

Functions and Methods for Mixed-Effects Models

Help Files
Version 3.3
June 2000

by José C. Pinheiro and Douglas M. Bates

Bell Labs, Lucent Technologies and University of Wisconsin — Madison

This method function extracts sub-matrices from the positive-definite matrix represented by `x`.

```
x[i, j, drop]  
x[i, j] <- value
```

ARGUMENTS

- `x`: an object inheriting from class `pdMat` representing a positive-definite matrix.
- `i, j`: optional subscripts applying respectively to the rows and columns of the positive-definite matrix represented by `object`. When `i` (`j`) is omitted, all rows (columns) are extracted.
- `drop`: a logical value. If `TRUE`, single rows or columns are converted to vectors. If `FALSE` the returned value retains its matrix representation.
- `value`: a vector, or matrix, with the replacement values for the relevant piece of the matrix represented by `x`.

VALUE

if `i` and `j` are identical, the returned value will be `pdMat` object with the same class as `x`. Otherwise, the returned value will be a matrix. In the case a single row (or column) is selected, the returned value may be converted to a vector, according to the rules above.

SEE ALSO

[,pdMat

EXAMPLE

```
pd1 <- pdSymm(diag(3))  
pd1[1, , drop = F]  
pd1[1:2, 1:2] <- 3 * diag(2)
```

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `gls` and `lme`.

```
ACF(object, maxLag, ...)
```

ARGUMENTS

`object`: any object from which an autocorrelation function can be obtained. Generally an object resulting from a model fit, from which residuals can be extracted.

`maxLag`: maximum lag for which the autocorrelation should be calculated.

`...`: some methods for this generic require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

SEE ALSO

`ACF.gls`, `ACF.lme`

EXAMPLE

```
## see the method function documentation
```

This method function calculates the empirical autocorrelation function for the residuals from an `gls` fit. If a grouping variable is specified in `form`, the autocorrelation values are calculated using pairs of residuals within the same group; otherwise all possible residual pairs are used. The autocorrelation function is useful for investigating serial correlation models for equally spaced data.

```
ACF(object, maxLag, resType, form, na.action)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted model.

`maxLag`: an optional integer giving the maximum lag for which the autocorrelation should be calculated. Defaults to maximum lag in the residuals.

`resType`: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "pearson".

`form`: an optional one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . The time covariate must be integer valued. When a grouping factor is present in `form`, the autocorrelations are calculated using residual pairs within the same group. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.

`na.action`: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes `ACF.gls` to print an error message and terminate if there are any incomplete observations.

VALUE

a data frame with columns `lag` and `ACF` representing, respectively, the lag between residuals within a pair and the corresponding empirical autocorrelation. The returned value inherits from class `ACF`.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

SEE ALSO

`ACF.gls`, `plot.ACF`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary)
ACF(fml, form = ~ 1 | Mare)
```

ACF.lme

Autocorrelation Function for lme Residuals

ACF.lme

This method function calculates the empirical autocorrelation function for the within-group residuals from an lme fit. The autocorrelation values are calculated using pairs of residuals within the innermost group level. The autocorrelation function is useful for investigating serial correlation models for equally spaced data.

```
ACF(object, maxLag, resType)
```

ARGUMENTS

object: an object inheriting from class lme, representing a fitted linear mixed-effects model.

maxLag: an optional integer giving the maximum lag for which the autocorrelation should be calculated. Defaults to maximum lag in the within-group residuals.

resType: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "pearson".

VALUE

a data frame with columns lag and ACF representing, respectively, the lag between residuals within a pair and the corresponding empirical autocorrelation. The returned value inherits from class ACF.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

SEE ALSO

```
ACF.gls, plot.ACF
```

EXAMPLE

```
fml <- lme(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          random = ~ sin(2*pi*Time) | Mare)
ACF(fml, maxLag = 11)
```

This generic function calculates the Akaike information criterion for one or several fitted model objects for which a log-likelihood value can be obtained, according to the formula $-2\log Lik + 2n_{par}$, where n_{par} represents the number of parameters in the fitted model. When comparing fitted objects, the smaller the AIC, the better the fit.

```
AIC(object, ...)
```

ARGUMENTS

object: a fitted model object, for which there exists a `logLik` method to extract the corresponding log-likelihood, or an object inheriting from class `logLik`.

...: optional fitted model objects.

VALUE

if just one object is provided, returns a numeric value with the corresponding AIC; if more than one object are provided, returns a `data.frame` with rows corresponding to the objects and columns representing the number of parameters in the model (`df`) and the AIC.

REFERENCES

Sakamoto, Y., Ishiguro, M., and Kitagawa G. (1986) "Akaike Information Criterion Statistics", D. Reidel Publishing Company.

SEE ALSO

`logLik`, `BIC`, `AIC.logLik`

EXAMPLE

```
fm1 <- lm(distance ~ age, data = Orthodont) # no random effects
fm2 <- lme(distance ~ age, data = Orthodont) # random is ~ age
AIC(fm1, fm2)
```

AIC.logLik

AIC of a logLik Object

AIC.logLik

This function calculates the Akaike information criterion for an object inheriting from class `logLik`, according to the formula $-2\log Lik + 2n_{par}$, where n_{par} represents the number of parameters in the fitted model. When comparing fitted objects, the smaller the AIC, the better the fit.

```
AIC(object)
```

ARGUMENTS

object: an object inheriting from class `logLik`, usually resulting from applying a `logLik` method to a fitted model object.

VALUE

a numeric value with the corresponding AIC.

REFERENCES

Sakamoto, Y., Ishiguro, M., and Kitagawa G. (1986) "Akaike Information Criterion Statistics", D. Reidel Publishing Company.

SEE ALSO

`AIC`, `logLik`, `BIC`

EXAMPLE

```
fml <- lm(distance ~ age, data = Orthodont)
AIC(logLik(fml))
```

allCoef

Extract Coefficients from a Set of Objects

allCoef

The extractor function is applied to each object in `...`, with the result being converted to a vector. A `map` attribute is included to indicate which pieces of the returned vector correspond to the original objects in `...`.

```
allCoef(..., extract)
```

ARGUMENTS

...: objects to which `extract` will be applied. Generally these will be model components, such as `corStruct` and `varFunc` objects.

extract: an optional extractor function. Defaults to `coef`.

VALUE

a vector with all elements, generally coefficients, obtained by applying `extract` to the objects in `...`

SEE ALSO

`modelStruct`

EXAMPLE

```
cs1 <- corAR1(0.1)
vf1 <- varPower(0.5)
allCoef(cs1, vf1)
```

anova.gls

Compare Likelihoods of Fitted Objects

anova.gls

When only one fitted model object is present, a data frame with the sums of squares, numerator degrees of freedom, F-values, and P-values for Wald tests for the terms in the model (when `terms` and `L` are `NULL`), a combination of model terms (when `terms` is not `NULL`), or linear combinations of the model coefficients (when `L` is not `NULL`). Otherwise, when multiple fitted objects are being compared, a data frame with the degrees of freedom, the (restricted) log-likelihood, the Akaike Information Criterion (AIC), and the Bayesian Information Criterion (BIC) of each object is returned. If `test=TRUE`, whenever two consecutive objects have different number of degrees of freedom, a likelihood ratio statistic, with the associated p-value is included in the returned data frame.

```
anova(object, ..., test, type, adjustSigma, Terms, L, verbose)
```

ARGUMENTS

object: a fitted model object inheriting from class `gl`s, representing a generalized least squares fit.

...: other optional fitted model objects inheriting from classes `gl`s, `gnls`, `lm`, `lme`, `lmList`, `nlme`, `nlsList`, or `nls`.

test: an optional logical value controlling whether likelihood ratio tests should be used to compare the fitted models represented by `object` and the objects in `...`. Defaults to `TRUE`.

type: an optional character string specifying the type of sum of squares to be used in F-tests for the terms in the model. If `"sequential"`, the sequential sum of squares obtained by including the terms in the order they appear in the model is used; else, if `"marginal"`, the marginal sum of squares obtained by deleting a term from the model at a time is used. This argument is only used when a single fitted object is passed to the function. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to `"sequential"`.

adjustSigma: an optional logical value. If `TRUE` and the estimation method used to obtain `object` was maximum likelihood, the residual standard error is multiplied by $\sqrt{n_{obs}/(n_{obs} - n_{par})}$, where n_{par} represents the number of coefficients and n_{obs} the number of observations in the fitted model, converting it to a REML-like estimate. This argument is only used when a single fitted object is passed to the function. Default is `TRUE`.

Terms: an optional integer or character vector specifying which terms in the model should be jointly tested to be zero using a Wald F-test. If given as a character vector, its elements must correspond to term names; else, if given as an integer vector, its elements must correspond to the order in which terms are included in the model. This argument is only used when a single fitted object is passed to the function. Default is `NULL`.

L: an optional numeric vector or array specifying linear combinations of the coefficients in the model that should be tested to be zero. If given as an array, its rows define the linear combinations to be tested. If names are assigned to the vector elements (array columns), they must correspond to coefficients names and will be used to map the linear combination(s) to the coefficients; else, if no names are available, the vector elements (array columns) are assumed in the same order as the coefficients appear in the model. This argument is only used when a single fitted object is passed to the function. Default is `NULL`.

verbose: an optional logical value. If `TRUE`, the calling sequences for each fitted model object are printed with the rest of the output, being omitted if `verbose = FALSE`. Defaults to `FALSE`.

VALUE

a data frame inheriting from class `anova.lme`.

NOTE

Likelihood comparisons are not meaningful for objects fit using restricted maximum likelihood and with different fixed effects.

SEE ALSO

`gls`, `gnls`, `lme`, `AIC`, `BIC`, `print.anova.lme`

EXAMPLE

```
# AR(1) errors within each Mare
fm1 <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          correlation = corAR1(form = ~ 1 | Mare))
anova(fm1)
# variance changes with a power of the absolute fitted values?
fm2 <- update(fm1, weights = varPower())
anova(fm1, fm2)
```

When only one fitted model object is present, a data frame with the sums of squares, numerator degrees of freedom, denominator degrees of freedom, F-values, and P-values for Wald tests for the terms in the model (when `terms` and `L` are `NULL`), a combination of model terms (when `terms` is not `NULL`), or linear combinations of the model coefficients (when `L` is not `NULL`). Otherwise, when multiple fitted objects are being compared, a data frame with the degrees of freedom, the (restricted) log-likelihood, the Akaike Information Criterion (AIC), and the Bayesian Information Criterion (BIC) of each object is returned. If `test=TRUE`, whenever two consecutive objects have different number of degrees of freedom, a likelihood ratio statistic, with the associated p-value is included in the returned data frame.

```
anova(object, ..., test, type, adjustSigma, Terms, L, verbose)
```

ARGUMENTS

object: a fitted model object inheriting from class `lme`, representing a mixed-effects model.

...: other optional fitted model objects inheriting from classes `gls`, `gnls`, `lm`, `lme`, `lmList`, `nlme`, `nlsList`, or `nls`.

test: an optional logical value controlling whether likelihood ratio tests should be used to compare the fitted models represented by `object` and the objects in **...**. Defaults to `TRUE`.

type: an optional character string specifying the type of sum of squares to be used in F-tests for the terms in the model. If `"sequential"`, the sequential sum of squares obtained by including the terms in the order they appear in the model is used; else, if `"marginal"`, the marginal sum of squares obtained by deleting a term from the model at a time is used. This argument is only used when a single fitted object is passed to the function. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to `"sequential"`.

adjustSigma: an optional logical value. If `TRUE` and the estimation method used to obtain `object` was maximum likelihood, the residual standard error is multiplied by $\sqrt{n_{obs}/(n_{obs} - n_{par})}$, where n_{par} represents the number of coefficients and n_{obs} the number of observations in the fitted model, converting it to a REML-like estimate. This argument is only used when a single fitted object is passed to the function. Default is `TRUE`.

Terms: an optional integer or character vector specifying which terms in the model should be jointly tested to be zero using a Wald F-test. If given as a character vector, its elements must correspond to term names; else, if given as an integer vector, its elements must correspond to the order in which terms are included in

the model. This argument is only used when a single fitted object is passed to the function. Default is `NULL`.

L: an optional numeric vector or array specifying linear combinations of the coefficients in the model that should be tested to be zero. If given as an array, its rows define the linear combinations to be tested. If names are assigned to the vector elements (array columns), they must correspond to coefficients names and will be used to map the linear combination(s) to the coefficients; else, if no names are available, the vector elements (array columns) are assumed in the same order as the coefficients appear in the model. This argument is only used when a single fitted object is passed to the function. Default is `NULL`.

verbose: an optional logical value. If `TRUE`, the calling sequences for each fitted model object are printed with the rest of the output, being omitted if `verbose = FALSE`. Defaults to `FALSE`.

VALUE

a data frame inheriting from class `anova.lme`.

NOTE

Likelihood comparisons are not meaningful for objects fit using restricted maximum likelihood and with different fixed effects.

SEE ALSO

`gls`, `gnls`, `nlme`, `lme`, `AIC`, `BIC`, `print.anova.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
anova(fml)
fm2 <- update(fml, random = pdDiag(~ age))
anova(fml, fm2)
```

as.matrix.corStruct

Matrix of a `corStruct` Object

as.matrix.corStruct

This method function extracts the correlation matrix, or list of correlation matrices, associated with `object`.

```
as.matrix(x)
```

ARGUMENTS

x: an object inheriting from class `corStruct`, representing a correlation structure.

VALUE

If the correlation structure includes a grouping factor, the returned value will be a list with components given by the correlation matrices for each group. Otherwise, the returned value will be a matrix representing the correlation structure associated with `object`.

SEE ALSO

`corClasses`, `corMatrix`

EXAMPLE

```
cst1 <- corAR1(form = ~ 1|Subject)
cst1 <- initialize(cst1, data = Orthodont)
as.matrix(cst1)
```

as.matrix.pdMat

Matrix of a pdMat Object

as.matrix.pdMat

This method function extracts the positive-definite matrix represented by `x`.

```
as.matrix(x)
```

ARGUMENTS

`x`: an object inheriting from class `pdMat`, representing a positive-definite matrix.

VALUE

a matrix corresponding to the positive-definite matrix represented by `x`.

SEE ALSO

`pdMat`, `corMatrix`

EXAMPLE

```
as.matrix(pdSymm(diag(4)))
```

as.matrix.reStruct

Matrices of an reStruct Object

as.matrix.reStruct

This method function extracts the positive-definite matrices corresponding to the `pdMat` elements of object.

```
as.matrix(object)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a list with components given by the positive-definite matrices corresponding to the elements of `object`.

SEE ALSO

`as.matrix.pdMat`, `reStruct`, `pdMat`

EXAMPLE

```
rs1 <- reStruct(pdSymm(diag(3)), form=~ Sex+age, data=Orthodont))
as.matrix(rs1)
```

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `pdMat`, `corStruct` and `varFunc`. See the appropriate method documentation for a description of the arguments.

```
asNatural(object, ...)
```

ARGUMENTS

object: an object whose representation is to be converted to a natural parameterization. Generally a model structure such as `pdMat`, `corStruct`, or `varFunc` objects.

...: some methods for this generic function may require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`asNatural.pdMat`, `asNatural.corStruct`, `asNatural.varFunc`

This method function converts `object` to a `corBandNat` object, represented in a "natural" parameterization. If `unconstrained = TRUE`, an unconstrained parameterization based on the logit of the correlations ($\log((1 + r)/(1 - r))$, with r denoting the correlation) is used. If `unconstrained` is set to `FALSE`, the (constrained) correlations are used to parameterize the returned `corBandNat` object.

```
asNatural(object, unconstrained)
```

ARGUMENTS

object: an object inheriting from class `corBand`, representing a banded correlation structure, whose representation is to be converted to a natural parameterization.

unconstrained: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

an object of class `corBandNat` representing a banded correlation structure in natural parameterization.

SEE ALSO

`asNatural.corStruct`, `corBand`, `corBandNat`

EXAMPLE

```
cst1 <- corBand(form = ~ 1|Subject)
cst1 <- initialize(cst1, data = Orthodont)
asNatural(cst1)
```

asNatural.corStruct corStruct Object in Natural Parameterization **asNatural.corStruct**

This method function converts object to another corStruct object represented in a "natural" parameterization. If unconstrained is set to TRUE, the natural unconstrained parameterization used is the same as the one used for object, which is returned unchanged. The exceptions are the corSymm and corBand classes (see the documentation on the appropriate asNatural method). If unconstrained is set to FALSE, a constrained natural parameterization is used to represent the object. In this case, a constructor with the name Nat appended to the end of the original object constructor is called (e.g., corAR1Nat is called for corAR1 objects). corStruct objects with constrained natural parameterization are represented by the coefficients returned by coef(object, uncons = FALSE), except for corSymm and corBand objects (see the documentation on corSymmNat and corBandNat, respectively).

```
asNatural(object, unconstrained)
```

ARGUMENTS

object: an object inheriting from class corStruct, representing a correlation structure, whose representation is to be converted to a natural parameterization.

unconstrained: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to TRUE.

VALUE

If unconstrained = TRUE, object is returned unchanged (except for corSymm and corBand objects), else, if unconstrained = FALSE, a corStruct object parameterized according to the constrained coefficients corresponding to object (except for corSymm and corBand objects) – the class corresponding to these objects is obtained by appending the name Nat to the end of the original class of object.

SEE ALSO

```
asNatural.corSymm, asNatural.corBand, corBandNat, corSymmNat
```

EXAMPLE

```
cst1 <- corCompSymm(0.5, form = ~ 1|Subject)
cst1 <- initialize(cst1, data = Orthodont)
asNatural(cst1, unc = F)
```

asNatural.corSymm corSymm Object in Natural Parameterization **asNatural.corSymm**

This method function converts `object` to a `corSymmNat` object, represented in a "natural" parameterization. If `unconstrained = TRUE`, an unconstrained parameterization based on the logit of the correlations ($\log((1 + r)/(1 - r))$), with r denoting the correlation) is used. If `unconstrained` is set to `FALSE`, the (constrained) correlations are used to parameterize the returned `corSymmNat` object.

```
asNatural(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `corSymm`, representing a general correlation structure, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

an object of class `corSymmNat` representing a general correlation structure in natural parameterization.

SEE ALSO

```
asNatural.corStruct, corSymm, corSymmNat
```

EXAMPLE

```
cst1 <- corSymm(form = ~ 1|Subject)
cst1 <- initialize(cst1, data = Orthodont)
asNatural(cst1)
```

This method function converts `object` to a `pdBandNat` object, represented in a "natural" parameterization. If `unconstrained = TRUE`, an unconstrained parameterization based on the logarithm of the standard deviations and the logit of the correlations ($\log((1+r)/(1-r))$, with r denoting the correlation) is used. If `unconstrained` is set to `FALSE`, the variances and covariances are used to parameterize the returned `pdBandNat` object.

```
asNatural(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `pdBand`, representing a banded positive-definite matrix, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

an object of class `pdBandNat` representing a banded positive-definite matrix in natural parameterization.

SEE ALSO

```
asNatural.pdMat, pdBand, pdBandNat
```

EXAMPLE

```
pd1 <- pdBand(diag(1:3), form = ~ age + age^2, data = Orthodont)
asNatural(pd1)
```


This method function converts `object` to another `pdMat` object represented in a "natural" parameterization. If `unconstrained` is set to `TRUE`, the natural unconstrained parameterization used is the same as the one used for `object`, which is returned unchanged. The exceptions are the `pdSymm` and `pdBand` classes (see the documentation on the appropriate `asNatural` method). If `unconstrained` is set to `FALSE`, a constrained natural parameterization is used to represent the object. In this case, a constructor with the name `Nat` appended to the end of the original object constructor is called (e.g., `pdDiagNat` is called for `pdDiag` objects). `pdMat` objects with constrained natural parameterization are represented by the variances and, when appropriate, the covariances of the positive-definite matrix they represent.

```
asNatural(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive-definite matrix, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

If `unconstrained = TRUE`, `object` is returned unchanged (except for `pdSymm` and `pdBand` objects), else, if `unconstrained = FALSE`, a `pdMat` object parameterized according to the variances and covariances of the positive-definite matrix they represent - the class corresponding to these objects is obtained by appending the name `Nat` to the end of the original class of `object`.

SEE ALSO

```
asNatural.pdSymm, asNatural.pdBand, pdBandNat, pdSymmNat
```

EXAMPLE

```
pd1 <- pdDiag(diag(1:3), form = ~ age+age^2, data = Orthodont)
asNatural(pd1, unc = F)
```

asNatural.pdSymm pdSymm Object in Natural Parameterization **asNatural.pdSymm**

This method function converts `object` to a `pdSymmNat` object, represented in a "natural" parameterization. If `unconstrained = TRUE`, an unconstrained parameterization based on the logarithm of the standard deviations and the logit of the correlations ($\log((1+r)/(1-r))$, with r denoting the correlation) is used. If `unconstrained` is set to `FALSE`, the variances and covariances are used to parameterize the returned `pdSymmNat` object.

