

Usage

```
stepPlot(fmla, ncomp = NULL, eqVar = FALSE, thetaStart = NULL,
         nsteps = 100, eps = 1e-06, digits = 7, maxTry = 5,
         seed = NULL, data)
```

Arguments

fmla	The formula specifying the regression in the mixture model that is to be fitted.
ncomp	Integer scalar; the number of components in the model. Defaults to 2.
eqVar	See mixreg().
thetaStart	See mixreg().
nsteps	Integer scalar; the maximum number of EM steps to be undertaken.

`eps`	See `mixreg()`.
`digits`	The number of digits to which the details about the EM steps are printed out.
`maxTry`	See `mixreg()`.
`seed`	See `mixreg()`.
`data`	A data frame containing the variables to which the model is to be fitted.

Details

Can only be used in an interactive session. *May* give some insight into convergence problems or into the reasons for unexpected results in the fit.

Value

An object of class `"mixreg"`. See `mixreg()`. The model being fitted may not have converged.

Author(s)

Rolf Turner <`r.turner@auckland.ac.nz`>

References

See `mixreg()`.

See Also

`mixreg()`.

Examples

```
## Not run:  # Interactive session is required.
    vfit <- visualFit(y~x,data=kilnAfull,ncomp=3)
    stepPlot(y ~ x,ncomp=3,data=kilnAfull,thetaStart=vfit$theta)

## End(Not run)
# The result is clearly slightly wrong in respect of the second ("middle")
# component, which has a slope that is too low.  The outlier in
# kilnAfull may be "pulling up" the left hand end of the fitted line.
```

`visualFit` *Fit a mixture of regressions model by "visual" means.*

Description

Displays a plot of the data and invites the user to "click" on points judged to lie on the various components.

Usage

```
visualFit(fmla, data=NULL, ncomp, eqVar=FALSE, chsnPts=NULL,
      keepPlotVisible = FALSE)
```

Arguments

`fmla`	A formula specifying the regression model to be fitted.
`data`	A list or data frame in which the variables specified by `fmla` may be searched for. Variables not found in `data` are searched for in the global environment.
`ncomp`	Positive integer scalar. The number of components in the mixture which is to be fitted.
`eqVar`	Logical scalar; should the error variance be the same for all components? (The alternative is that each component should be allowed to have a different error variance.)
`chsnPts`	A list with `ncomp` components each of which is a list of length two with components `x` and `y`. Each of `x` and `y` is a vector of length two, constituting the x and y coordinates respectively of two points on a line that presumably underlies the corresponding component.
`keepPlotVisible`	Logical scalar. Should the plot of the data, produced by this function, be kept visible after the model has been "fitted"? (Rather than being dismissed by `dev.off()`.)

Details

If the model involves more than one predictor, or if the specified predictor is a matrix with more than one column, then an error is thrown. This function is intended for use only with one-variable regression.

If there is an intercept in the model, then for each component (numbered 1 to `ncomp`) the user is invited to click on *two* points judged to lie on a line underlying that component. If there is no intercept, then the user is invited to click on a single point for each component, with the origin `(0,0)` taken (silently) to be the second point needed to determine the line.

The fit that this function returns is calculated by assigning a component to each point in the data set, based on which of the visually determined lines that point is closest to.

If `eqVar` is `TRUE` then the model is constructed using a factor, whose entries are these assigned components, as a predictor (along with the "x" variable in `fmla`) in a call to `lm()`. If code `eqVar` is `FALSE` then a model is fitted separately to each component. (See the code for details.) "Obviously" the linear coefficient estimates will be the same in either case. Only the error variance estimates will differ.

If `eq.var` is `TRUE` then the number of parameters in the model, as used in the calculation of `aic`, is `M = 2*K + (K-1) + 1 = 3*K` when there is an intercept term and `M = K + (K-1) + 1 = 2*K` when there is no intercept term.

If `eq.var` is `FALSE` then the number of parameters is `M = 2*K + (K-1) + K = 4*K -1` if there is an intercept, and `M = K + (K-1) + K = 3*K -1` if there is no intercept.

The argument chsnPts allows one to use this function in a non-interactive session by creating and saving, *a priori*, an object to be supplied as the value of chsnPts. If chsnPts is supplied then the method employed isn't really "visual", but presumably the object supplied would have been created in a visual manner. Be that as it may, this function is mainly intended to be used visually, that is *without* supplying chsnPts.

If chsnPts is not supplied then ("obviously"!!!) this function can be used *only* in an interactive session.

Value

An object of class "mixreg". (See mixreg().) Components nsteps and converged are set to NA. Component data has an extra column groups appended to it. This column is a factor specifying the components assigned to the points on the basis of distances from the lines determined by the chosen points.

The value also has an attribute "chsnPts". This is the list of points judged by the user to lie on the component lines (or the value of the chsnPts argument if this was supplied).

Author(s)

Rolf Turner <r.turner@auckland.ac.nz>

See Also

mixreg()

Examples

```
## Not run: 
    vfita <- visualFit(plntsInf ~ aphRel,data=aphids,ncomp=2)
    plot(vfita)
    vfitk1 <- visualFit(y ~ x, data=kilnAoneOut, ncomp=3)
    cp     <- attr(vfitk1,"chsnPts")
    vfitk2 <- visualFit(y ~ x, data=kilnAoneOut, eqVar=TRUE, chsnPts=cp)
    vfitk1$parmat
    vfitk2$parmat # Same as above except for the "sigsq" column.

## End(Not run)
```

Index