```
asNatural(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `pdSymm`, representing a general positive-definite matrix, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

an object of class `pdSymmNat` representing a general positive-definite matrix in natural parameterization.

SEE ALSO

```
asNatural.pdMat, pdSymm, pdSymmNat
```

EXAMPLE

```
pd1 <- pdSymm(diag(1:3), form = ~ age + age^2, data = Orthodont)
asNatural(pd1)
```

This method function converts `object` to another `varFunc` object represented in a "natural" parameterization. If `unconstrained` is set to `TRUE`, the natural unconstrained parameterization used is the same as the one used for `object`, which is returned unchanged. If `unconstrained` is set to `FALSE`, a constrained natural parameterization is used to represent the object. In this case, a constructor with the name `Nat` appended to the end of the original `object` constructor is called (e.g., `varPowerNat` is called for `varPower` objects). `varFunc` objects with constrained natural parameterization are represented by the coefficients returned by `coef(object, uncons = FALSE)`.

```
asNatural(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `varFunc`, representing a variance function structure, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

If `unconstrained = TRUE`, `object` is returned unchanged, else, if `unconstrained = FALSE`, a `varFunc` object parameterized according to the constrained coefficients corresponding to `object` - the class corresponding to these objects is obtained by appending the name `Nat` to the end of the original class of `object`.

EXAMPLE

```
vfl <- varConstPower(2, 0.5, form = ~ age)
vfl <- initialize(vfl, data = Orthodont)
asNatural(vfl, unc = F)
```

asOneFormula

Combine Formulas of a Set of Objects

asOneFormula

The names of all variables used in the formulas extracted from the objects defined in ... are converted into a single linear formula, with the variables names separated by +.

```
asOneFormula(..., omit)
```

ARGUMENTS

...: objects, or lists of objects, from which a formula can be extracted.

omit: an optional character vector with the names of variables to be omitted from the returned formula. Defaults to `c(".", "pi")`.

VALUE

a one-sided linear formula with all variables named in the formulas extracted from the objects in ..., except the ones listed in omit.

SEE ALSO

`formula`, `all.vars`

EXAMPLE

```
asOneFormula(y ~ x + z | g, list(~ w, ~ t * sin(2 * pi)))
```

asOneSidedFormula

Convert to One-Sided Formula

asOneSidedFormula

Names, expressions, and strings are converted to one-sided formulas. If object is a formula, it must be one-sided, in which case it is returned unaltered.

```
asOneSidedFormula(object)
```

ARGUMENTS

object: a one-sided formula, an expression, a numeric value, or a character string.

VALUE

a one-sided formula representing object

SEE ALSO

`formula`

EXAMPLE

```
asOneSidedFormula("age")
asOneSidedFormula(~ age)
```

asTable	Convert groupedData to a matrix	asTable
----------------	---------------------------------	----------------

Create a tabular representation of the response in a balanced groupedData object.

```
asTable(object)
```

ARGUMENTS

object: A balanced groupedData object

VALUE

A matrix. The data in the matrix are the values of the response. The columns correspond to the distinct values of the primary covariate and are labelled as such. The rows correspond to the distinct levels of the grouping factor and are labelled as such.

SEE ALSO

groupedData, isBalanced, balancedGrouped

EXAMPLE

```
asTable( Orthodont )
```

augPred	Augmented Predictions	augPred
----------------	-----------------------	----------------

Predicted values are obtained at the specified values of `primary`. If `object` has a grouping structure (i.e. `getGroups(object)` is not `NULL`), predicted values are obtained for each group. If `level` has more than one element, predictions are obtained for each level of the `max(level)` grouping factor. If other covariates besides `primary` are used in the prediction model, their average (numeric covariates) or most frequent value (categorical covariates) are used to obtain the predicted values. The original observations are also included in the returned object.

```
augPred(object, primary, minimum, maximum, length.out, level, ...)
```

ARGUMENTS

object: a fitted model object from which predictions can be extracted, using a `predict` method.

primary: an optional one-sided formula specifying the primary covariate to be used to generate the augmented predictions. By default, if a covariate can be extracted from the data used to generate `object` (using `getCovariate`), it will be used as `primary`.

minimum: an optional lower limit for the primary covariate. Defaults to `min(primary)`.

maximum: an optional upper limit for the primary covariate. Defaults to `max(primary)`.

`length.out`: an optional integer with the number of primary covariate values at which to evaluate the predictions. Defaults to 51.

`level`: an optional integer vector specifying the desired prediction levels. Levels increase from outermost to innermost grouping, with level 0 representing the population (fixed effects) predictions. Defaults to the innermost level.

...: some methods for the generic may require additional arguments.

VALUE

a data frame with four columns representing, respectively, the values of the primary covariate, the groups (if `object` does not have a grouping structure, all elements will be 1), the predicted or observed values, and the type of value in the third column: `original` for the observed values and `predicted` (single or no grouping factor) or `predict.groupVar` (multiple levels of grouping), with `groupVar` replaced by the actual grouping variable name (`fixed` is used for population predictions). The returned object inherits from class `augPred`.

NOTE

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `gls`, `lme`, and `lmList`.

SEE ALSO

`plot.augPred`, `getGroups`, `predict`

EXAMPLE

```
fml <- lme(Orthodont)
augPred(fml, length.out = 2, level = c(0,1))
```

balancedGrouped	Create a <code>groupedData</code> object from a matrix	balancedGrouped
------------------------	--	------------------------

Create a `groupedData` object from a data matrix. This function can be used only with balanced data. The opposite conversion, from a `groupedData` object to a matrix, is performed by `asTable`.

```
balancedGrouped(form, data, labels=NULL, units=NULL)
```

ARGUMENTS

`form`: A formula of the form $y \sim x \mid g$ giving the name of the response, the primary covariate, and the grouping factor.

`data`: A matrix or data frame containing the values of the response grouped according to the levels of the grouping factor (rows) and the distinct levels of the primary covariate (columns). The `dimnames` of the matrix are used to construct the levels of the grouping factor and the primary covariate.

labels: an optional list of character strings giving labels for the response and the primary covariate. The label for the primary covariate is named `x` and that for the response is named `y`. Either label can be omitted.

units: an optional list of character strings giving the units for the response and the primary covariate. The units string for the primary covariate is named `x` and that for the response is named `y`. Either units string can be omitted.

VALUE

A balanced `groupedData` object.

SEE ALSO

`groupedData`, `isBalanced`, `asTable`

EXAMPLE

```
OrthoMat <- asTable( Orthodont )
Orth2 <- balancedGrouped(distance ~ age | Subject, data = OrthoMat,
  labels = list(x = "Age",
    y = "Distance from pituitary to pterygomaxillary fissure"),
  units = list(x = "(yr)", y = "(mm)"))
Orth2[ 1:10, ]      ## check the first few entries
```

BIC	Bayesian Information Criterion	BIC
------------	--------------------------------	------------

This generic function calculates the Bayesian information criterion, also known as Schwarz's Bayesian criterion (SBC), for one or several fitted model objects for which a log-likelihood value can be obtained, according to the formula $-2\log Lik + n_{par} \log(n_{obs})$, where n_{par} represents the number of parameters and n_{obs} the number of observations in the fitted model.

```
BIC(object, ...)
```

ARGUMENTS

object: a fitted model object, for which there exists a `logLik` method to extract the corresponding log-likelihood, or an object inheriting from class `logLik`.

...: optional fitted model objects.

VALUE

if just one object is provided, returns a numeric value with the corresponding BIC; if more than one object are provided, returns a `data.frame` with rows corresponding to the objects and columns representing the number of parameters in the model (`df`) and the BIC.

REFERENCES

Schwarz, G. (1978) "Estimating the Dimension of a Model", *Annals of Statistics*, 6, 461-464.

SEE ALSO

`logLik`, `AIC`, `BIC.logLik`

EXAMPLE

```
fm1 <- lm(distance ~ age, data = Orthodont) # no random effects
fm2 <- lme(distance ~ age, data = Orthodont) # random is ~ age
BIC(fm1, fm2)
```

BIC.logLik	BIC of a logLik Object	BIC.logLik
-------------------	------------------------	-------------------

This function calculates the Bayesian information criterion, also known as Schwarz's Bayesian criterion (SBC) for an object inheriting from class `logLik`, according to the formula $-2\log Lik + n_{par} \log(n_{obs})$, where n_{par} represents the number of parameters and n_{obs} the number of observations in the fitted model. When comparing fitted objects, the smaller the BIC, the better the fit.

```
BIC(object)
```

ARGUMENTS

`object`: an object inheriting from class `logLik`, usually resulting from applying a `logLik` method to a fitted model object.

VALUE

a numeric value with the corresponding BIC.

REFERENCES

Schwarz, G. (1978) "Estimating the Dimension of a Model", *Annals of Statistics*, 6, 461-464.

SEE ALSO

`BIC`, `logLik`, `AIC`

EXAMPLE

```
fm1 <- lm(distance ~ age, data = Orthodont)
BIC(logLik(fm1))
```


This method function extracts the coefficients associated with the correlation structure represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `corStruct`, representing a correlation structure.

`unconstrained`: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in "natural", possibly constrained, form. Defaults to `TRUE`.

`value`: a vector with the replacement values for the coefficients associated with `object`. It must be a vector with the same length of `coef(object)` and must be given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`. Object must be initialized (using `initialize`) before new values can be assigned to its coefficients.

SEE ALSO

`corClasses`, `initialize`

EXAMPLE

```
cst1 <- corARMA(p = 1, q = 1)
coef(cst1)
```

coef.gnls

Extract gnls Coefficients

coef.gnls

The estimated coefficients for the nonlinear model represented by `object` are extracted.

```
coef(object)
```

ARGUMENTS

`object`: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

VALUE

a vector with the estimated coefficients for the nonlinear model represented by `object`.

SEE ALSO

`gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
coef(fml)
```

coef.lmList

Extract lmList Coefficients

coef.lmList

The coefficients of each `lm` object in the `object` list are extracted and organized into a data frame, with rows corresponding to the `lm` components and columns corresponding to the coefficients. Optionally, the returned data frame may be augmented with covariates summarized over the groups associated with the `lm` components.

```
coef(object, augFrame, data, which, FUN, omitGroupingFactor)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

`augFrame`: an optional logical value. If `TRUE`, the returned data frame is augmented with variables defined in the data frame used to produce `object`; else, if `FALSE`, only the coefficients are returned. Defaults to `FALSE`.

`data`: an optional data frame with the variables to be used for augmenting the returned data frame when `augFrame = TRUE`. Defaults to the data frame used to fit `object`.

which: an optional positive integer or character vector specifying which columns of the data frame used to produce `object` should be used in the augmentation of the returned data frame. Defaults to all variables in the data.

FUN: an optional summary function or a list of summary functions to be applied to group-varying variables, when collapsing the data by groups. Group-invariant variables are always summarized by the unique value that they assume within that group. If `FUN` is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If `FUN` is a list of functions, the names in the list should designate classes of variables in the frame such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any group-varying variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

omitGroupingFactor: an optional logical value. When `TRUE` the grouping factor itself will be omitted from the group-wise summary of data but the levels of the grouping factor will continue to be used as the row names for the returned data frame. Defaults to `FALSE`.

VALUE

a data frame inheriting from class `coef.lmList` with the estimated coefficients for each `lm` component of `object` and, optionally, other covariates summarized over the groups corresponding to the `lm` components. The returned object also inherits from classes `ranef.lmList` and `data.frame`.

SEE ALSO

`lmList`, `fixef.lmList`, `ranef.lmList`, `plot.ranef.lmList`, `gsummary`

EXAMPLE

```
fml <- lmList(distance ~ age|Subject, data = Orthodont)
coef(fml)
coef(fml, augFrame = TRUE)
```

The estimated coefficients at level *i* are obtained by adding together the fixed effects estimates and the corresponding random effects estimates at grouping levels less or equal to *i*. The resulting estimates are returned as a data frame, with rows corresponding to groups and columns to coefficients. Optionally, the returned data frame may be augmented with covariates summarized over groups.

```
coef(object, augFrame, level, data, which, FUN, omitGroupingFactor)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

augFrame: an optional logical value. If `TRUE`, the returned data frame is augmented with variables defined in `data`; else, if `FALSE`, only the coefficients are returned. Defaults to `FALSE`.

level: an optional positive integer giving the level of grouping to be used in extracting the coefficients from an object with multiple nested grouping levels. Defaults to the highest or innermost level of grouping.

data: an optional data frame with the variables to be used for augmenting the returned data frame when `augFrame = TRUE`. Defaults to the data frame used to fit `object`.

which: an optional positive integer or character vector specifying which columns of `data` should be used in the augmentation of the returned data frame. Defaults to all columns in `data`.

FUN: an optional summary function or a list of summary functions to be applied to group-varying variables, when collapsing `data` by groups. Group-invariant variables are always summarized by the unique value that they assume within that group. If `FUN` is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If `FUN` is a list of functions, the names in the list should designate classes of variables in the frame such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any group-varying variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

omitGroupingFactor: an optional logical value. When `TRUE` the grouping factor itself will be omitted from the group-wise summary of `data` but the levels of the grouping factor will continue to be used as the row names for the returned data frame. Defaults to `FALSE`.

VALUE

a data frame inheriting from class `coef.lme` with the estimated coefficients at level `level` and, optionally, other covariates summarized over groups. The returned object also inherits from classes `ranef.lme` and `data.frame`.

SEE ALSO

`lme`, `fixef.lme`, `ranef.lme`, `plot.ranef.lme`, `gsummary`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
coef(fml)
coef(fml, augFrame = TRUE)
```

coef.modelStruct

Extract modelStruct Coefficients

coef.modelStruct

This method function extracts the coefficients associated with each component of the `modelStruct` list.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

object: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

unconstrained: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in "natural", possibly constrained, form. Defaults to `TRUE`.

value: a vector with the replacement values for the coefficients associated with `object`. It must be a vector with the same length of `coef(object)` and must be given in unconstrained form.

VALUE

a vector with all coefficients corresponding to the components of `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`. Object must be initialized (using `initialize`) before new values can be assigned to its coefficients.

SEE ALSO

`initialize`

EXAMPLE

```
lms1 <- lmeStruct(reStruct = reStruct(pdDiag(diag(2), ~ age)),
  corStruct = corAR1(0.3))
coef(lms1)
```

This method function extracts the coefficients associated with the positive-definite matrix represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `pdCompSymm`, representing a positive-definite matrix with compound symmetry structure.

`unconstrained`: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the standard deviation and the correlation coefficient of the compound symmetry of positive-definite matrix represented by `object` are returned. Defaults to `TRUE`.

`value`: a vector with the replacement values for the coefficients associated with `object`. It must be a vector of length two and must be given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`. `Object` must be initialized (using `initialize`) before new values can be assigned to its coefficients.

SEE ALSO

`coef.pdMat`, `pdMat`

EXAMPLE

```
coef(pdCompSymm(diag(3)), F)
```

This method function extracts the coefficients associated with the positive-definite matrix represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

object: an object inheriting from class `pdDiag`, representing a positive-definite matrix with diagonal structure.

unconstrained: a logical value. If `TRUE` the logarithm of the standard deviations corresponding to the variance-covariance matrix represented by `object` are returned. If `FALSE` the standard deviations are returned. Defaults to `TRUE`.

value: a vector with the replacement values for the coefficients associated with `object`. It must be a vector with the same length of `coef(object)` and must be given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`. Object must be initialized (using `initialize`) before new values can be assigned to its coefficients.

SEE ALSO

`coef.pdMat`, `pdMat`

EXAMPLE

```
coef(pdDiag(diag(3)))
```

This method function extracts the coefficients associated with the positive-definite matrix represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `pdIdent`, representing a multiple of the identity positive-definite matrix.

`unconstrained`: a logical value. If `TRUE` the logarithm of the standard deviation corresponding to the variance-covariance matrix represented by `object` is returned. If `FALSE` the standard deviation is returned. Defaults to `TRUE`.

`value`: a vector with the replacement values for the coefficients associated with `object`. It must be a numeric value given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`. `Object` must be initialized (using `initialize`) before new values can be assigned to its coefficients.

SEE ALSO

`coef.pdMat`, `pdMat`

EXAMPLE

```
coef(pdIdent(diag(3)))
```


This method function extracts the coefficients associated with the positive-definite matrix represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive-definite matrix.

`unconstrained`: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the upper triangular elements of the positive-definite matrix represented by `object` are returned. Defaults to `TRUE`.

`value`: a vector with the replacement values for the coefficients associated with `object`. It must be a vector with the same length of `coef(object)` and must be given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`.

REFERENCES

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

SEE ALSO

`pdMat`

EXAMPLE

```
coef(pdSymm(diag(3)))
```

This method function extracts the coefficients associated with the positive-definite matrix represented by `object`.

```
coef(object, unconstrained)
coef(object) <- value
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

unconstrained: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in "natural", possibly constrained, form. Defaults to `TRUE`.

value: a vector with the replacement values for the coefficients associated with `object`. It must be a vector with the same length of `coef(object)` and must be given in unconstrained form.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`.

SEE ALSO

`coef.pdMat`, `reStruct`, `pdMat`

EXAMPLE

```
rs1 <- reStruct(list(A = pdSymm(diag(1:3), form = ~ Score),
  B = pdDiag(2 * diag(4), form = ~ Educ)))
coef(rs1)
```

This method function extracts the coefficients associated with the variance function structure represented by `object`.

```
coef(object, unconstrained, allCoef)
coef(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `varFunc` representing a variance function structure.

`unconstrained`: a logical value. If `TRUE` the coefficients are returned in unconstrained form (the same used in the optimization algorithm). If `FALSE` the coefficients are returned in "natural", generally constrained form. Defaults to `TRUE`.

`allCoef`: a logical value. If `FALSE` only the coefficients which may vary during the optimization are returned. If `TRUE` all coefficients are returned. Defaults to `FALSE`.

`value`: a vector with the replacement values for the coefficients associated with `object`. It must have the same length of `coef(object)` and must be given in unconstrained form. `object` must be initialized before new values can be assigned to its coefficients.

VALUE

a vector with the coefficients corresponding to `object`.

SIDE EFFECTS

On the left side of an assignment, sets the values of the coefficients of `object` to `value`.

SEE ALSO

`varFunc`

EXAMPLE

```
vf1 <- varPower(1)
coef(vf1)
coef(vf1) <- 2
```

coef<-

Assign Values to Coefficients

coef<-

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include all `pdMat`, `corStruct`, and `varFunc` classes, `reStruct`, and `modelStruct`.

```
coef(object, ...) <- value
```

ARGUMENTS

object: any object representing a fitted model, or, by default, any object with a `coef` component.

...: some methods for this generic function may require additional arguments.

value: a value to be assigned to the coefficients associated with `object`.

VALUE

will depend on the method function; see the appropriate documentation.

SEE ALSO

`coef`

EXAMPLE

```
## see the method function documentation
```

collapse

Collapse According to Groups

collapse

This function is generic; method functions can be written to handle specific classes of objects. Currently, only a `groupedData` method is available.

```
collapse(object, ...)
```

ARGUMENTS

object: an object to be collapsed, usually a data frame.

...: some methods for the generic may require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`collapse.groupedData`

EXAMPLE

```
## see the method function documentation
```

If `object` has a single grouping factor, it is returned unchanged. Else, it is summarized by the values of the `displayLevel` grouping factor (or the combination of its values and the values of the covariate indicated in `preserve`, if any is present). The collapsed data is used to produce a new `groupedData` object, with grouping factor given by the `displayLevel` factor.

```
collapse(object, collapseLevel, displayLevel, outer, inner,
         preserve, FUN, subset)
```

ARGUMENTS

- `object`: an object inheriting from class `groupedData`, generally with multiple grouping factors.
- `collapseLevel`: an optional positive integer or character string indicating the grouping level to use when collapsing the data. Level values increase from outermost to innermost grouping. Default is the highest or innermost level of grouping.
- `displayLevel`: an optional positive integer or character string indicating the grouping level to use as the grouping factor for the collapsed data. Default is `collapseLevel`.
- `outer`: an optional logical value or one-sided formula, indicating covariates that are outer to the `displayLevel` grouping factor. If equal to `TRUE`, the `displayLevel` element `attr(object, "outer")` is used to indicate the outer covariates. An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. Defaults to `NULL`, meaning that no outer covariates are to be used.
- `inner`: an optional logical value or one-sided formula, indicating a covariate that is inner to the `displayLevel` grouping factor. If equal to `TRUE`, `attr(object, "inner")` is used to indicate the inner covariate. An inner covariate can change within the sets of rows defined by the grouping factor. Defaults to `NULL`, meaning that no inner covariate is present.
- `preserve`: an optional one-sided formula indicating a covariate whose levels should be preserved when collapsing the data according to the `collapseLevel` grouping factor. The collapsing factor is obtained by pasting together the levels of the `collapseLevel` grouping factor and the values of the covariate to be preserved. Default is `NULL`, meaning that no covariates need to be preserved.
- `FUN`: an optional summary function or a list of summary functions to be used for collapsing the data. The function or functions are applied only to variables in `object` that vary within the groups defined by `collapseLevel`. Invariant variables are always summarized by group using the unique value that they assume

within that group. If `FUN` is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If `FUN` is a list of functions, the names in the list should designate classes of variables in the data such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any non-invariant variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

`subset`: an optional named list. Names can be either positive integers representing grouping levels, or names of grouping factors. Each element in the list is a vector indicating the levels of the corresponding grouping factor to be preserved in the collapsed data. Default is `NULL`, meaning that all levels are used.

VALUE

a `groupedData` object with a single grouping factor given by the `displayLevel` grouping factor, resulting from collapsing `object` over the levels of the `collapseLevel` grouping factor.

REFERENCES

Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at <http://nlme.stat.wisc.edu>.

SEE ALSO

`groupedData`, `plot.nmGroupedData`

EXAMPLE

```
# collapsing by Dog
# same as collapse(Pixel, collapse = "Dog")
collapse(Pixel, collapse = 1)
```

The columns in `object1` and `object2` are put together in matrices which allow direct comparison of the individual elements for each object. Missing columns in either object are replaced by `NA`s.

```
compareFits(object1, object2, which)
```

ARGUMENTS

`object1, object2`: data frames, or matrices, with the same row names, but possibly different column names. These will usually correspond to coefficients from fitted objects with a grouping structure (e.g. `lme` and `lmList` objects).

`which`: an optional integer or character vector indicating which columns in `object1` and `object2` are to be used in the returned object. Defaults to all columns.

VALUE

a three-dimensional array, with the third dimension given by the number of unique column names in either `object1` or `object2`. To each column name there corresponds a matrix with as many rows as the rows in `object1` and two columns, corresponding to `object1` and `object2`. The returned object inherits from class `compareFits`.

SEE ALSO

```
plot.compareFits, pairs.compareFits, comparePred, coef, ranef
```

EXAMPLE

```
fm1 <- lmList(Orthodont)
fm2 <- lme(fm1)
compareFits(coef(fm1), coef(fm2))
```

Predicted values are obtained at the specified values of `primary` for each object. If either `object1` or `object2` have a grouping structure (i.e. `getGroups(object)` is not `NULL`), predicted values are obtained for each group. When both objects determine groups, the group levels must be the same. If other covariates besides `primary` are used in the prediction model, their group-wise averages (numeric covariates) or most frequent values (categorical covariates) are used to obtain the predicted values. The original observations are also included in the returned object.

```
comparePred(object1, object2, primary, minimum, maximum,
            length.out, level, ...)
```

ARGUMENTS

`object1, object2`: fitted model objects, from which predictions can be extracted using the `predict` method.

`primary`: an optional one-sided formula specifying the primary covariate to be used to generate the augmented predictions. By default, if a covariate can be extracted from the data used to generate the objects (using `getCovariate`), it will be used as `primary`.

`minimum`: an optional lower limit for the primary covariate. Defaults to `min(primary)`.

`maximum`: an optional upper limit for the primary covariate. Defaults to `max(primary)`.

`length.out`: an optional integer with the number of primary covariate values at which to evaluate the predictions. Defaults to 51.

`level`: an optional integer specifying the desired prediction level. Levels increase from outermost to innermost grouping, with level 0 representing the population (fixed effects) predictions. Only one level can be specified. Defaults to the innermost level.

`...`: some methods for the generic may require additional arguments.

VALUE

a data frame with four columns representing, respectively, the values of the primary covariate, the groups (if `object` does not have a grouping structure, all elements will be 1), the predicted or observed values, and the type of value in the third column: the objects' names are used to classify the predicted values and `original` is used for the observed values. The returned object inherits from classes `comparePred` and `augPred`.

NOTE

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `gls`, `lme`, and `lmList`.

SEE ALSO

`augPred`, `getGroups`

EXAMPLE

```
fm1 <- lme(distance ~ age * Sex, data = Orthodont, random = ~ age)
fm2 <- update(fm1, distance ~ age)
comparePred(fm1, fm2, length.out = 2)
```

corAR1	AR(1) Correlation Structure	corAR1
---------------	-----------------------------	---------------

This function is a constructor for the `corAR1` class, representing an autocorrelation structure of order 1. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corAR1(value, form, fixed)
```

ARGUMENTS

value: the value of the lag 1 autocorrelation, which must be between -1 and 1. Defaults to 0 (no autocorrelation).

form: a one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.

fixed: an optional logical value indicating whether the coefficient should be allowed to vary in the optimization, or kept fixed at its initial value. Defaults to `FALSE`, in which case the coefficient is allowed to vary.

VALUE

an object of class `corAR1`, representing an autocorrelation structure of order 1.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

SEE ALSO

`initialize.corStruct`

EXAMPLE

```
## covariate is observation order and grouping factor is Mare
cs1 <- corAR1(0.2, form = ~ 1 | Mare)
```

This function is a constructor for the `corARMA` class, representing an autocorrelation-moving average correlation structure of order (p, q). Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corARMA(value, form, p, q, fixed)
```

ARGUMENTS

- value:** a vector with the values of the autoregressive and moving average parameters, which must have length $p + q$ and all elements between -1 and 1. Defaults to a vector of zeros, corresponding to uncorrelated observations.
- form:** a one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must be integer valued. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- p, q:** non-negative integers specifying respectively the autoregressive order and the moving average order of the ARMA structure. Both default to 0.
- fixed:** an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corARMA`, representing an autocorrelation-moving average correlation structure.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

SEE ALSO

```
initialize.corStruct
```

EXAMPLE

```
## ARMA(1,2) structure, with observation order as a covariate and
## Mare as grouping factor
csl <- corARMA(c(0.2, 0.3, -0.1), form = ~ 1 | Mare, p = 1, q = 2)
```

This function is a constructor for the `corBand` class, representing a banded correlation structure. The internal representation of this structure, in terms of unconstrained parameters, uses the spherical parameterization defined in Pinheiro and Bates (1996). Objects created using this constructor must later be initialized using the appropriate `initialize` method.

```
corBand(value, form, fixed, ord)
```

ARGUMENTS

- `value`: an optional vector with the parameter values. Default is `numeric(0)`, which results in a vector of zeros of appropriate dimension being assigned to the parameters when object is initialized (corresponding to an identity correlation structure).
- `form`: a one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . A covariate for this correlation structure must such that its unique values form a sequence of consecutive integers. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- `fixed`: an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial value. Defaults to `FALSE`, in which case the coefficients are allowed to vary.
- `ord`: an optional integer specifying the order of the banded matrix represented by the returned `corBand` object, defined as the number of nonzero diagonals in the upper-triangular part of the matrix. Defaults to 2.

VALUE

an object of class `corBand` representing a banded correlation structure.

REFERENCES

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

SEE ALSO

```
initialize.corStruct
```

EXAMPLE

```
## covariate is observation order and grouping factor is Subject
cs1 <- corBand(form = ~ 1 | Subject)
```

This function is a constructor for the `corCAR1` class, representing an autocorrelation structure of order 1, with a continuous time covariate. Objects created using this constructor need to be later initialized using the appropriate `initialize` method.

```
corCAR1(value, form, fixed)
```

ARGUMENTS

- value:** the correlation between two observations one unit of time apart. Must be between 0 and 1. Defaults to 0.2.
- form:** a one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . Covariates for this correlation structure need not be integer valued. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- fixed:** an optional logical value indicating whether the coefficient should be allowed to vary in the optimization, or kept fixed at its initial value. Defaults to `FALSE`, in which case the coefficient is allowed to vary.

VALUE

an object of class `corCAR1`, representing an autocorrelation structure of order 1, with a continuous time covariate.

REFERENCES

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Jones, R.H. (1993) "Longitudinal Data with Serial Correlation: A State-space Approach", Chapman and Hall

SEE ALSO

```
initialize.corStruct
```

EXAMPLE

```
## covariate is Time and grouping factor is Mare
csl <- corCAR1(0.2, form = ~ Time | Mare)
```

corClasses	Correlation Structure Classes	corClasses
-------------------	-------------------------------	-------------------

Standard classes of correlation structures (`corStruct`) available in the `nlme` library.

STANDARD CLASSES

`corAR1`: autoregressive process of order 1.

`corARMA`: autoregressive moving average process, with arbitrary orders for the autoregressive and moving average components.

`corBand`: banded correlation structure.

`corCAR1`: continuous autoregressive process (AR(1) process for a continuous time covariate).

`corBand`: banded correlation structure.

`corCompSymm`: compound symmetry structure corresponding to a constant correlation.

`corExp`: exponential spatial correlation.

`corGaus`: Gaussian spatial correlation.

`corLin`: linear spatial correlation.

`corRatio`: Rational Quadratic spatial correlation.

`corSpher`: spherical spatial correlation.

`corSymm`: general correlation matrix, with no additional structure.

NOTE

Users may define their own `corStruct` classes by specifying a constructor function and, at a minimum, methods for the functions `corMatrix` and `coef`. For examples of these functions, see the methods for classes `corSymm` and `corAR1`.

SEE ALSO

`corAR1`, `corARMA`, `corBand`, `corCAR1`, `corCompSymm`, `corExp`, `corGaus`, `corLin`, `corRatio`, `corSpher`, `corSymm`

This function is a constructor for the `corCompSymm` class, representing a compound symmetry structure corresponding to uniform correlation. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corCompSymm(value, form, fixed)
```

ARGUMENTS

value: the correlation between any two correlated observations. Defaults to 0.

form: a one sided formula of the form $\sim t$, or $\sim t \mid g$, specifying a time covariate t and, optionally, a grouping factor g . When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.

fixed: an optional logical value indicating whether the coefficient should be allowed to vary in the optimization, or kept fixed at its initial value. Defaults to `FALSE`, in which case the coefficient is allowed to vary.

VALUE

an object of class `corCompSymm`, representing a compound symmetry correlation structure.

REFERENCES

Milliken, G. A. and Johnson, D. E. (1992) "Analysis of Messy Data, Volume I: Designed Experiments", Van Nostrand Reinhold.

SEE ALSO

`initialize.corStruct`

EXAMPLE

```
## covariate is observation order and grouping factor is Subject
cs1 <- corCompSymm(0.5, form = ~ 1 | Subject)
```

This function is a constructor for the `corExp` class, representing an exponential spatial correlation structure. Letting d denote the range and n denote the nugget effect, the correlation between two observations a distance r apart is $\exp(-r/d)$ when no nugget effect is present and $n \exp(-r/d)$ when a nugget effect is assumed. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corExp(value, form, nugget, metric, fixed)
```

ARGUMENTS

value: an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the Exponential correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90% of the minimum distance and a nugget ratio of 0.9 being assigned to the parameters when object is initialized.

form: a one sided formula of the form $\sim S_1 + \dots + S_p$, or $\sim S_1 + \dots + S_p \mid g$, specifying spatial covariates S_1 through S_p and, optionally, a grouping factor g . When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.

nugget: an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.

metric: an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".

fixed: an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corExp`, also inheriting from class `corSpatial`, representing an exponential spatial correlation structure.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
Littel, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`initialize.corStruct`, `dist`

EXAMPLE

```
spl <- corExp(form = ~ x + y + z)
```

corFactor	Factor of a Correlation Matrix	corFactor
------------------	--------------------------------	------------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include all `corStruct` classes.

```
corFactor(object, ...)
```

ARGUMENTS

`object`: an object from which a correlation matrix can be extracted.

`...`: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

EXAMPLE

```
## see the method function documentation
```


This method function extracts a transpose inverse square-root factor, or a series of transpose inverse square-root factors, of the correlation matrix, or list of correlation matrices, represented by `object`. Letting Σ denote a correlation matrix, a square-root factor of Σ is any square matrix L such that $\Sigma = L^t L$. This method extracts L^{-t} .

```
corFactor(object)
```

ARGUMENTS

`object`: an object inheriting from class `corStruct` representing a correlation structure, which must have been initialized (using `initialize`).

VALUE

If the correlation structure does not include a grouping factor, the returned value will be a vector with a transpose inverse square-root factor of the correlation matrix associated with `object` stacked column-wise. If the correlation structure includes a grouping factor, the returned value will be a vector with transpose inverse square-root factors of the correlation matrices for each group, stacked by group and stacked column-wise within each group.

NOTE

This method function is used intensively in optimization algorithms and its value is returned as a vector for efficiency reasons. The `corMatrix` method function can be used to obtain transpose inverse square-root factors in matrix form.

SEE ALSO

```
corMatrix.corStruct, recalc.corStruct, initialize.corStruct
```

EXAMPLE

```
cs1 <- corAR1(form = ~ 1 | Subject)
cs1 <- initialize(cs1, data = Orthodont)
corFactor(cs1)
```

This function is a constructor for the `corGaus` class, representing a Gaussian spatial correlation structure. Letting d denote the range and n denote the nugget effect, the correlation between two observations a distance r apart is $\exp(-(r/d)^2)$ when no nugget effect is present and $n \exp(-(r/d)^2)$ when a nugget effect is assumed. Objects created using this constructor need to be later initialized using the appropriate `initialize` method.

```
corGaus(value, form, nugget, metric, fixed)
```

ARGUMENTS

- value:** an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the Gaussian correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90% of the minimum distance and a nugget ratio of 0.9 being assigned to the parameters when object is initialized.
- form:** a one sided formula of the form $\sim S_1 + \dots + S_p$, or $\sim S_1 + \dots + S_p \mid g$, specifying spatial covariates S_1 through S_p and, optionally, a grouping factor g . When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- nugget:** an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.
- metric:** an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".
- fixed:** an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corGaus`, also inheriting from class `corSpatial`, representing a Gaussian spatial correlation structure.

REFERENCES

- Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
 Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
 Littell, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`initialize.corStruct`, `dist`

EXAMPLE

```
spl <- corGaus(form = ~ x + y + z)
```

corLin	Linear Correlation Structure	corLin
---------------	------------------------------	---------------

This function is a constructor for the `corLin` class, representing a linear spatial correlation structure. Letting d denote the range and n denote the nugget effect, the correlation between two observations a distance $r < d$ apart is $1 - (r/d)$ when no nugget effect is present and $n(1 - (r/d))$ when a nugget effect is assumed. If $r \geq d$ the correlation is zero. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corLin(value, form, nugget, metric, fixed)
```

ARGUMENTS

- value**: an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the Linear correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90% of the minimum distance and a nugget ratio of 0.9 being assigned to the parameters when object is initialized.
- form**: a one sided formula of the form `~S1+...+Sp`, or `~S1+...+Sp | g`, specifying spatial covariates `S1` through `Sp` and, optionally, a grouping factor `g`. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to `~1`, which corresponds to using the order of the observations in the data as a covariate, and no groups.
- nugget**: an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.

metric: an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".

fixed: an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to FALSE, in which case the coefficients are allowed to vary.

VALUE

an object of class `corLin`, also inheriting from class `corSpatial`, representing a linear spatial correlation structure.

REFERENCES

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
 Littell, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`initialize.corStruct`, `dist`

EXAMPLE

```
spl <- corLin(form = ~ x + y)
```

corMatrix	Extract Correlation Matrix	corMatrix
------------------	----------------------------	------------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include all `corStruct` classes.

```
corMatrix(object, ...)
```

ARGUMENTS

object: an object for which a correlation matrix can be extracted.

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

EXAMPLE

```
## see the method function documentation
```

This method function extracts the correlation matrix (or its transpose inverse square-root factor), or list of correlation matrices (or their transpose inverse square-root factors) corresponding to `covariate` and `object`. Letting Σ denote a correlation matrix, a square-root factor of Σ is any square matrix L such that $\Sigma = L^t L$. When `corr = FALSE`, this method extracts L^{-t} .

```
corMatrix(object, covariate, corr)
```

ARGUMENTS

`object`: an object inheriting from class `corStruct` representing a correlation structure.

`covariate`: an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to `getCovariate(object)`.

`corr`: a logical value. If `TRUE` the function returns the correlation matrix, or list of correlation matrices, represented by `object`. If `FALSE` the function returns a transpose inverse square-root of the correlation matrix, or a list of transpose inverse square-root factors of the correlation matrices.

VALUE

If `covariate` is a vector (matrix), the returned value will be an array with the corresponding correlation matrix (or its transpose inverse square-root factor). If the `covariate` is a list of vectors (matrices), the returned value will be a list with the correlation matrices (or their transpose inverse square-root factors) corresponding to each component of `covariate`.

SEE ALSO

```
corFactor.corStruct, initialize.corStruct
```

EXAMPLE

```
cs1 <- corAR1(0.3)
corMatrix(cs1, covariate = 1:4)
corMatrix(cs1, covariate = 1:4, corr = F)
```

corMatrix.pdMat

pdMat Correlation Matrix

corMatrix.pdMat

The correlation matrix corresponding to the positive-definite matrix represented by `object` is obtained.

```
corMatrix(object)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive definite matrix.

VALUE

the correlation matrix corresponding to the positive-definite matrix represented by `object`.

SEE ALSO

`as.matrix.pdMat`, `pdMatrix`

EXAMPLE

```
pd1 <- pdSymm(diag(1:4))
corMatrix(pd1)
```

corMatrix.reStruct

reStruct Correlation Matrix

corMatrix.reStruct

This method function extracts the correlation matrices corresponding to the `pdMat` elements of `object`.

```
corMatrix(object)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a list with components given by the correlation matrices corresponding to the elements of `object`.

SEE ALSO

`as.matrix.reStruct`, `reStruct`, `pdMat`

EXAMPLE

```
rs1 <- reStruct(pdSymm(diag(3), form=~ Sex+age, data=Orthodont))
corMatrix(rs1)
```

This function is a constructor for the `corRatio` class, representing a Rational Quadratic spatial correlation structure. Letting d denote the range and n denote the nugget effect, the correlation between two observations a distance r apart is $(r/d)^2 / (1 + (r/d)^2)$ when no nugget effect is present and $(1-n)(r/d)^2 / (1 + (r/d)^2)$ when a nugget effect is assumed. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corRatio(value, form, nugget, metric, fixed)
```

ARGUMENTS

- value:** an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the Rational Quadratic correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between two observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90 being assigned to the parameters when object is initialized.
- form:** a one sided formula of the form $\sim S_1 + \dots + S_p$, or $\sim S_1 + \dots + S_p \mid g$, specifying spatial covariates S_1 through S_p and, optionally, a grouping factor g . When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- nugget:** an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.
- metric:** an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".
- fixed:** an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial value. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corRatio`, also inheriting from class `corSpatial`, representing a rational quadratic spatial correlation structure.

REFERENCES

- Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
 Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
 Littell, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`initialize.corStruct, dist`

EXAMPLE

```
spl <- corRatio(form = ~ x + y + z)
```

corSpatial	Spatial Correlation Structure	corSpatial
-------------------	-------------------------------	-------------------

This function is a constructor for the `corSpatial` class, representing a spatial correlation structure. This class is "virtual", having four "real" classes, corresponding to specific spatial correlation structures, associated with it: `corExp`, `corGaus`, `corLin`, `corRatio`, and `corSpher`. The returned object will inherit from one of these "real" classes, determined by the `type` argument, and from the "virtual" `corSpatial` class. Objects created using this constructor need to be later initialized using the appropriate `initialize` method.

```
corSpatial(value, form, nugget, type, metric, fixed)
```

ARGUMENTS

- value**: an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the spatial correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90% of the minimum distance and a nugget ratio of 0.9 being assigned to the parameters when object is initialized.
- form**: a one sided formula of the form `~S1+...+Sp`, or `~S1+...+Sp | g`, specifying spatial covariates `S1` through `Sp` and, optionally, a grouping factor `g`. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to `~1`, which corresponds to using the order of the observations in the data as a covariate, and no groups.
- nugget**: an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.

- type:** an optional character string specifying the desired type of correlation structure. Available types include "spherical", "exponential", "gaussian", and "linear". See the documentation on the functions `corSpher`, `corExp`, `corGaus`, and `corLin` for a description of these correlation structures. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "spherical".
- metric:** an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".
- fixed:** an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class determined by the `type` argument and also inheriting from class `corSpatial`, representing a spatial correlation structure.

REFERENCES

- Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
 Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
 Littell, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`corExp`, `corGaus`, `corLin`, `corSpher`, `initialize.corStruct`, `dist`

EXAMPLE

```
spl <- corSpatial(form = ~ x + y + z, type = "g", metric = "man")
```

This function is a constructor for the `corSpher` class, representing a spherical spatial correlation structure. Letting d denote the range and n denote the nugget effect, the correlation between two observations a distance $r < d$ apart is $1 - 1.5(r/d) + 0.5(r/d)^3$ when no nugget effect is present and $n(1 - 1.5(r/d) + 0.5(r/d)^3)$ when a nugget effect is assumed. If $r \geq d$ the correlation is zero. Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corSpher(value, form, nugget, metric, fixed)
```

ARGUMENTS

- `value`: an optional vector with the parameter values in constrained form. If `nugget` is `FALSE`, `value` can have only one element, corresponding to the "range" of the Spherical correlation structure, which must be greater than zero. If `nugget` is `TRUE`, meaning that a nugget effect is present, `value` can contain one or two elements, the first being the "range" and the second the "nugget effect" (one minus the correlation between observations taken arbitrarily close together); the first must be greater than zero and the second must be between zero and one. Defaults to `numeric(0)`, which results in a range of 90% of the minimum distance and a nugget ratio of 0.9 being assigned to the parameters when object is initialized.
- `form`: a one sided formula of the form $\sim S_1 + \dots + S_p$, or $\sim S_1 + \dots + S_p \mid g$, specifying spatial covariates S_1 through S_p and, optionally, a grouping factor g . When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1 , which corresponds to using the order of the observations in the data as a covariate, and no groups.
- `nugget`: an optional logical value indicating whether a nugget effect is present. Defaults to `FALSE`.
- `metric`: an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".
- `fixed`: an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corSpher`, also inheriting from class `corSpatial`, representing a spherical spatial correlation structure.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.
Littel, Milliken, Stroup, and Wolfinger (1996) "SAS Systems for Mixed Models", SAS Institute.

SEE ALSO

`initialize.corStruct`, `dist`

EXAMPLE

```
spl <- corSpher(form = ~ x + y)
```

corStrat	Stratified Correlation Structure	corStrat
-----------------	----------------------------------	-----------------

This function is a constructor for the `corStrat` class, representing a set of correlation structures corresponding to different strata, defined by `strata`. Different `corStruct` classes can be used for different strata. The number of coefficients associated with this class is equal to the sum of the number of coefficients of each `corStruct` object it includes. Objects created using this constructor must later be initialized using the appropriate `initialize` method.

```
corStrat(value, form, strata)
```

ARGUMENTS

- value:** a `corStruct` object, or a list of `corStruct` objects. If given as a single object, it is repeated for all strata.
- strata:** a one-sided formula specifying the stratification variable for the different `corStruct` objects represented by `object`. Its right hand side must evaluate to a factor or an integer.
- form:** an optional one sided formula of the form `~t`, or `~t | g`, specifying a time covariate `t` and, optionally, a grouping factor `g`. The same formula is used for all `corStruct` represented in `object`. When a grouping factor is present, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated.

VALUE

a `corStrat` object representing a set of correlation structures corresponding to different strata.

SEE ALSO

`initialize.corStruct`

EXAMPLE

```
cs1 <- corStrat(corAR1(0.2, form = ~ 1 | Subject), strata = ~ Sex)
cs1 <- corStrat(list(Male = corAR1(), Female = corARMA(p=2)),
                 form = ~ 1 | Subject, strata = ~ Sex)
```

corSymm	General Correlation Structure	corSymm
----------------	-------------------------------	----------------

This function is a constructor for the `corSymm` class, representing a general correlation structure. The internal representation of this structure, in terms of unconstrained parameters, uses the spherical parametrization defined in Pinheiro and Bates (1996). Objects created using this constructor must be later initialized using the appropriate `initialize` method.

```
corSymm(value, form, fixed)
```

ARGUMENTS

value: an optional vector with the parameter values. Default is `numeric(0)`, which results in a vector of zeros of appropriate dimension being assigned to the parameters when object is initialized (corresponding to an identity correlation structure).

form: a one sided formula of the form `~t`, or `~t | g`, specifying a time covariate `t` and, optionally, a grouping factor `g`. A covariate for this correlation structure must such that its unique values form a sequence of consecutive integers. When a grouping factor is present in `form`, the correlation structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to `~1`, which corresponds to using the order of the observations in the data as a covariate, and no groups.

fixed: an optional logical value indicating whether the coefficients should be allowed to vary in the optimization, or kept fixed at their initial values. Defaults to `FALSE`, in which case the coefficients are allowed to vary.

VALUE

an object of class `corSymm` representing a general correlation structure.

REFERENCES

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

SEE ALSO

```
initialize.corStruct
```

EXAMPLE

```
## covariate is observation order and grouping factor is Subject
cs1 <- corSymm(form = ~ 1 | Subject)
```

This method function converts `object` to a `corSymmNat` object, represented in a "natural" parameterization. If `unconstrained = TRUE`, an unconstrained parameterization based on the logit of the correlations ($\log((1 + r)/(1 - r))$, with r denoting the correlation) is used. If `unconstrained` is set to `FALSE`, the (constrained) correlations are used to parameterize the returned `corSymmNat` object. This `corStruct` class does not define an unrestricted parameterization and, thus, should NOT be used for optimization. It is mostly used for deriving approximate confidence intervals on parameters following the optimization of an objective function.

```
corSymmNat(object, unconstrained)
```

ARGUMENTS

`object`: an object inheriting from class `corSymm`, representing a general correlation structure, whose representation is to be converted to a natural parameterization.

`unconstrained`: an optional logical value indicating whether the natural parameterization should be unconstrained. Defaults to `TRUE`.

VALUE

an object of class `corSymmNat` representing a general correlation structure in natural parameterization.

SEE ALSO

```
asNatural.corSymmNat, corSymm
```

EXAMPLE

```
cst1 <- corSymm(form = ~ 1|Subject)
cst1 <- initialize(cst1, data = Orthodont)
corSymmNat(cst1)
```

covariate<-

Assign Covariate Values

covariate<-

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include all `varFunc` classes.

```
covariate(object) <- value
```

ARGUMENTS

object: any object with a `covariate` component.

value: a value to be assigned to the covariate associated with `object`.

VALUE

will depend on the method function; see the appropriate documentation.

SEE ALSO

`getCovariate`

EXAMPLE

```
## see the method function documentation
```

covariate<-.varFunc

Assign varFunc Covariate

covariate<-.varFunc

The covariate(s) used in the calculation of the weights of the variance function represented by `object` is (are) replaced by `value`. If `object` has been initialized, `value` must have the same dimensions as `getCovariate(object)`.

```
covariate(object) <- value
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

value: a value to be assigned to the covariate associated with `object`.

VALUE

a `varFunc` object similar to `object`, but with its `covariate` attribute replaced by `value`.

SEE ALSO

`getCovariate.varFunc`

EXAMPLE

```
vfl <- varPower(1.1, form = ~ age)
covariate(vfl) <- Orthodont[["age"]]
```

Dim	Extract Dimensions from an Object	Dim
------------	-----------------------------------	------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corSpatial`, `corStruct`, `pdCompSymm`, `pdDiag`, `pdIdent`, `pdMat`, and `pdSymm`.

```
Dim(object, ...)
```

ARGUMENTS

`object`: any object for which dimensions can be extracted.

`...`: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

NOTE

If `dim` allowed more than one argument, there would be no need for this generic function.

SEE ALSO

`Dim.pdMat`, `Dim.corStruct`

EXAMPLE

```
## see the method function documentation
```

Dim.corSpatial	Dimensions of a <code>corSpatial</code> Object	Dim.corSpatial
-----------------------	--	-----------------------

if `groups` is missing, it returns the `Dim` attribute of `object`; otherwise, calculates the dimensions associated with the grouping factor.

```
Dim(object, groups)
```

ARGUMENTS

`object`: an object inheriting from class `corSpatial`, representing a spatial correlation structure.

`groups`: an optional factor defining the grouping of the observations; observations within a group are correlated and observations in different groups are uncorrelated.

VALUE

a list with components:

`N`: length of `groups`

M: number of groups

spClass: an integer representing the spatial correlation class; 0 = user defined class, 1 = corSpher, 2 = corExp, 3 = corGaus, 4 = corLin

sumLenSq: sum of the squares of the number of observations per group

len: an integer vector with the number of observations per group

start: an integer vector with the starting position for the distance vectors in each group, beginning from zero

SEE ALSO

Dim, Dim.corStruct

EXAMPLE

```
Dim(corGaus(), getGroups(Orthodont))
```

Dim.corStruct	Dimensions of a corStruct Object	Dim.corStruct
---------------	----------------------------------	---------------

if groups is missing, it returns the Dim attribute of object; otherwise, calculates the dimensions associated with the grouping factor.

```
Dim(object, groups)
```

ARGUMENTS

object: an object inheriting from class corStruct, representing a correlation structure.

groups: an optional factor defining the grouping of the observations; observations within a group are correlated and observations in different groups are uncorrelated.

VALUE

a list with components:

N: length of groups

M: number of groups

maxLen: maximum number of observations in a group

sumLenSq: sum of the squares of the number of observations per group

len: an integer vector with the number of observations per group

start: an integer vector with the starting position for the observations in each group, beginning from zero

SEE ALSO

Dim, Dim.corSpatial

EXAMPLE

```
Dim(corAR1(), getGroups(Orthodont))
```

Dim.pdMat	Dimensions of a pdMat Object	Dim.pdMat
------------------	------------------------------	------------------

This method function returns the dimensions of the matrix represented by object.

```
Dim(object)
```

ARGUMENTS

object: an object inheriting from class `pdMat`, representing a positive-definite matrix.

VALUE

an integer vector with the number of rows and columns of the matrix represented by object.

SEE ALSO

`Dim`

EXAMPLE

```
Dim(pdSymm(diag(3)))
```

fitted.gls	Extract gls Fitted Values	fitted.gls
-------------------	---------------------------	-------------------

The fitted values for the linear model represented by `object` are extracted.

```
fitted(object)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

VALUE

a vector with the fitted values for the linear model represented by object.

SEE ALSO

`gls`, `residuals.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
fitted(fml)
```

fitted.glsStruct

Calculate glsStruct Fitted Values

fitted.glsStruct

The fitted values for the linear model represented by `object` are extracted.

```
fitted(object, glsFit)
```

ARGUMENTS

`object`: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

`glsFit`: an optional list with components `logLik` (log-likelihood), `beta` (coefficients), `sigma` (standard deviation for error term), `varBeta` (coefficients' covariance matrix), `fitted` (fitted values), and `residuals` (residuals). Defaults to `attr(object, "glsFit")`.

VALUE

a vector with the fitted values for the linear model represented by `object`.

NOTE

This method function is primarily used inside `gls` and `fitted.gls`.

SEE ALSO

`gls`, `fitted.gls`, `residuals.glsStruct`

fitted.gnls

Extract gnls Fitted Values

fitted.gnls

The fitted values for the nonlinear model represented by `object` are extracted.

```
fitted(object)
```

ARGUMENTS

`object`: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

VALUE

a vector with the fitted values for the nonlinear model represented by `object`.

SEE ALSO

`gnls`, `residuals.gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
fitted(fml)
```

fitted.gnlsStruct	Calculate gnlsStruct Fitted Values	fitted.gnlsStruct
--------------------------	------------------------------------	--------------------------

The fitted values for the nonlinear model represented by `object` are extracted.

```
fitted(object)
```

ARGUMENTS

`object`: an object inheriting from class `gnlsStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects, and attributes specifying the underlying nonlinear model and the response variable.

VALUE

a vector with the fitted values for the nonlinear model represented by `object`.

NOTE

This method function is primarily used inside `gnls` and `fitted.gnls`.

SEE ALSO

`gnls`, `fitted.gnls`, `residuals.gnlsStruct`

fitted.lmList	Extract lmList Fitted Values	fitted.lmList
----------------------	------------------------------	----------------------

The fitted values are extracted from each `lm` component of `object` and arranged into a list with as many components as `object`, or combined into a single vector.

```
fitted(object, subset, asList)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

`subset`: an optional character or integer vector naming the `lm` components of `object` from which the fitted values are to be extracted. Default is `NULL`, in which case all components are used.

`asList`: an optional logical value. If `TRUE`, the returned object is a list with the fitted values split by groups; else the returned value is a vector. Defaults to `FALSE`.

VALUE

a list with components given by the fitted values of each `lm` component of `object`, or a vector with the fitted values for all `lm` components of `object`.

SEE ALSO

`lmList`, `residuals.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
fitted(fml)
```

fitted.lmList

Extract lmList Fitted Values

fitted.lmList

The fitted values are extracted from each `lm` component of `object` and arranged into a list with as many components as `object`, or combined into a single vector.

```
fitted(object, subset, asList)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

subset: an optional character or integer vector naming the `lm` components of `object` from which the fitted values are to be extracted. Default is `NULL`, in which case all components are used.

asList: an optional logical value. If `TRUE`, the returned object is a list with the fitted values split by groups; else the returned value is a vector. Defaults to `FALSE`.

VALUE

a list with components given by the fitted values of each `lm` component of `object`, or a vector with the fitted values for all `lm` components of `object`.

SEE ALSO

```
lmList, residuals.lmList
```

EXAMPLE

```
fml <- lmList(distance ~ age, Orthodont, groups = ~ Subject)
fitted(fml)
```

The fitted values at level *i* are obtained by adding together the population fitted values (based only on the fixed effects estimates) and the estimated contributions of the random effects to the fitted values at grouping levels less or equal to *i*. The resulting values estimate the best linear unbiased predictions (BLUPs) at level *i*.

```
fitted(object, level, asList)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

level: an optional integer vector giving the level(s) of grouping to be used in extracting the fitted values from **object**. Level values increase from outermost to innermost grouping, with level zero corresponding to the population fitted values. Defaults to the highest or innermost level of grouping.

asList: an optional logical value. If `TRUE` and a single value is given in **level**, the returned object is a list with the fitted values split by groups; else the returned value is either a vector or a data frame, according to the length of **level**. Defaults to `FALSE`.

VALUE

if a single level of grouping is specified in **level**, the returned value is either a list with the fitted values split by groups (`asList = TRUE`) or a vector with the fitted values (`asList = FALSE`); else, when multiple grouping levels are specified in **level**, the returned object is a data frame with columns given by the fitted values at different levels and the grouping factors.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`lme`, `residuals.lme`

EXAMPLE

```
fml <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)
fitted(fml, level = 0:1)
```

The fitted values at level *i* are obtained by adding together the population fitted values (based only on the fixed effects estimates) and the estimated contributions of the random effects to the fitted values at grouping levels less or equal to *i*. The resulting values estimate the best linear unbiased predictions (BLUPs) at level *i*.

```
fitted(object, levels, lmeFit, conLin)
```

ARGUMENTS

- object**: an object inheriting from class `lmeStruct`, representing a list of linear mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.
- level**: an optional integer vector giving the level(s) of grouping to be used in extracting the fitted values from `object`. Level values increase from outermost to innermost grouping, with level zero corresponding to the population fitted values. Defaults to the highest or innermost level of grouping.
- lmeFit**: an optional list with components `beta` and `b` containing respectively the fixed effects estimates and the random effects estimates to be used to calculate the fitted values. Defaults to `attr(object, "lmeFit")`.
- conLin**: an optional condensed linear model object, consisting of a list with components `"Xy"`, corresponding to a regression matrix (*x*) combined with a response vector (*y*), and `"logLik"`, corresponding to the log-likelihood of the underlying lme model. Defaults to `attr(object, "conLin")`.

VALUE

if a single level of grouping is specified in `level`, the returned value is a vector with the fitted values at the desired level; else, when multiple grouping levels are specified in `level`, the returned object is a matrix with columns given by the fitted values at different levels.

NOTE

This method function is primarily used inside `lme` and `fitted.lme`.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

```
lme, fitted.lme, residuals.lmeStruct
```

The fitted values at level *i* are obtained by adding together the contributions from the estimated fixed effects and the estimated random effects at levels less or equal to *i* and evaluating the model function at the resulting estimated parameters. The resulting values estimate the predictions at level *i*.

```
fitted(object, levels, nlmeFit, conLin)
```

ARGUMENTS

- object**: an object inheriting from class `nlmeStruct`, representing a list of mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects, plus attributes specifying the underlying nonlinear model and the response variable.
- level**: an optional integer vector giving the level(s) of grouping to be used in extracting the fitted values from `object`. Level values increase from outermost to innermost grouping, with level zero corresponding to the population fitted values. Defaults to the highest or innermost level of grouping.
- conLin**: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying nlme model. Defaults to `attr(object, "conLin")`.

VALUE

if a single level of grouping is specified in `level`, the returned value is a vector with the fitted values at the desired level; else, when multiple grouping levels are specified in `level`, the returned object is a matrix with columns given by the fitted values at different levels.

NOTE

This method function is primarily used inside `nlme` and `fitted.nlme`

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`nlme`, `fitted.nlme`, `residuals.nlmeStruct`

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `lmList` and `lme`.

```
fixed.effects(object, ...)
fixef(object, ...)
```

ARGUMENTS

object: any fitted model object from which fixed effects estimates can be extracted.

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`fixef.lmList`, `fixef.lme`

EXAMPLE

```
## see the method function documentation
```

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `lmList` and `lme`.

```
fixef(object, ...)
fixed.effects(object, ...)
```

ARGUMENTS

object: any fitted model object from which fixed effects estimates can be extracted.

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`fixef.lmList`, `fixef.lme`

EXAMPLE

```
## see the method function documentation
```


fixef.lmList

Extract lmList Fixed Effects

fixef.lmList

The average of the coefficients corresponding to the `lm` components of `object` is calculated.

```
fixef(object)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

VALUE

a vector with the average of the individual `lm` coefficients in `object`.

SEE ALSO

`lmList`, `ranef.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
fixef(fml)
```

fixef.lme

Extract lme Fixed Effects

fixef.lme

The fixed effects estimates corresponding to the linear mixed-effects model represented by `object` are returned.

```
fixef(object)
```

ARGUMENTS

`object`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

VALUE

a vector with the fixed effects estimates corresponding to `object`.

SEE ALSO

`coef.lme`, `ranef.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
fixef(fml)
```

formula.corStruct

Extract corStruct Formula

formula.corStruct

This method function extracts the formula associated with a `corStruct` object, in which the covariate and the grouping factor, if any is present, are defined.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `corStruct` representing a correlation structure.

VALUE

an object of class `formula` specifying the covariate and the grouping factor, if any is present, associated with `object`.

SEE ALSO

`formula`

EXAMPLE

```
cs1 <- corCAR1(form = ~ Time | Mare)
formula(cs1)
```

formula.gls

Extract gls Formula

formula.gls

This method function extracts the linear model formula associated with `object`.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

VALUE

a two-sided linear formula specifying the linear model used to obtain `object`.

SEE ALSO

`gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
formula(fml)
```

This method function extracts the nonlinear model formula associated with object.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

VALUE

a two-sided formula specifying the nonlinear model used to obtain object.

SEE ALSO

`gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
formula(fml)
```

This method function extracts the display formula associated with a `groupedData` object. This is a two-sided formula of the form `resp ~ cov | group`, where `resp` is the response, `cov` is the primary covariate, and `group` is the grouping structure.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `groupedData`.

VALUE

a two-sided formula with a conditioning expression, representing the display formula for object.

SEE ALSO

`groupedData`

EXAMPLE

```
formula(Orthodont)
```

formula.lmList

Extract lmList Object Formula

formula.lmList

This method function extracts the common linear model formula associated with each `lm` component of `object`.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

VALUE

a two-sided linear formula specifying the linear model used to obtain the `lm` components of `object`.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
formula(fml)
```

formula.lme

Extract lme Formula

formula.lme

This method function extracts the fixed effects model formula associated with `object`.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

VALUE

a two-sided linear formula specifying the fixed effects model used to obtain `object`.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
formula(fml)
```

formula.modelStruct

Extract modelStructFormula

formula.modelStruct

This method function extracts a formula from each of the components of object, returning a list of formulas.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

VALUE

a list with the formulas of each component of object.

SEE ALSO

`formula`

EXAMPLE

```
lms1 <- lmeStruct(reStruct = reStruct(pdDiag(diag(2), ~ age)),  
  corStruct = corAR1(0.3))  
formula(lms1)
```

formula.nlme

Extract nlme Object Formula

formula.nlme

This method function extracts the nonlinear model formula associated with object.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `nlme`, representing a fitted nonlinear mixed-effects model.

VALUE

a two-sided nonlinear formula specifying the model used to obtain object.

SEE ALSO

`nlme`

EXAMPLE

```
fm1 <- nlme(weight ~ SSlogis(Time, Asym, xmid, scal),  
  data = Soybean, fixed = Asym + xmid + scal ~ 1,  
  start = c(18, 52, 7.5))  
formula(fm1)
```

formula.nlsList

Extract nlsList Object Formula

formula.nlsList

This method function extracts the common nonlinear model formula associated with each `nls` component of `object`.

```
formula(object)
```

ARGUMENTS

`object`: an object inheriting from class `nlsList`, representing a list of `nls` objects with a common model.

VALUE

a two-sided nonlinear formula specifying the model used to obtain the `nls` components of `object`.

SEE ALSO

`nlsList`

EXAMPLE

```
fml <- nlsList(weight ~ SSlogis(Time, Asym, xmid, scal)|Plot,
               data=Soybean)
formula(fml)
```

formula.nls

Extract Model Formula from nls Object

formula.nls

Returns the model used to fit `object`.

```
formula(object)
```

ARGUMENTS

`object`: an object inheriting from class `nls`, representing a non-linear least squares fit.

VALUE

a formula representing the model used to obtain `object`.

SEE ALSO

`nls`, `formula`

EXAMPLE

```
fml <- nls(circumference ~ A/(1+exp((B-age)/C)), Orange,
          start = list(A=160, B=700, C = 350))
formula(fml)
```

formula.pdBlocked

Extract pdBlocked Formula

formula.pdBlocked

The `formula` attributes of the `pdMat` elements of `object` are extracted and returned as a list, in case `asList=TRUE`, or converted to a single one-sided formula when `asList=FALSE`. If the `pdMat` elements do not have a `formula` attribute, a `NULL` value is returned.

```
formula(object, asList)
```

ARGUMENTS

object: an object inheriting from class `pdBlocked`, representing a positive definite block diagonal matrix.

asList: an optional logical value. If `TRUE`, a list with the formulas for the individual block diagonal elements of `object` is returned; else, if `FALSE`, a one-sided formula combining all individual formulas is returned. Defaults to `FALSE`.

VALUE

a list of one-sided formulas, or a single one-sided formula, or `NULL`.

SEE ALSO

`pdBlocked`, `pdMat`

EXAMPLE

```
pd1 <- pdBlocked(list(~ age, ~ Sex - 1))
formula(pd1)
formula(pd1, asList = TRUE)
```

formula.pdMat

Extract pdMat Formula

formula.pdMat

This method function extracts the formula associated with a `pdMat` object, in which the column and row names are specified.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `pdMat`, representing a positive definite matrix.

VALUE

if `object` has a `formula` attribute, its value is returned, else `NULL` is returned.

NOTE

Because factors may be present in `formula(object)`, the `pdMat` object needs to have access to a data frame where the variables named in the formula can be evaluated, before it can resolve its row and column names from the formula.

SEE ALSO

`pdMat`

EXAMPLE

```
pd1 <- pdSymm(~ Sex*age)
formula(pd1)
```

formula.reStruct	Extract reStruct Formula	formula.reStruct
-------------------------	--------------------------	-------------------------

This method function extracts a formula from each of the components of object, returning a list of formulas.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a list with the formulas of each component of object.

SEE ALSO

`formula`

EXAMPLE

```
rs1 <- reStruct(list(A = pdDiag(diag(2), ~ age), B = ~ 1))
formula(rs1)
```

formula.varFunc	Extract varFunc Formula	formula.varFunc
------------------------	-------------------------	------------------------

This method function extracts the formula associated with a `varFunc` object, in which covariates and grouping factors are specified.

```
formula(object)
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

VALUE

if object has a `formula` attribute, its value is returned; else `NULL` is returned.

EXAMPLE

```
formula(varPower(form = ~ fitted(.) | Sex))
```


Applies the function to the distinct sets of rows of the data frame defined by groups.

```
gapply(object, which, FUN, form, level, groups, ...)
```

ARGUMENTS

object: an object to which the function will be applied - usually a `groupedData` object or a `data.frame`.

which: an optional character or positive integer vector specifying which columns of `object` should be used with `FUN`. Defaults to all columns in `object`.

FUN: function to apply to the distinct sets of rows of the data frame `object` defined by the values of `groups`.

form: an optional one-sided formula that defines the groups. When this formula is given the right-hand side is evaluated in `object`, converted to a factor if necessary, and the unique levels are used to define the groups. Defaults to `formula(object)`.

level: an optional positive integer giving the level of grouping to be used in an object with multiple nested grouping levels. Defaults to the highest or innermost level of grouping.

groups: an optional factor that will be used to split the rows into groups. Defaults to `getGroups(object, form, level)`.

...: optional additional arguments to the summary function `FUN`. Often it is helpful to specify `na.rm = TRUE`.

VALUE

Returns a data frame with as many rows as there are levels in the `groups` argument.

SEE ALSO

`gsummary`

EXAMPLE

```
## Find number of non-missing "conc" observations for each Subject
gapply( Quinidine, FUN = function(x) sum(!is.na(x$conc)) )
```

getCovariate	Extract Covariate from an Object	getCovariate
---------------------	----------------------------------	---------------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `corStruct`, `corSpatial`, `data.frame`, and `varFunc`.

```
getCovariate(object, form, data)
```

ARGUMENTS

- object**: any object with a covariate component
- form**: an optional one-sided formula specifying the covariate(s) to be extracted. Defaults to `formula(object)`.
- data**: a data frame in which to evaluate the variables defined in `form`.

VALUE
will depend on the method function used; see the appropriate documentation.

SEE ALSO
`getCovariate.corStruct`, `getCovariate.data.frame`, `getCovariate.varFunc`, `getCovariateFormula`

EXAMPLE
see the method function documentation

getCovariate.corStruct	corStruct Covariate	getCovariate.corStruct
-------------------------------	---------------------	-------------------------------

This method function extracts the covariate(s) associated with `object`.

```
getCovariate(object, form, data)
```

ARGUMENTS

- object**: an object inheriting from class `corStruct` representing a correlation structure.
- form**: this argument is included to make the method function compatible with the generic. It will be assigned the value of `formula(object)` and should not be modified.
- data**: an optional data frame in which to evaluate the variables defined in `form`, in case `object` is not initialized and the covariate needs to be evaluated.

VALUE
when the correlation structure does not include a grouping factor, the returned value will be a vector or a matrix with the covariate(s) associated with `object`. If a grouping factor is present, the returned value will be a list of vectors or matrices with the covariate(s) corresponding to each grouping level.

SEE ALSO
`getCovariate`

EXAMPLE

```
cs1 <- corAR1(form = ~ 1 | Subject)
getCovariate(cs1, data = Orthodont)
```

getCovariate.data.frame	Data Frame Covariate	getCovariate.data.frame
--------------------------------	----------------------	--------------------------------

The right hand side of `form`, stripped of any conditioning expression (i.e. an expression following a `|` operator), is evaluated in `object`.

```
getCovariate(object, form)
```

ARGUMENTS

object: an object inheriting from class `data.frame`.

form: an optional formula specifying the covariate to be evaluated in `object`. Defaults to `formula(object)`.

VALUE

the value of the right hand side of `form`, stripped of any conditional expression, evaluated in `object`.

SEE ALSO

`getCovariateFormula`

EXAMPLE

```
getCovariate(Orthodont)
```

getCovariate.varFunc	Extract varFunc Covariate	getCovariate.varFunc
-----------------------------	---------------------------	-----------------------------

This method function extracts the covariate(s) associated with the variance function represented by `object`, if any is present.

```
getCovariate(object)
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

VALUE

if `object` has a `covariate` attribute, its value is returned; else `NULL` is returned.

SEE ALSO

`covariate<- .varFunc`

EXAMPLE

```
vf1 <- varPower(1.1, form = ~ age)
covariate(vf1) <- Orthodont[["age"]]
getCovariate(vf1)
```

getCovariateFormula

Extract Covariates Formula

getCovariateFormula

The right hand side of `formula(object)`, without any conditioning expressions (i.e. any expressions after a `|` operator) is returned as a one-sided formula.

```
getCovariateFormula(object)
```

ARGUMENTS

object: any object from which a formula can be extracted.

VALUE

a one-sided formula describing the covariates associated with `formula(object)`.

SEE ALSO

`getCovariate`

EXAMPLE

```
getCovariateFormula(y ~ x | g)
getCovariateFormula(y ~ x)
```

getData

Extract Data from an Object

getData

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `gls`, `lme`, and `lmList`.

```
getData(object)
```

ARGUMENTS

object: an object from which a `data.frame` can be extracted, generally a fitted model object.

VALUE

will depend on the method function used; see the appropriate documentation.

EXAMPLE

```
## see the method function documentation
```

If present in the calling sequence used to produce `object`, the data frame used to fit the model is obtained.

```
getData(object)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

VALUE

if a data argument is present in the calling sequence that produced `object`, the corresponding data frame (with `na.action` and `subset` applied to it, if also present in the call that produced `object`) is returned; else, `NULL` is returned.

SEE ALSO

`gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), data = Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
getData(fml)
```

If present in the calling sequence used to produce `object`, the data frame used to fit the model is obtained.

```
getData(object)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

VALUE

if a data argument is present in the calling sequence that produced `object`, the corresponding data frame (with `na.action` and `subset` applied to it, if also present in the call that produced `object`) is returned; else, `NULL` is returned.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
getData(fml)
```

If present in the calling sequence used to produce `object`, the data frame used to fit the model is obtained.

```
getData(object)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a linear mixed-effects fitted model.

VALUE

if a `data` argument is present in the calling sequence that produced `object`, the corresponding data frame (with `na.action` and `subset` applied to it, if also present in the call that produced `object`) is returned; else, `NULL` is returned.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), data = Ovary,
          random = ~ sin(2*pi*Time))
getData(fml)
```

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `corStruct`, `data.frame`, `gls`, `lme`, `lmList`, and `varFunc`.

```
getGroups(object, form, level, data)
```

ARGUMENTS

object: any object

form: an optional formula with a conditioning expression on its right hand side (i.e. an expression involving the `|` operator). Defaults to `formula(object)`.

level: a positive integer vector with the level(s) of grouping to be used when multiple nested levels of grouping are present. This argument is optional for most methods of this generic function and defaults to all levels of nesting.

data: a data frame in which to interpret the variables named in `form`. Optional for most methods.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`getGroupsFormula`, `getGroups.data.frame`, `getGroups.gls`, `getGroups.lmList`,
`getGroups.lme`

EXAMPLE

```
## see the method function documentation
```

getGroups.corStruct	Extract corStruct Groups	getGroups.corStruct
----------------------------	--------------------------	----------------------------

This method function extracts the grouping factor associated with `object`, if any is present.

```
getGroups(object, form, data, level)
```

ARGUMENTS

object: an object inheriting from class `corStruct` representing a correlation structure.

form: this argument is included to make the method function compatible with the generic. It will be assigned the value of `formula(object)` and should not be modified.

data: an optional data frame in which to evaluate the variables defined in `form`, in case `object` is not initialized and the grouping factor needs to be evaluated.

level: this argument is included to make the method function compatible with the generic and is not used.

VALUE

if a grouping factor is present in the correlation structure represented by `object`, the function returns the corresponding factor vector; else the function returns `NULL`.

SEE ALSO

`getGroups`

EXAMPLE

```
cs1 <- corAR1(form = ~ 1 | Subject)  
getGroups(cs1, data = Orthodont)
```

Each variable named in the expression after the `|` operator on the right hand side of `form` is evaluated in `object`. If more than one variable is indicated in `level` they are combined into a data frame; else the selected variable is returned as a vector. When multiple grouping levels are defined in `form` and `level > 1`, the levels of the returned factor are obtained by pasting together the levels of the grouping factors of level greater or equal to `level`, to ensure their uniqueness.

```
getGroups(object, form, level)
```

ARGUMENTS

`object`: an object inheriting from class `data.frame`.

`form`: an optional formula with a conditioning expression on its right hand side (i.e. an expression involving the `|` operator). Defaults to `formula(object)`.

`level`: a positive integer vector with the level(s) of grouping to be used when multiple nested levels of grouping are present. Defaults to all levels of nesting.

VALUE

either a data frame with columns given by the grouping factors indicated in `level`, from outer to inner, or, when a single level is requested, a factor representing the selected grouping factor.

SEE ALSO

`getGroupsFormula`

EXAMPLE

```
getGroups(Pixel)
getGroups(Pixel, level = 2)
```


getGroups.gls

Extract gls Object Groups

getGroups.gls

If present, the grouping factor associated to the correlation structure for the linear model represented by `object` is extracted.

```
getGroups(object)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

VALUE

if the linear model represented by `object` incorporates a correlation structure and the corresponding `corStruct` object has a grouping factor, a vector with the group values is returned; else, `NULL` is returned.

SEE ALSO

`gls`, `corClasses`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,  
          correlation = corAR1(form = ~ 1 | Mare))  
getGroups(fml)
```

getGroups.lmList

Extract lmList Object Groups

getGroups.lmList

The grouping factor determining the partitioning of the observations used to produce the `lm` components of `object` is extracted.

```
getGroups(object)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

VALUE

a vector with the grouping factor corresponding to the `lm` components of `object`.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)  
getGroups(fml)
```

The grouping factors corresponding to the linear mixed-effects model represented by `object` are extracted. If more than one level is indicated in `level`, the corresponding grouping factors are combined into a data frame; else the selected grouping factor is returned as a vector.

```
getGroups(object, form, level)
```

ARGUMENTS

`object`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

`form`: this argument is included to make the method function compatible with the generic and is ignored in this method.

`level`: an optional integer vector giving the level(s) of grouping to be extracted from `object`. Defaults to the highest or innermost level of grouping.

VALUE

either a data frame with columns given by the grouping factors indicated in `level`, or, when a single level is requested, a factor representing the selected grouping factor.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(pixel ~ day + day^2, Pixel,
  random = list(Dog = ~ day, Side = ~ 1))
getGroups(fml, level = 1:2)
```

This method function extracts the grouping factor associated with the variance function represented by `object`, if any is present.

```
getGroups(object)
```

ARGUMENTS

`object`: an object inheriting from class `varFunc`, representing a variance function structure.

VALUE

if `object` has a `groups` attribute, its value is returned; else `NULL` is returned.

EXAMPLE

```
vf1 <- varPower(form = ~ age | Sex)
vf1 <- initialize(vf1, Orthodont)
getGroups(vf1)
```

getGroupsFormula

Extract Grouping Formula

getGroupsFormula

The conditioning expression associated with `formula(object)` (i.e. an expression after the `|` operator) is returned either as a named list of one-sided formulas, or a single one-sided formula, depending on the value of `asList`. The components of the returned list are ordered from outermost to innermost level and are named after the grouping factor expression.

```
getGroupsFormula(object, asList)
```

ARGUMENTS

object: any object from which a formula can be extracted.

asList: an optional logical value. If `TRUE` the returned value will be a list of formulas; else, if `FALSE` the returned value will be a one-sided formula. Defaults to `FALSE`.

VALUE

a one-sided formula, or a list of one-sided formulas, with the grouping structure associated with `formula(object)`. If no conditioning expression is present in `formula(object)` a `NULL` value is returned.

SEE ALSO

```
getGroupsFormula.gls, getGroupsFormula.lmList, getGroupsFormula.lme,
getGroupsFormula.reStruct, getGroups
```

EXAMPLE

```
getGroupsFormula(y ~ x | g1/g2)
```

If present, the grouping formula associated with the correlation structure (`corStruct`) of `object` is returned either as a named list with a single one-sided formula, or a single one-sided formula, depending on the value of `asList`. If `object` does not include a correlation structure, or if the correlation structure does not include groups, `NULL` is returned.

```
getGroupsFormula(object, asList)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

`asList`: an optional logical value. If `TRUE` the returned value will be a list of formulas; else, if `FALSE` the returned value will be a one-sided formula. Defaults to `FALSE`.

VALUE

if a correlation structure with groups is included in `object`, a one-sided formula, or a list with a single one-sided formula, with the corresponding grouping structure, else `NULL`.

SEE ALSO

`corClasses`, `getGroups.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          correlation = corAR1(form = ~ 1 | Mare))
getGroupsFormula(fml)
```

getGroupsFormula.lmList	lmList Grouping Formula	getGroupsFormula.lmList
--------------------------------	-------------------------	--------------------------------

A formula representing the grouping factor determining the partitioning of the observations used to produce the `lm` components of `object` is obtained and returned as a list with a single component, or as a one-sided formula.

```
getGroupsFormula(object, asList)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

asList: an optional logical value. If `TRUE` the returned value will be a list of formulas; else, if `FALSE` the returned value will be a one-sided formula. Defaults to `FALSE`.

VALUE

a one-sided formula, or a list with a single one-sided formula, representing the grouping factor corresponding to the `lm` components of `object`.

SEE ALSO

`lmList`, `getGroups.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
getGroupsFormula(fml)
```

getGroupsFormula.lme	lme Grouping Formula	getGroupsFormula.lme
-----------------------------	----------------------	-----------------------------

The grouping formula associated with the random effects structure (`reStruct`) of `object` is returned either as a named list of one-sided formulas, or a single one-sided formula, depending on the value of `asList`. The components of the returned list are ordered from outermost to innermost level and are named after the grouping factor expression.

```
getGroupsFormula(object, asList)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

asList: an optional logical value. If `TRUE` the returned value will be a list of formulas; else, if `FALSE` the returned value will be a one-sided formula. Defaults to `FALSE`.

VALUE

a one-sided formula, or a list of one-sided formulas, with the grouping structure associated with the random effects structure of `object`.

SEE ALSO

`reStruct`, `getGroups.lme`

EXAMPLE

```
fml <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)
getGroupsFormula(fml)
```

getGroupsFormula.reStruct reStruct Grouping Formula **getGroupsFormula.reStruct**

The names of the object components are used to construct a one-sided formula, or a named list of formulas, depending on the value of `asList`. The components of the returned list are ordered from outermost to innermost level.

```
getGroupsFormula(object, asList)
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

asList: an optional logical value. If `TRUE` the returned value will be a list of formulas; else, if `FALSE` the returned value will be a one-sided formula. Defaults to `FALSE`.

VALUE

a one-sided formula, or a list of one-sided formulas, with the grouping structure associated with `object`.

SEE ALSO

`reStruct`, `getGroups`

EXAMPLE

```
rsl <- reStruct(list(A = pdDiag(diag(2), ~ age), B = ~ 1))
getGroupsFormula(rsl)
```

getInitial

Get Initial Parameter Estimates

getInitial

This function evaluates initial parameter estimates for a nonlinear regression model. If `data` is a parameterized data frame or `pframe` object, its `parameters` attribute is returned. Otherwise the object is examined to see if it contains a call to a `selfStart` object whose `initial` attribute can be evaluated.

```
getInitial(object, data, ...)
```

ARGUMENTS

`object`: a formula or a `selfStart` model that defines a nonlinear regression model

`data`: a data frame in which the expressions in the formula or arguments to the `selfStart` model can be evaluated

`...`: optional additional arguments

VALUE

A named numeric vector or list of starting estimates for the parameters. The construction of many `selfStart` models is such that these "starting" estimates are, in fact, the converged parameter estimates.

SEE ALSO

`nls`, `nlsList`, `selfStart`, `selfStart.default`, `selfStart.formula`

EXAMPLE

```
PurTrt <- Puromycin[ Puromycin$state == "treated", ]
getInitial( vel ~ SSmicmen( conc, Vm, K ), PurTrt )
```

getResponse

Extract Response Variable from an Object

getResponse

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `data.frame`, `gls`, `lme`, and `lmList`.

```
getResponse(object, form, data)
```

ARGUMENTS

`object`: any object

`form`: an optional two-sided formula. Defaults to `formula(object)`.

`data`: a data frame in which to interpret the variables named in `form`. Optional for most methods.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`getResponseFormula`

EXAMPLE

```
## see the method function documentation
```

getResponse.data.frame	Response from Data Frame	getResponse.data.frame
-------------------------------	--------------------------	-------------------------------

The left hand side of `form` is evaluated in `object`.

```
getResponse(object, form)
```

ARGUMENTS

`object`: an object inheriting from class `data.frame`.

`form`: an optional formula specifying the response to be evaluated in `object`. Defaults to `formula(object)`.

VALUE

the value of the left hand side of `form` evaluated in `object`.

SEE ALSO

`getResponseFormula`

EXAMPLE

```
getResponse(Orthodont)
```

getResponse.gls	Extract gls Object Response	getResponse.gls
------------------------	-----------------------------	------------------------

This method function extracts the response variable used in fitting the linear model corresponding to `object`.

```
getResponse(object)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

VALUE

a vector with the response variable corresponding to the linear model represented by `object`.

SEE ALSO

`gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
getResponse(fml)
```


getResponse.lmList

Extract lmList Object Response

getResponse.lmList

The response vectors from each of the `lm` components of `object` are extracted and combined into a single vector.

```
getResponse(object)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

VALUE

a vector with the response vectors corresponding to the `lm` components of `object`.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
getResponse(fml)
```

getResponse.lme

Extract lme Object Response

getResponse.lme

This method function extracts the response variable used in fitting the linear mixed-effects model corresponding to `object`.

```
getResponse(object)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

VALUE

a vector with the response variable corresponding to the linear mixed-effects model represented by `object`.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
getResponse(fml)
```

getResponseFormula	Response Formula	getResponseFormula
---------------------------	------------------	---------------------------

The left hand side of `formula(object)` is returned as a one-sided formula.

```
getResponseFormula(object)
```

ARGUMENTS

object: any object from which a formula can be extracted.

VALUE

a one-sided formula with the response variable associated with `formula(object)`.

SEE ALSO

`getResponse`

EXAMPLE

```
getResponseFormula(y ~ x | g)
```

getStrata	Extract Stratification Variable	getStrata
------------------	---------------------------------	------------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `data.frame`.

```
getStrata(object, ...)
```

ARGUMENTS

object: any object

...: some methods for this generic function may require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`getStrataFormula`, `getStrata.data.frame`

EXAMPLE

```
## see the method function documentation
```

getStrata.data.frame

Extract Strata from a Data Frame

getStrata.data.frame

The right hand side of `form`, or of each element of `form`, when given as a list, is evaluated in `data` and must result in a factor, or an integer vector. In the latter case, the unique elements of the stratification variable, when sorted, must form a sequence of consecutive integers.

```
getStrata(object, form)
```

ARGUMENTS

`object`: an object inheriting from class `data.frame`.

`form`: a one-sided formula specifying an stratification variable, or a list of one-sided formulas specifying stratification variables.

VALUE

either an integer vector with values ranging from zero to the number of strata minus one, or, when `form` is given as a list, a list of such integer vectors.

SEE ALSO

```
getStrataFormula
```

EXAMPLE

```
getStrata(Orthodont, ~ Sex)
getStrata(Pixel, list(~ Dog, ~ Side))
```

getStrataFormula

Extract Stratification Formula

getStrataFormula

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, `pdMat` and `reStruct`.

```
getStrataFormula(object, ...)
```

ARGUMENTS

`object`: any object from which a formula representing a stratification variable can be extracted.

`...`: some methods for this generic function may require additional arguments.

VALUE

a one-sided formula with the stratification variable associated with `object`. For most methods, if no stratification variable is present, a `NULL` value is returned.

SEE ALSO

```
corStrat, pdStrat
```

EXAMPLE

```
pd1 <- pdStrat(~ age, data = Orthodont, strata = ~ Sex)
getStrataFormula(pd1)
```

gls

Fit Linear Model Using Generalized Least Squares

gls

This function fits a linear model using generalized least squares. The errors are allowed to be correlated and/or have unequal variances.

```
gls(model, data, correlation, weights, subset, method, na.action,
     control, verbose)
```

ARGUMENTS

model: a two-sided linear formula object describing the model, with the response on the left of a `~` operator and the terms, separated by `+` operators, on the right.

data: an optional data frame containing the variables named in **model**, **correlation**, **weights**, and **subset**. By default the variables are taken from the environment from which **gls** is called.

correlation: an optional **corStruct** object describing the within-group correlation structure. See the documentation of **corClasses** for a description of the available **corStruct** classes. If a grouping variable is to be used, it must be specified in the **form** argument to the **corStruct** constructor. Defaults to **NULL**, corresponding to uncorrelated errors.

weights: an optional **varFunc** object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to **varFixed**, corresponding to fixed variance weights. See the documentation on **varClasses** for a description of the available **varFunc** classes. Defaults to **NULL**, corresponding to homoscedastic errors.

subset: an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

method: a character string. If **"REML"** the model is fit by maximizing the restricted log-likelihood. If **"ML"** the log-likelihood is maximized. Defaults to **"REML"**.

na.action: a function that indicates what should happen when the data contain **NA**s. The default action (**na.fail**) causes **gls** to print an error message and terminate if there are any incomplete observations.

control: a list of control values for the estimation algorithm to replace the default values returned by the function **glsControl**. Defaults to an empty list.

`verbose`: an optional logical value. If `TRUE` information on the evolution of the iterative algorithm is printed. Default is `FALSE`.

VALUE

an object of class `gls` representing the linear model fit. Generic functions such as `print`, `plot`, and `summary` have methods to show the results of the fit. See `glsObject` for the components of the fit. The functions `resid`, `coef`, and `fitted` can be used to extract some of its components.

REFERENCES

The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear models is presented in detail in Carrol, R.J. and Ruppert, D. (1988) and Davidian, M. and Giltinan, D.M. (1995).

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Carrol, R.J. and Ruppert, D. (1988) "Transformation and Weighting in Regression", Chapman and Hall.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`glsControl`, `glsObject`, `corClasses`, `varClasses`, `corClasses`, `varClasses`

EXAMPLE

```
# AR(1) errors within each Mare
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          correlation = corAR1(form = ~ 1 | Mare))
# variance increases as a power of the absolute fitted values
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          weights = varPower())
```

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the control argument to the `gls` function.

```
glsControl(maxIter, msMaxIter, tolerance, msTol, msScale,
           msVerbose, singular.ok, qrTol, returnObject,
           apVar, .relStep, natural, natUnconstrained, sigma)
```

ARGUMENTS

`maxIter`: maximum number of iterations for the `gls` optimization algorithm. Default is 50.

`msMaxIter`: maximum number of iterations for the `ms` optimization step inside the `gls` optimization. Default is 50.

`tolerance`: tolerance for the convergence criterion in the `gls` algorithm. Default is $1e-6$.

`msTol`: tolerance for the convergence criterion in `ms`, passed as the `rel.tolerance` argument to the function (see documentation on `ms`). Default is $1e-7$.

`msScale`: scale function passed as the `scale` argument to the `ms` function (see documentation on that function). Default is `lmeScale`.

`msVerbose`: a logical value passed as the `trace` argument to `ms` (see documentation on that function). Default is `FALSE`.

`singular.ok`: a logical value indicating whether non-estimable coefficients (resulting from linear dependencies among the columns of the regression matrix) should be allowed. Default is `FALSE`.

`qrTol`: a tolerance for detecting linear dependencies among the columns of the regression matrix in its QR decomposition. Default is `.Machine$single.eps`.

`returnObject`: a logical value indicating whether the fitted object should be returned when the maximum number of iterations is reached without convergence of the algorithm. Default is `FALSE`.

`apVar`: a logical value indicating whether the approximate covariance matrix of the variance-covariance parameters should be calculated. Default is `TRUE`.

`.relStep`: relative step for numerical derivatives calculations. Default is `.Machine$double.eps1/3`.

`natural`: a logical value, or a named list of logical values, indicating whether a natural parameterization should be used for the model structures, when the approximate covariance matrix of the estimators is calculated. If given as a single logical value, it is used for all model structures (`corStruct` and `varFunc` objects) used

in the fit. If given as a list, it must have names `corStruct`, and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`natUnconstrained`: a logical value, or a named list of logical values, indicating whether an unconstrained parameterization should be used for the natural parameters of the model structures. If given as a single logical value, it is used for all model structures (`corStruct` and `varFunc` objects) used in the fit. If given as a list, it must have names `corStruct` and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`sigma`: a numeric value indicating the value at which the residual standard error should be kept fixed during the optimization of the objective function. Defaults to `NULL`, in which case the residual standard error is estimated together with the other model parameters. Must be a non-negative numeric value - setting it to zero has the same effect as the default (`NULL`).

VALUE

a list with components for each of the possible arguments.

SEE ALSO

`gls`, `ms`, `lmeScale`

EXAMPLE

```
# decrease the maximum number iterations in the ms call and
# request that information on the evolution of the ms iterations
# be printed
glsControl(msMaxIter = 20, msVerbose = TRUE)
```

glsObject

Fitted gls Object

glsObject

An object returned by the `gls` function, inheriting from class `gls` and representing a generalized least squares fitted linear model. Objects of this class have methods for the generic functions `anova`, `coef`, `fitted`, `formula`, `getGroups`, `getResponse`, `intervals`, `logLik`, `plot`, `predict`, `print`, `residuals`, `summary`, and `update`.

VALUE

The following components must be included in a legitimate `gls` object.

COMPONENTS

`apVar`: an approximate covariance matrix for the variance-covariance coefficients. If `apVar = FALSE` in the list of control values used in the call to `gls`, this component is equal to `NULL`.

`call`: a list containing an image of the `gls` call that produced the object.

`coefficients`: a vector with the estimated linear model coefficients.

`contrasts`: a list with the contrasts used to represent factors in the model formula. This information is important for making predictions from a new data frame in which not all levels of the original factors are observed. If no factors are used in the model, this component will be an empty list.

`dims`: a list with basic dimensions used in the model fit, including the components `N` - the number of observations in the data and `p` - the number of coefficients in the linear model.

`fitted`: a vector with the fitted values..

`glsStruct`: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

`groups`: a vector with the correlation structure grouping factor, if any is present.

`logLik`: the log-likelihood at convergence.

`method`: the estimation method: either "ML" for maximum likelihood, or "REML" for restricted maximum likelihood.

`numIter`: the number of iterations used in the iterative algorithm.

`residuals`: a vector with the residuals.

`sigma`: the estimated residual standard error.

`varBeta`: an approximate covariance matrix of the coefficients estimates.

SEE ALSO

`gls`, `glsStruct`

A generalized least squares structure is a list of model components representing different sets of parameters in the linear model. A `glsStruct` may contain `corStruct` and `varFunc` objects. `NULL` arguments are not included in the `glsStruct` list.

```
glsStruct(corStruct, varStruct)
```

ARGUMENTS

`corStruct`: an optional `corStruct` object, representing a correlation structure. Default is `NULL`.

`varStruct`: an optional `varFunc` object, representing a variance function structure. Default is `NULL`.

VALUE

a list of model variance-covariance components determining the parameters to be estimated for the associated linear model.

SEE ALSO

`gls`, `corClasses`, `varClasses`

EXAMPLE

```
gls1 <- glsStruct(corAR1(), varPower())
```

This function fits a nonlinear model using generalized least squares. The errors are allowed to be correlated and/or have unequal variances.

```
gnls(model, data, params, start, correlation, weights, subset,
      na.action, naPattern, control, verbose)
```

ARGUMENTS

`model`: a two-sided formula object describing the model, with the response on the left of a `~` operator and a nonlinear expression involving parameters and covariates on the right. If `data` is given, all names used in the formula should be defined as parameters or variables in the data frame.

`data`: an optional data frame containing the variables named in `model`, `correlation`, `weights`, `subset`, and `naPattern`. By default the variables are taken from the environment from which `gnls` is called.

- params:** an optional two-sided linear formula of the form $p_1 + \dots + p_n \sim x_1 + \dots + x_m$, or list of two-sided formulas of the form $p_1 \sim x_1 + \dots + x_m$, with possibly different models for each parameter. The p_1, \dots, p_n represent parameters included on the right hand side of `model` and $x_1 + \dots + x_m$ define a linear model for the parameters (when the left hand side of the formula contains several parameters, they are all assumed to follow the same linear model described by the right hand side expression). A 1 on the right hand side of the formula(s) indicates a single fixed effects for the corresponding parameter(s). By default, the parameters are obtained from the names of `start`.
- start:** an optional named list, or numeric vector, with the initial values for the parameters in `model`. It can be omitted when a `selfStarting` function is used in `model`, in which case the starting estimates will be obtained from a single call to the `nls` function.
- correlation:** an optional `corStruct` object describing the within-group correlation structure. See the documentation of `corClasses` for a description of the available `corStruct` classes. If a grouping variable is to be used, it must be specified in the `form` argument to the `corStruct` constructor. Defaults to `NULL`, corresponding to uncorrelated errors.
- weights:** an optional `varFunc` object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to `varFixed`, corresponding to fixed variance weights. See the documentation on `varClasses` for a description of the available `varFunc` classes. Defaults to `NULL`, corresponding to homoscedastic errors.
- subset:** an optional expression indicating the subset of the rows of `data` that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
- na.action:** a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes `gnls` to print an error message and terminate if there are any incomplete observations.
- naPattern:** an expression or formula object, specifying which returned values are to be regarded as missing.
- control:** a list of control values for the estimation algorithm to replace the default values returned by the function `gnlsControl`. Defaults to an empty list.
- verbose:** an optional logical value. If `TRUE` information on the evolution of the iterative algorithm is printed. Default is `FALSE`.

VALUE

an object of class `gnls`, also inheriting from class `gls`, representing the non-linear model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `gnlsObject` for the components of

the fit. The functions `resid`, `coef`, and `fitted` can be used to extract some of its components.

REFERENCES

- The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear models is presented in detail in Carroll, R.J. and Ruppert, D. (1988) and Davidian, M. and Giltinan, D.M. (1995).
- Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.
- Carroll, R.J. and Ruppert, D. (1988) "Transformation and Weighting in Regression", Chapman and Hall.
- Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.
- Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.
- Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`gnlsControl`, `gnlsObject`, `corClasses`, `varClasses`, `corClasses`, `varClasses`

EXAMPLE

```
# variance increases with a power of the absolute fitted values
fm1 <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
# errors follow an auto-regressive process of order 1
fm2 <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            correlation = corAR1())
```

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the control argument to the `gnls` function.

```
gnlsControl(maxIter,nlsMaxIter,msMaxIter,minScale,tolerance,
            nlsTol,msTol,msScale,returnObject,msVerbose,
            apVar,.relStep,natural,natUnconstrained,sigma)
```

ARGUMENTS

`maxIter`: maximum number of iterations for the `gnls` optimization algorithm. Default is 50.

`nlsMaxIter`: maximum number of iterations for the `nls` optimization step inside the `gnls` optimization. Default is 7.

`msMaxIter`: maximum number of iterations for the `ms` optimization step inside the `gnls` optimization. Default is 50.

`minScale`: minimum factor by which to shrink the default step size in an attempt to decrease the sum of squares in the `nls` step. Default 0.001.

`tolerance`: tolerance for the convergence criterion in the `gnls` algorithm. Default is $1e-6$.

`nlsTol`: tolerance for the convergence criterion in `nls` step. Default is $1e-3$.

`msTol`: tolerance for the convergence criterion in `ms`, passed as the `rel.tolerance` argument to the function (see documentation on `ms`). Default is $1e-7$.

`msScale`: scale function passed as the `scale` argument to the `ms` function (see documentation on that function). Default is `lmeScale`.

`returnObject`: a logical value indicating whether the fitted object should be returned when the maximum number of iterations is reached without convergence of the algorithm. Default is `FALSE`.

`msVerbose`: a logical value passed as the `trace` argument to `ms` (see documentation on that function). Default is `FALSE`.

`apVar`: a logical value indicating whether the approximate covariance matrix of the variance-covariance parameters should be calculated. Default is `TRUE`.

`.relStep`: relative step for numerical derivatives calculations. Default is $.Machine$double.eps^{1/3}$.

`natural`: a logical value, or a named list of logical values, indicating whether a natural parameterization should be used for the model structures, when the approximate covariance matrix of the estimators is calculated. If given as a single logical value, it is used for all model structures (`corStruct` and `varFunc` objects) used in the fit. If given as a list, it must have names `corStruct`, and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`natUnconstrained`: a logical value, or a named list of logical values, indicating whether an unconstrained parameterization should be used for the natural parameters of the model structures. If given as a single logical value, it is used for all model structures (`corStruct` and `varFunc` objects) used in the fit. If given as a list, it must have names `corStruct` and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`sigma`: a numeric value indicating the value at which the residual standard error should be kept fixed during the optimization of the objective function. Defaults to `NULL`, in which case the residual standard error is estimated together with the other model parameters. Must be a non-negative numeric value - setting it to zero has the same effect as the default (`NULL`).

VALUE

a list with components for each of the possible arguments.

SEE ALSO

`gnls`, `ms`, `lmeScale`

EXAMPLE

```
# decrease the maximum number iterations in the ms call and
# request that information on the evolution of the ms iterations
# be printed
gnlsControl(msMaxIter = 20, msVerbose = TRUE)
```

gnlsObject	Fitted gnls Object	gnlsObject
-------------------	---------------------------	-------------------

An object returned by the `gnls` function, inheriting from class `gnls` and also from class `gls`, and representing a generalized nonlinear least squares fitted model. Objects of this class have methods for the generic functions `anova`, `coef`, `fitted`, `formula`, `getGroups`, `getResponse`, `intervals`, `logLik`, `plot`, `predict`, `print`, `residuals`, `summary`, and `update`.

VALUE

The following components must be included in a legitimate `gnls` object.

COMPONENTS

`apVar`: an approximate covariance matrix for the variance-covariance coefficients. If `apVar = FALSE` in the list of control values used in the call to `gnls`, this component is equal to `NULL`.

`call`: a list containing an image of the `gnls` call that produced the object.

`coefficients`: a vector with the estimated nonlinear model coefficients.

`contrasts`: a list with the contrasts used to represent factors in the model formula. This information is important for making predictions from a new data frame in which not all levels of the original factors are observed. If no factors are used in the model, this component will be an empty list.

`dims`: a list with basic dimensions used in the model fit, including the components `N` - the number of observations used in the fit and `p` - the number of coefficients in the nonlinear model.

`fitted`: a vector with the fitted values.

`modelStruct`: an object inheriting from class `gnlsStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

`groups`: a vector with the correlation structure grouping factor, if any is present.

`logLik`: the log-likelihood at convergence.

`numIter`: the number of iterations used in the iterative algorithm.

`residuals`: a vector with the residuals.

`sigma`: the estimated residual standard error.

`varBeta`: an approximate covariance matrix of the coefficients estimates.

SEE ALSO

`gnls`, `gnlsStruct`

gnlsStruct

Generalized Nonlinear Least Squares Structure

gnlsStruct

A generalized nonlinear least squares structure is a list of model components representing different sets of parameters in the nonlinear model. A `gnlsStruct` may contain `corStruct` and `varFunc` objects. `NULL` arguments are not included in the `gnlsStruct` list.

```
gnlsStruct(corStruct, varStruct)
```

ARGUMENTS

`corStruct`: an optional `corStruct` object, representing a correlation structure. Default is `NULL`.

`varStruct`: an optional `varFunc` object, representing a variance function structure. Default is `NULL`.

VALUE

a list of model variance-covariance components determining the parameters to be estimated for the associated nonlinear model.

SEE ALSO

`gnls`, `corClasses`, `varClasses`

EXAMPLE

```
gnls1 <- gnlsStruct(corAR1(), varPower())
```

groupedData

Construct a groupedData Object

groupedData

An object of the `groupedData` class is constructed from the `formula` and `data` by attaching the `formula` as an attribute of the data, along with any of `outer`, `inner`, `labels`, and `units` that are given. If `order.groups` is `TRUE` the grouping factor is converted to an ordered factor with the ordering determined by `FUN`. Depending on the number of grouping levels and the type of primary covariate, the returned object will be of one of three classes: `nfnGroupedData` - numeric covariate, single level of nesting; `nffGroupedData` - factor covariate, single level of nesting; and `nmGroupedData` - multiple levels of nesting. Several modeling and plotting functions can use the `formula` stored with a `groupedData` object to construct default plots and models.

```
groupedData(formula, data, order.groups, FUN, outer, inner,
  labels, units)
```

ARGUMENTS

- formula:** a formula of the form `resp ~ cov | group` where `resp` is the response, `cov` is the primary covariate, and `group` is the grouping factor. The expression `1` can be used for the primary covariate when there is no other suitable candidate. Multiple nested grouping factors can be listed separated by the `/` symbol as in `fact1/fact2`. In an expression like this the `fact2` factor is nested within the `fact1` factor.
- data:** a data frame in which the expressions in `formula` can be evaluated. The resulting `groupedData` object will consist of the same data values in the same order but with additional attributes.
- order.groups:** an optional logical value, or list of logical values, indicating if the grouping factors should be converted to ordered factors according to the function `FUN` applied to the response from each group. If multiple levels of grouping are present, this argument can be either a single logical value (which will be repeated for all grouping levels) or a list of logical values. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). Ordering within a level of grouping is done within the levels of the grouping factors which are outer to it. Changing the grouping factor to an ordered factor does not affect the ordering of the rows in the data frame but it does affect the order of the panels in a trellis display of the data or models fitted to the data. Defaults to `TRUE`.
- FUN:** an optional summary function that will be applied to the values of the response for each level of the grouping factor, when `order.groups = TRUE`, to determine the ordering. Defaults to the `max` function.
- outer:** an optional one-sided formula, or list of one-sided formulas, indicating covariates that are outer to the grouping factor(s). If multiple levels of grouping are present, this argument can be either a single one-sided formula, or a list of one-sided formulas. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. When plotting a `groupedData` object, the argument `outer = TRUE` causes the panels to be determined by the `outer` formula. The points within the panels are associated by level of the grouping factor. Defaults to `NULL`, meaning that no outer covariates are present.
- inner:** an optional one-sided formula, or list of one-sided formulas, indicating covariates that are inner to the grouping factor(s). If multiple levels of grouping are present, this argument can be either a single one-sided formula, or a list of one-sided formulas. If no names are assigned to the list elements, they are assumed in the same order as the group levels (outermost to innermost grouping). An inner covariate can change within the sets of rows defined by the grouping factor. An inner formula can be used to associate points in a plot of a `groupedData` object. Defaults to `NULL`, meaning that no inner covariates are present.

labels: an optional list of character strings giving labels for the response and the primary covariate. The label for the primary covariate is named *x* and that for the response is named *y*. Either label can be omitted.

units: an optional list of character strings giving the units for the response and the primary covariate. The units string for the primary covariate is named *x* and that for the response is named *y*. Either units string can be omitted.

VALUE

an object inheriting from one of the classes `nfnGroupedData`, `nffGroupedData`, or `nmGroupedData`, and also inheriting from classes `groupedData` and `data.frame`.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1997), "Software Design for Longitudinal Data", in "Modeling Longitudinal and Spatially Correlated Data: Methods, Applications and Future Directions", T.G. Gregoire (ed.), Springer-Verlag, New York.
Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at <http://nlme.stat.wisc.edu>.

SEE ALSO

`formula`, `gapply`, `gsummary`, `lme`

EXAMPLE

```
Orth.new <- # create a new copy of the groupedData object
  groupedData(distance ~ age | Subject,
    data = as.data.frame(Orthodont),
    FUN = mean, outer = ~ Sex,
    labels = list( x = "Age",
      y="Distance from pituitary to pterygomaxillary fissure"),
    units = list(x = "(yr)", y = "(mm)"))
plot( Orth.new )      # trellis plot by Subject
formula( Orth.new )   # extractor for the formula
gsummary( Orth.new )  # apply summary by Subject
fml <- lme( Orth.new ) # fixed and groups formulae extracted
                      # from object
```

Provide a summary of the variables in a data frame by groups of rows. This is most useful with a `groupedData` object to examine the variables by group.

```
gsummary(object, FUN, omitGroupingFactor, form, level,  
         groups, invariantsOnly, ...)
```

ARGUMENTS

object: an object to be summarized - usually a `groupedData` object or a `data.frame`.

FUN: an optional summary function or a list of summary functions to be applied to each variable in the frame. The function or functions are applied only to variables in `object` that vary within the groups defined by `groups`. Invariant variables are always summarized by group using the unique value that they assume within that group. If `FUN` is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If `FUN` is a list of functions, the names in the list should designate classes of variables in the frame such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any non-invariant variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

omitGroupingFactor: an optional logical value. When `TRUE` the grouping factor itself will be omitted from the group-wise summary but the levels of the grouping factor will continue to be used as the row names for the data frame that is produced by the summary. Defaults to `FALSE`.

form: an optional one-sided formula that defines the groups. When this formula is given, the right-hand side is evaluated in `object`, converted to a factor if necessary, and the unique levels are used to define the groups. Defaults to `formula(object)`.

level: an optional positive integer giving the level of grouping to be used in an object with multiple nested grouping levels. Defaults to the highest or innermost level of grouping.

groups: an optional factor that will be used to split the rows into groups. Defaults to `getGroups(object, form, level)`.

invariantsOnly: an optional logical value. When `TRUE` only those covariates that are invariant within each group will be summarized. The summary value for the group is always the unique value taken on by that covariate within the group. The columns in the summary are of the same class as the corresponding columns in `object`. By definition, the grouping factor itself must be an invariant. When

combined with `omitGroupingFactor = TRUE`, this option can be used to discover if there are invariant covariates in the data frame. Defaults to `FALSE`.

...: optional additional arguments to the summary functions that are invoked on the variables by group. Often it is helpful to specify `na.rm = TRUE`.

VALUE

A `data.frame` with one row for each level of the grouping factor. The number of columns is at most the number of columns in `object`.

SEE ALSO

`summary`, `groupedData`, `getGroups`

EXAMPLE

```
gsummary( Orthodont ) # default summary by Subject
## gsummary with invariantsOnly = TRUE and
## omitGroupingFactor = TRUE determines whether there
## are covariates like Sex that are invariant
## within the repeated observations on the same Subject.
gsummary( Orthodont, inv = TRUE, omit = TRUE )
```

initialize	Initialize Object	initialize
-------------------	-------------------	-------------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, `lmeStruct`, `reStruct`, and `varFunc`.

```
initialize(object, data, ...)
```

ARGUMENTS

`object`: any object requiring initialization, e.g. "plug-in" structures such as `corStruct` and `varFunc` objects.

`data`: a data frame to be used in the initialization procedure.

...: some methods for this generic function require additional arguments.

VALUE

an initialized object with the same class as `object`. Changes introduced by the initialization procedure will depend on the method function used; see the appropriate documentation.

EXAMPLE

```
## see the method function documentation
```

initialize.corStruct

Initialize corStruct Object

initialize.corStruct

This method initializes `object` by evaluating its associated covariate(s) and grouping factor, if any is present, in `data`, calculating various dimensions and constants used by optimization algorithms involving `corStruct` objects (see the appropriate `Dim` method documentation), and assigning initial values for the coefficients in `object`, if none were present.

```
initialize(object, data, ...)
```

ARGUMENTS

`object`: an object inheriting from class `corStruct` representing a correlation structure.

`data`: a data frame in which to evaluate the variables defined in `formula(object)`.

`...`: this argument is included to make this method compatible with the generic.

VALUE

an initialized object with the same class as `object` representing a correlation structure.

SEE ALSO

`Dim.corStruct`

EXAMPLE

```
csl <- corAR1(form = ~ 1 | Subject)
csl <- initialize(csl, data = Orthodont)
```

initialize.glsStruct

Initialize a glsStruct Object

initialize.glsStruct

The individual linear model components of the `glsStruct` list are initialized.

```
initialize(object, data, control)
```

ARGUMENTS

`object`: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

`data`: a data frame in which to evaluate the variables defined in `formula(object)`.

`control`: an optional list with control parameters for the initialization and optimization algorithms used in `gls`. Defaults to `list(singular.ok = FALSE, qrTol = .Machine$single.eps)`, implying that linear dependencies are not allowed in the model and that the tolerance for detecting linear dependencies among the columns of the regression matrix is `.Machine$single.eps`.

VALUE

a `glsStruct` object similar to `object`, but with initialized model components.

SEE ALSO

`gls`, `initialize.corStruct`, `initialize.varFunc`

initialize.lmeStruct	Initialize an <code>lmeStruct</code> Object	initialize.lmeStruct
-----------------------------	---	-----------------------------

The individual linear mixed-effects model components of the `lmeStruct` list are initialized.

```
initialize(object, data, groups, conLin, control)
```

ARGUMENTS

`object`: an object inheriting from class `lmeStruct`, representing a list of linear mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

`data`: a data frame in which to evaluate the variables defined in `formula(object)`.

`groups`: a data frame with the grouping factors corresponding to the `lme` model associated with `object` as columns, sorted from innermost to outermost grouping level.

`conLin`: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (x) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying `lme` model. Defaults to `attr(object, "conLin")`.

`control`: an optional list with control parameters for the initialization and optimization algorithms used in `lme`. Defaults to `list(niterEM=20, gradHess=TRUE)`, implying that 20 EM iterations are to be used in the derivation of initial estimates for the coefficients of the `reStruct` component of `object` and, if possible, numerical gradient vectors and Hessian matrices for the log-likelihood function are to be used in the optimization algorithm.

VALUE

an `lmeStruct` object similar to `object`, but with initialized model components.

SEE ALSO

`lme`, `initialize.reStruct`, `initialize.corStruct`, `initialize.varFunc`

Initial estimates for the parameters in the `pdMat` objects forming `object`, which have not yet been initialized, are obtained using the methodology described in Bates and Pinheiro (1998). These estimates may be refined using a series of EM iterations, as described in Bates and Pinheiro (1998). The number of EM iterations to be used is defined in `control`.

```
initialize(object, data, conLin, control)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

`data`: a data frame in which to evaluate the variables defined in `formula(object)`.

`conLin`: a condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying model.

`control`: an optional list with a single component `niterEM` controlling the number of iterations for the EM algorithm used to refine initial parameter estimates. It is given as a list for compatibility with other `initialize` methods. Defaults to `list(niterEM = 20)`.

VALUE

an `reStruct` object similar to `object`, but with all `pdMat` components initialized.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`reStruct`, `pdMat`

initialize.varFunc

Initialize varFunc Object

initialize.varFunc

This method initializes `object` by evaluating its associated covariate(s) and grouping factor, if any is present, in `data`; determining if the covariate(s) need to be updated when the values of the coefficients associated with `object` change; initializing the log-likelihood and the weights associated with `object`; and assigning initial values for the coefficients in `object`, if none were present. The covariate(s) will only be initialized if no update is needed when `coef(object)` changes.

```
initialize(object, data, ...)
```

ARGUMENTS

`object`: an object inheriting from class `varFunc`, representing a variance function structure.

`data`: a data frame in which to evaluate the variables named in `formula(object)`.

`...`: this argument is included to make this method compatible with the generic.

VALUE

an initialized object with the same class as `object` representing a variance function structure.

SEE ALSO

EXAMPLE

```
vfl <- varPower( form = ~ age | Sex )
vfl <- initialize( vfl, Orthodont )
```

intervals

Confidence Intervals on Coefficients

intervals

Confidence intervals on the parameters associated with the model represented by `object` are obtained. This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `gls`, `lme`, and `lmList`.

```
intervals(object, level, ...)
```

ARGUMENTS

`object`: a fitted model object from which parameter estimates can be extracted.

`level`: an optional numeric value for the interval confidence level. Defaults to 0.95.

`...`: some methods for the generic may require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`intervals.gls`, `intervals.lme`, `intervals.lmList`

EXAMPLE

```
## see the method documentation
```

intervals.gls

Confidence Intervals on gls Parameters

intervals.gls

Approximate confidence intervals for the parameters in the linear model represented by `object` are obtained, using a normal approximation to the distribution of the (restricted) maximum likelihood estimators (the estimators are assumed to have a normal distribution centered at the true parameter values and with covariance matrix equal to the negative inverse Hessian matrix of the (restricted) log-likelihood evaluated at the estimated parameters). Confidence intervals are obtained in an unconstrained scale first, using the normal approximation, and, if necessary, transformed to the constrained scale, unless the control parameter `natUnconstrained` is set to `FALSE` (see the documentation on `glsControl`).

```
intervals(object, level, which)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

`level`: an optional numeric value with the confidence level for the intervals. Defaults to 0.95.

`which`: an optional character string specifying the subset of parameters for which to construct the confidence intervals. Possible values are "all" for all parameters, "var-cov" for the variance-covariance parameters only, and "coef" for the linear model coefficients only. Defaults to "all".

VALUE

a list with components given by data frames with rows corresponding to parameters and columns `lower`, `est.`, and `upper` representing respectively lower confidence limits, the estimated values, and upper confidence limits for the parameters. Possible components are:

ARGUMENTS

`coef`: linear model coefficients, only present when `which` is not equal to "var-cov".

`corStruct`: correlation parameters, only present when `which` is not equal to "coef" and a correlation structure is used in `object`.

`varFunc`: variance function parameters, only present when `which` is not equal to "coef" and a variance function structure is used in `object`.

`sigma`: residual standard error.

SEE ALSO

`gls`, `print.intervals.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
intervals(fml)
```

intervals.lmList

Confidence Intervals on lmList Coefficients

intervals.lmList

Confidence intervals on the linear model coefficients are obtained for each `lm` component of `object` and organized into a three dimensional array. The first dimension corresponding to the names of the `object` components. The second dimension is given by `lower`, `est.`, and `upper` corresponding, respectively, to the lower confidence limit, estimated coefficient, and upper confidence limit. The third dimension is given by the coefficients names.

```
intervals(object, level, pool)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

`level`: an optional numeric value with the confidence level for the intervals. Defaults to 0.95.

`pool`: an optional logical value indicating whether a pooled estimate of the residual standard error should be used. Default is `attr(object, "pool")`.

VALUE

a three dimensional array with the confidence intervals and estimates for the coefficients of each `lm` component of `object`.

SEE ALSO

`lmList`, `plot.intervals.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
intervals(fml)
```

Approximate confidence intervals for the parameters in the linear mixed-effects model represented by `object` are obtained, using a normal approximation to the distribution of the (restricted) maximum likelihood estimators (the estimators are assumed to have a normal distribution centered at the true parameter values and with covariance matrix equal to the negative inverse Hessian matrix of the (restricted) log-likelihood evaluated at the estimated parameters). Confidence intervals are obtained in an unconstrained scale first, using the normal approximation, and, if necessary, transformed to the constrained scale, unless the control parameter `natUnconstrained` is set to `FALSE` (see the documentation on `lmeControl`).

```
intervals(object, level, which)
```

ARGUMENTS

`object`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

`level`: an optional numeric value with the confidence level for the intervals. Defaults to 0.95.

`which`: an optional character string specifying the subset of parameters for which to construct the confidence intervals. Possible values are "all" for all parameters, "var-cov" for the variance-covariance parameters only, and "fixed" for the fixed effects only. Defaults to "all".

VALUE

a list with components given by data frames with rows corresponding to parameters and columns `lower`, `est.`, and `upper` representing respectively lower confidence limits, the estimated values, and upper confidence limits for the parameters. Possible components are:

ARGUMENTS

`fixed`: fixed effects, only present when `which` is not equal to "var-cov".

`reStruct`: random effects variance-covariance parameters, only present when `which` is not equal to "fixed".

`corStruct`: within-group correlation parameters, only present when `which` is not equal to "fixed" and a correlation structure is used in `object`.

`varFunc`: within-group variance function parameters, only present when `which` is not equal to "fixed" and a variance function structure is used in `object`.

`sigma`: within-group standard deviation.

SEE ALSO

```
lme, print.intervals.lme,
```

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
intervals(fml)
```

isBalanced	Check a Design for Balance	isBalanced
-------------------	----------------------------	-------------------

Check the design of the experiment or study for balance.

```
isBalanced(object, countOnly, level)
```

ARGUMENTS

object: A `groupedData` object containing a data frame and a formula that describes the roles of variables in the data frame. The object will have one or more nested grouping factors and a primary covariate.

countOnly: A logical value indicating if the check for balance should only consider the number of observations at each level of the grouping factor(s). Defaults to `FALSE`.

level: an optional positive integer giving the level of grouping to be used with multi-level data. Defaults to the highest or innermost level of grouping.

VALUE

`TRUE` or `FALSE` according to whether the data are balanced or not

SEE ALSO

`table`, `groupedData`

EXAMPLE

```
isBalanced(Orthodont)           # should return TRUE
isBalanced(Orthodont, countOnly = TRUE) # should return TRUE
isBalanced(Pixel)               # should return FALSE
isBalanced(Pixel, level = 1)    # should return FALSE
```

isInitialized

Check if Object is Initialized

isInitialized

Checks if `object` has been initialized (generally through a call to `initialize`), by searching for components and attributes which are modified during initialization.

```
isInitialized(object)
```

ARGUMENTS

`object`: any object requiring initialization.

VALUE

a logical value indicating whether `object` has been initialized.

SEE ALSO

`initialize`

EXAMPLE

```
pd1 <- pdDiag(~ age)
isInitialized(pd1)
```

isInitialized.reStruct

Check reStruct Initialization

isInitialized.reStruct

Checks if all `pdMat` components of `object` have been initialized.

```
isInitialized(object)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a logical value indicating whether all components of `object` have been initialized.

SEE ALSO

`initialize`, `reStruct`

EXAMPLE

```
rs1 <- reStruct(~ age|Subject)
isInitialized(rs1)
```

isInitialized<-

Set Initialization Status

isInitialized<-

Sets the initialization status of `object` to `value`. This is a generic function and specific methods must be implemented for different classes. In general, methods for this function can only change the initialization status to `FALSE` and trying to use them with `value = TRUE` will cause an error.

```
isInitialized(object) <- value
```

ARGUMENTS

`object`: any object requiring initialization.

`value`: a logical value indicating the initialization status to be used for `object`.

VALUE

An object of the same class as `original`, but with some of its components modified to indicate the initialization status specified in `value`.

SEE ALSO

`isInitialized`, `initialize`

EXAMPLE

```
cs1 <- corSymm(form = ~ 1|Subject)
cs1 <- initialize(cs1, data = Orthodont)
cs1
isInitialized(cs1) <- FALSE
cs1
```

lmList

List of lm Objects with a Common Model

lmList

Data is partitioned according to the levels of the grouping factor `g` and individual `lm` fits are obtained for each data partition, using the model defined in `object`.

```
lmList(object, data, level, na.action, pool)
```

ARGUMENTS

`object`: either a linear formula object of the form $y \sim x_1 + \dots + x_n \mid g$ or a grouped-Data object. In the formula object, y represents the response, x_1, \dots, x_n the covariates, and g the grouping factor specifying the partitioning of the data according to which different `lm` fits should be performed. The grouping factor g may be omitted from the formula, in which case the grouping structure will be obtained from `data`, which must inherit from class `groupedData`. The method function `lmList.groupedData` is documented separately.

`data`: an data frame in which to interpret the variables named in `object`.

`level`: an optional integer specifying the level of grouping to be used when multiple nested levels of grouping are present.

`na.action`: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes `lmList` to print an error message and terminate if there are any incomplete observations.

`pool`: an optional logical value that is preserved as an attribute of the returned value. This will be used as the default for `pool` in calculations of standard deviations or standard errors for summaries.

VALUE

a list of `lm` objects with as many components as the number of groups defined by the grouping factor. Generic functions such as `coef`, `fixef`, `lme`, `pairs`, `plot`, `predict`, `ranef`, `summary`, and `update` have methods that can be applied to an `lmList` object.

SEE ALSO

`lm`, `lme.lmList`.

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
```

lmList.groupedData	lmList Fit from a groupedData Object	lmList.groupedData
---------------------------	--------------------------------------	---------------------------

The response variable and primary covariate in `formula(object)` are used to construct the linear model formula. This formula and the `groupedData` object are passed as the `object` and `data` arguments to `lmList.formula`, together with any other additional arguments in the function call. See the documentation on `lmList.formula` for a description of that function.

```
lmList(object, data, level, na.action, pool)
```

ARGUMENTS

`object`: a data frame inheriting from class `groupedData`.

`data`: this argument is included for consistency with the generic function. It is ignored in this method function.

`other arguments`: identical to the arguments in the generic function call. See the documentation for `lmList`.

VALUE

a list of `lm` objects with as many components as the number of groups defined by the grouping factor. Generic functions such as `coef`, `fixef`, `lme`, `pairs`, `plot`, `predict`, `ranef`, `summary`, and `update` have methods that can be applied to an `lmList` object.

SEE ALSO

`groupedData`, `lm`, `lme.lmList`, `lmList.formula`

EXAMPLE

```
fml <- lmList(Orthodont)
```

lme	Linear Mixed-Effects Models	lme
------------	------------------------------------	------------

This generic function fits a linear mixed-effects model in the formulation described in Laird and Ware (1982) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances.

```
lme(fixed, data, random, correlation, weights, subset, method,  
    na.action, control)
```

ARGUMENTS

- fixed:** a two-sided linear formula object describing the fixed-effects part of the model, with the response on the left of a \sim operator and the terms, separated by + operators, on the right, an `lmList` object, or a `groupedData` object. The method functions `lme.lmList` and `lme.groupedData` are documented separately.
- data:** an optional data frame containing the variables named in `fixed`, `random`, `correlation`, `weights`, and `subset`. By default the variables are taken from the environment from which `lme` is called.
- random:** optionally, any of the following: (i) a one-sided formula of the form $\sim x_1 + \dots + x_n$ | $g_1 / \dots / g_m$, with $x_1 + \dots + x_n$ specifying the model for the random effects and $g_1 / \dots / g_m$ the grouping structure (m may be equal to 1, in which case no / is required). The random effects formula will be repeated for all levels of grouping, in the case of multiple levels of grouping; (ii) a list of one-sided formulas of the form $\sim x_1 + \dots + x_n$ | g , with possibly different random effects models for each grouping level. The order of nesting will be assumed the same as the order of the elements in the list; (iii) a one-sided formula of the form $\sim x_1 + \dots + x_n$, or a `pMat` object with a formula (i.e. a non-NULL value for `formula(object)`), or a list of such formulas or `pMat` objects. In this case, the grouping structure formula will be derived from the data used to fit the linear mixed-effects model, which should inherit from class `groupedData`; (iv) a named list of formulas or `pMat` objects as in (iii), with the grouping factors as names. The order of nesting will be assumed the same as the order of the elements in the list; (v) an `reStruct` object. See the documentation on `pClasses` for a description of the available `pMat` classes. Defaults to a formula consisting of the right hand side of `fixed`.
- correlation:** an optional `corStruct` object describing the within-group correlation structure. See the documentation of `corClasses` for a description of the available `corStruct` classes. Defaults to `NULL`, corresponding to no within-group correlations.

`weights`: an optional `varFunc` object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to `varFixed`, corresponding to fixed variance weights. See the documentation on `varClasses` for a description of the available `varFunc` classes. Defaults to `NULL`, corresponding to homoscedastic within-group errors.

`subset`: an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

`method`: a character string. If `"REML"` the model is fit by maximizing the restricted log-likelihood. If `"ML"` the log-likelihood is maximized. Defaults to `"REML"`.

`na.action`: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes `lme` to print an error message and terminate if there are any incomplete observations.

`control`: a list of control values for the estimation algorithm to replace the default values returned by the function `lmeControl`. Defaults to an empty list.

VALUE

an object of class `lme` representing the linear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `lmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixef`, and `ranef` can be used to extract some of its components.

REFERENCES

The computational methods are described in Bates, D.M. and Pinheiro, J.C. (1998) and follow on the general framework of Lindstrom, M.J. and Bates, D.M. (1988). The model formulation is described in Laird, N.M. and Ware, J.H. (1982). The variance-covariance parametrizations are described in Pinheiro, J.C. and Bates, D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", *Biometrics*, 38, 963-974.

Lindstrom, M.J. and Bates, D.M. (1988) "Newton-Raphson and EM Algorithms

for Linear Mixed-Effects Models for Repeated-Measures Data”, Journal of the American Statistical Association, 83, 1014-1022.

Littel, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) ”SAS Systems for Mixed Models”, SAS Institute.

Pinheiro, J.C. and Bates., D.M. (1996) ”Unconstrained Parametrizations for Variance-Covariance Matrices”, Statistics and Computing, 6, 289-296.

Venables, W.N. and Ripley, B.D. (1997) ”Modern Applied Statistics with S-plus”, 2nd Edition, Springer-Verlag.

SEE ALSO

`lmeControl`, `lme.lmList`, `lme.groupedData`, `lmeObject`, `lmList`, `reStruct`, `reStruct`, `pdClasses`, `corClasses`, `varClasses`

EXAMPLE

```
fml <- lme(distance ~ age, data = Orthodont) # random is ~ age
fm2 <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)
```

`lme.groupedData`

LME fit from groupedData Object

`lme.groupedData`

The response variable and primary covariate in `formula(fixed)` are used to construct the fixed effects model formula. This formula and the `groupedData` object are passed as the `fixed` and `data` arguments to `lme.formula`, together with any other additional arguments in the function call. See the documentation on `lme.formula` for a description of that function.

```
lme(fixed, data, random, correlation, weights, subset, method,
    na.action, control)
```

ARGUMENTS

fixed: a data frame inheriting from class `groupedData`.

data: this argument is included for consistency with the generic function. It is ignored in this method function.

other arguments: identical to the arguments in the generic function call. See the documentation on `lme`.

VALUE

an object of class `lme` representing the linear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `lmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixef`, and `ranef` can be used to extract some of its components.

REFERENCES

The computational methods are described in Bates, D.M. and Pinheiro, J.C. (1998) and follow on the general framework of Lindstrom, M.J. and Bates,

D.M. (1988). The model formulation is described in Laird, N.M. and Ware, J.H. (1982). The variance-covariance parametrizations are described in Pinheiro, J.C. and Bates., D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", *Biometrics*, 38, 963-974.

Lindstrom, M.J. and Bates, D.M. (1988) "Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data", *Journal of the American Statistical Association*, 83, 1014-1022.

Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`lme`, `groupedData`, `lmeObject`

EXAMPLE

```
fml <- lme(Orthodont)
```

If the random effects names defined in `random` are a subset of the `lmList` object coefficient names, initial estimates for the covariance matrix of the random effects are obtained (overwriting any values given in `random`). `formula(fixed)` and the `data` argument in the calling sequence used to obtain `fixed` are passed as the `fixed` and `data` arguments to `lme.formula`, together with any other additional arguments in the function call. See the documentation on `lme.formula` for a description of that function.

```
lme(fixed, data, random, correlation, weights, subset, method,
    na.action, control)
```

ARGUMENTS

`fixed`: an object inheriting from class `lmList`, representing a list of `lm` fits with a common model.

`data`: this argument is included for consistency with the generic function. It is ignored in this method function.

`random`: an optional one-sided linear formula with no conditioning expression, or a `pdMat` object with a `formula` attribute. Multiple levels of grouping are not allowed with this method function. Defaults to a formula consisting of the right hand side of `formula(fixed)`.

`other arguments`: identical to the arguments in the generic function call. See the documentation on `lme`.

VALUE

an object of class `lme` representing the linear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `lmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixef`, and `ranef` can be used to extract some of its components.

REFERENCES

The computational methods are described in Bates, D.M. and Pinheiro, J.C. (1998) and follow on the general framework of Lindstrom, M.J. and Bates, D.M. (1988). The model formulation is described in Laird, N.M. and Ware, J.H. (1982). The variance-covariance parametrizations are described in Pinheiro, J.C. and Bates., D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", Biometrics, 38, 963-974.

Lindstrom, M.J. and Bates, D.M. (1988) "Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data", Journal of the American Statistical Association, 83, 1014-1022.

Littel, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", Statistics and Computing, 6, 289-296.

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`lme`, `lmList`, `lmeObject`

EXAMPLE

```
fml <- lmList(Orthodont)
fm2 <- lme(fml)
```

lmeControl	Control Values for lme Fit	lmeControl
-------------------	----------------------------	-------------------

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the `control` argument to the `lme` function.

```
lmeControl(maxIter, msMaxIter, tolerance, niterEM, msTol,
           msScale, msVerbose, returnObject, gradHess, apVar,
           .relStep, natural, natUnconstrained, sigma)
```

ARGUMENTS

maxIter: maximum number of iterations for the `lme` optimization algorithm. Default is 50.

msMaxIter: maximum number of iterations for the `ms` optimization step inside the `lme` optimization. Default is 50.

tolerance: tolerance for the convergence criterion in the `lme` algorithm. Default is 1e-6.

niterEM: number of iterations for the EM algorithm used to refine the initial estimates of the random effects variance-covariance coefficients. Default is 25.

- `msTol`: tolerance for the convergence criterion in `ms`, passed as the `rel.tolerance` argument to the function (see documentation on `ms`). Default is `1e-7`.
- `msScale`: scale function passed as the `scale` argument to the `ms` function (see documentation on that function). Default is `lmeScale`.
- `msVerbose`: a logical value passed as the `trace` argument to `ms` (see documentation on that function). Default is `FALSE`.
- `returnObject`: a logical value indicating whether the fitted object should be returned when the maximum number of iterations is reached without convergence of the algorithm. Default is `FALSE`.
- `gradHess`: a logical value indicating whether numerical gradient vectors and Hessian matrices of the log-likelihood function should be used in the `ms` optimization. This option is only available when the correlation structure (`corStruct`) and the variance function structure (`varFunc`) have no "varying" parameters and the `pdMat` classes used in the random effects structure are `pdSymm` (general positive-definite), `pdDiag` (diagonal), `pdIdent` (multiple of the identity), or `pdCompSymm` (compound symmetry). Default is `TRUE`.
- `apVar`: a logical value indicating whether the approximate covariance matrix of the variance-covariance parameters should be calculated. Default is `TRUE`.
- `.relStep`: relative step for numerical derivatives calculations. Default is `.Machine$double.eps1/3`.
- `natural`: a logical value, or a named list of logical values, indicating whether a natural parameterization should be used for the model structures, when the approximate covariance matrix of the estimators is calculated. If given as a single logical value, it is used for all model structures (`pdMat`, `corStruct`, and `varFunc` objects) used in the fit. If given as a list, it must have names `reStruct`, `corStruct`, and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.
- `natUnconstrained`: a logical value, or a named list of logical values, indicating whether an unconstrained parameterization should be used for the natural parameters of the model structures. If given as a single logical value, it is used for all model structures (`pdMat`, `corStruct` and `varFunc` objects) used in the fit. If given as a list, it must have names `reStruct`, `corStruct` and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.
- `sigma`: a numeric value indicating the value at which the within-group standard error should be kept fixed during the optimization of the objective function. Defaults to `NULL`, in which case the within-group standard error is estimated together with the other model parameters. Must be a non-negative numeric value - setting it to zero has the same effect as the default (`NULL`).

VALUE

a list with components for each of the possible arguments.

SEE ALSO

`lme`, `ms`, `lmeScale`

EXAMPLE

```
# decrease the maximum number iterations in the ms call and
# request that information on the evolution of the ms iterations be printed
lmeControl(msMaxIter = 20, msVerbose = TRUE)
```

lmeObject	Fitted lme Object	lmeObject
------------------	--------------------------	------------------

An object returned by the `lme` function, inheriting from class `lme` and representing a fitted linear mixed-effects model. Objects of this class have methods for the generic functions `anova`, `coef`, `fitted`, `fixef`, `formula`, `getGroups`, `getResponse`, `intervals`, `logLik`, `pairs`, `plot`, `predict`, `print`, `ranef`, `residuals`, `summary`, and `update`.

VALUE

The following components must be included in a legitimate `lme` object.

COMPONENTS

- `apVar`: an approximate covariance matrix for the variance-covariance coefficients. If `apVar = FALSE` in the list of control values used in the call to `lme`, this component is equal to `NULL`.
- `call`: a list containing an image of the `lme` call that produced the object.
- `coefficients`: a list with two components, `fixed` and `random`, where the first is a vector containing the estimated fixed effects and the second is a list of matrices with the estimated random effects for each level of grouping. For each matrix in the `random` list, the columns refer to the random effects and the rows to the groups.
- `contrasts`: a list with the contrasts used to represent factors in the fixed effects formula and/or random effects formula. This information is important for making predictions from a new data frame in which not all levels of the original factors are observed. If no factors are used in the `lme` model, this component will be an empty list.
- `dims`: a list with basic dimensions used in the `lme` fit, including the components `N` - the number of observations in the data, `Q` - the number of grouping levels, `qvec` - the number of random effects at each level from innermost to outermost (last two values are equal to zero and correspond to the fixed effects and the response), `ngrps` - the number of groups at each level from innermost to outermost (last two values are one and correspond to the fixed effects and the response), and `ncol` - the number of columns in the model matrix for each level of grouping from innermost to outermost (last two values are equal to the number of fixed effects and one).

`fitted`: a data frame with the fitted values as columns. The leftmost column corresponds to the population fixed effects (corresponding to the fixed effects only) and successive columns from left to right correspond to increasing levels of grouping.

`fixDF`: a list with components `x` and `terms` specifying the denominator degrees of freedom for, respectively, t-tests for the individual fixed effects and F-tests for the fixed-effects terms in the models.

`groups`: a data frame with the grouping factors as columns. The grouping level increases from left to right.

`logLik`: the (restricted) log-likelihood at convergence.

`method`: the estimation method: either "ML" for maximum likelihood, or "REML" for restricted maximum likelihood.

`modelStruct`: an object inheriting from class `lmeStruct`, representing a list of mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

`numIter`: the number of iterations used in the iterative algorithm.

`residuals`: a data frame with the residuals as columns. The leftmost column corresponds to the population residuals and successive columns from left to right correspond to increasing levels of grouping.

`sigma`: the estimated within-group error standard deviation.

`varFix`: an approximate covariance matrix of the fixed effects estimates.

SEE ALSO

`lme`, `lmeStruct`

This function calculates the scales to be used for each coefficient estimated through an `ms` optimization in the `lme` function. If all initial values are zero, the scale is set to one for all coefficients; else, the scale for a coefficient with non-zero initial value is equal to the inverse of its initial value and the scale for the coefficients with initial value equal to zero is set to the median of the non-zero initial value coefficients.

```
lmeScale(start)
```

ARGUMENTS

start: the starting values for the coefficients to be estimated.

VALUE

a vector with the scales to be used in `ms` for estimating the coefficients.

SEE ALSO

`ms`

A linear mixed-effects structure is a list of model components representing different sets of parameters in the linear mixed-effects model. An `lmeStruct` list must contain at least a `reStruct` object, but may also contain `corStruct` and `varFunc` objects. `NULL` arguments are not included in the `lmeStruct` list.

```
lmeStruct(reStruct, corStruct, varStruct)
```

ARGUMENTS

reStruct: a `reStruct` representing a random effects structure.

corStruct: an optional `corStruct` object, representing a correlation structure. Default is `NULL`.

varStruct: an optional `varFunc` object, representing a variance function structure. Default is `NULL`.

VALUE

a list of model components determining the parameters to be estimated for the associated linear mixed-effects model.

SEE ALSO

`lme`, `reStruct`, `corClasses`, `varClasses`

EXAMPLE

```
lms1 <- lmeStruct(reStruct(~ age), corAR1(), varPower())
```

logDet

Extract the Logarithm of the Determinant

logDet

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, several `pdMat` classes, and `reStruct`.

```
logDet(object, ...)
```

ARGUMENTS

object: an object from which a matrix, or list of matrices, can be extracted

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`logLik`

EXAMPLE

```
## see the method function documentation
```

logDet.corStruct

Extract corStruct Log-Determinant

logDet.corStruct

This method function extracts the logarithm of the determinant of a square-root factor of the correlation matrix associated with `object`, or the sum of the log-determinants of square-root factors of the list of correlation matrices associated with `object`.

```
logDet(object, covariate)
```

ARGUMENTS

object: an object inheriting from class `corStruct`, representing a correlation structure.

covariate: an optional covariate vector (matrix), or list of covariate vectors (matrices), at which values the correlation matrix, or list of correlation matrices, are to be evaluated. Defaults to `getCovariate(object)`.

VALUE

the log-determinant of a square-root factor of the correlation matrix associated with `object`, or the sum of the log-determinants of square-root factors of the list of correlation matrices associated with `object`.

SEE ALSO

`logLik.corStruct`, `corMatrix.corStruct`

EXAMPLE

```
cs1 <- corAR1(0.3)
logDet(cs1, covariate = 1:4)
```

logDet.pdMat	pdMat Log-Determinant	logDet.pdMat
---------------------	-----------------------	---------------------

This method function extracts the logarithm of the determinant of a square-root factor of the positive-definite matrix represented by `object`.

```
logDet(object)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive definite matrix.

VALUE

the log-determinant of a square-root factor of the positive-definite matrix represented by `object`.

SEE ALSO

`pdMat`

EXAMPLE

```
pd1 <- pdSymm(diag(1:3))
logDet(pd1)
```

logDet.reStruct	Extract reStruct Log-Determinants	logDet.reStruct
------------------------	-----------------------------------	------------------------

Calculates, for each of the `pdMat` components of `object`, the logarithm of the determinant of a square-root factor.

```
logDet(object)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a vector with the log-determinants of square-root factors of the `pdMat` components of `object`.

SEE ALSO

`reStruct`, `pdMat`

EXAMPLE

```
rs1 <- reStruct(list(A = pdSymm(diag(1:3), form = ~ Score),
  B = pdDiag(2 * diag(4), form = ~ Educ)))
logDet(rs1)
```

logLik	Extract Log-Likelihood	logLik
---------------	------------------------	---------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, `gls`, `lm`, `lme`, `lmList`, `lmeStruct`, `reStruct`, and `varFunc`.

```
logLik(object, ...)
```

ARGUMENTS

object: any object from which a log-likelihood value, or a contribution to a log-likelihood value, can be extracted.

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

EXAMPLE

```
## see the method function documentation
```

logLik.corStruct	corStruct Log-Likelihood	logLik.corStruct
-------------------------	--------------------------	-------------------------

This method function extracts the component of a Gaussian log-likelihood associated with the correlation structure, which is equal to the negative of the logarithm of the determinant (or sum of the logarithms of the determinants) of the matrix (or matrices) represented by `object`.

```
logLik(object, data)
```

ARGUMENTS

object: an object inheriting from class `corStruct`, representing a correlation structure.

data: this argument is included to make this method function compatible with other `logLik` methods and will be ignored.

VALUE

the negative of the logarithm of the determinant (or sum of the logarithms of the determinants) of the correlation matrix (or matrices) represented by `object`.

SEE ALSO

```
logDet.corStruct
```

EXAMPLE

```
cs1 <- corAR1(0.2)
cs1 <- initialize(cs1, data = Orthodont)
logLik(cs1)
```

logLik.gls

Log-Likelihood of a gls Object

logLik.gls

If `REML=FALSE`, returns the log-likelihood value of the linear model represented by `object` evaluated at the estimated coefficients; else, the restricted log-likelihood evaluated at the estimated coefficients is returned.

```
logLik(object, REML)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

REML: an optional logical value. If `TRUE` the restricted log-likelihood is returned, else, if `FALSE`, the log-likelihood is returned. Defaults to `FALSE`.

VALUE

the (restricted) log-likelihood of the linear model represented by `object` evaluated at the estimated coefficients.

REFERENCES

Harville, D.A. (1974) "Bayesian Inference for Variance Components Using Only Error Contrasts", *Biometrika*, 61, 383-385.

SEE ALSO

`gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
logLik(fml)
logLik(fml, REML = FALSE)
```

`Pars` is used to update the coefficients of the model components of `object` and the individual (restricted) log-likelihood contributions of each component are added together. The type of log-likelihood (restricted or not) is determined by the `settings` attribute of `object`.

```
logLik(object, Pars, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

`Pars`: the parameter values at which the (restricted) log-likelihood is to be evaluated.

`conLin`: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying linear model. Defaults to `attr(object, "conLin")`.

VALUE

the (restricted) log-likelihood for the linear model described by `object`, evaluated at `Pars`.

SEE ALSO

`gls`, `glsStruct`

Returns the log-likelihood value of the nonlinear model represented by `object` evaluated at the estimated coefficients.

```
logLik(object)
```

ARGUMENTS

`object`: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

VALUE

the log-likelihood of the linear model represented by `object` evaluated at the estimated coefficients.

SEE ALSO

`gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
logLik(fml)
```

logLik.gnlsStruct

Log-Likelihood of a gnlsStruct Object

logLik.gnlsStruct

`Pars` is used to update the coefficients of the model components of `object` and the individual log-likelihood contributions of each component are added together.

```
logLik(object, Pars, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `gnlsStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects, and attributes specifying the underlying nonlinear model and the response variable.

`Pars`: the parameter values at which the log-likelihood is to be evaluated.

`conLin`: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (\mathbf{X}) combined with a response vector (\mathbf{y}), and "logLik", corresponding to the log-likelihood of the underlying nonlinear model. Defaults to `attr(object, "conLin")`.

VALUE

the log-likelihood for the linear model described by `object`, evaluated at `Pars`.

SEE ALSO

`gnls`, `gnlsStruct`

If `REML=FALSE`, returns the log-likelihood value of the linear model represented by `object` evaluated at the estimated coefficients; else, the restricted log-likelihood evaluated at the estimated coefficients is returned.

```
logLik(object, REML)
```

ARGUMENTS

object: an object inheriting from class `lm`.

REML: an optional logical value. If `TRUE` the restricted log-likelihood is returned, else, if `FALSE`, the log-likelihood is returned. Defaults to `FALSE`.

VALUE

the (restricted) log-likelihood of the linear model represented by `object` evaluated at the estimated coefficients.

REFERENCES

Harville, D.A. (1974) "Bayesian Inference for Variance Components Using Only Error Contrasts", *Biometrika*, 61, 383-385.

SEE ALSO

`lm`

EXAMPLE

```
fml <- lm(distance ~ Sex * age, Orthodont)
logLik(fml)
logLik(fml, REML = TRUE)
```

If `pool=FALSE`, the (restricted) log-likelihoods of the `lm` components of `object` are summed together. Else, the (restricted) log-likelihood of the `lm` fit with different coefficients for each level of the grouping factor associated with the partitioning of the `object` components is obtained.

```
logLik(object, REML, pool)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

REML: an optional logical value. If `TRUE` the restricted log-likelihood is returned, else, if `FALSE`, the log-likelihood is returned. Defaults to `FALSE`.

pool: an optional logical value indicating whether all `lm` components of `object` may be assumed to have the same error variance. Default is `attr(object, "pool")`.

VALUE

either the sum of the (restricted) log-likelihoods of each `lm` component in `object`, or the (restricted) log-likelihood for the `lm` fit with separate coefficients for each component of `object`.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
logLik(fml)
```


If `REML=FALSE`, returns the log-likelihood value of the linear mixed-effects model represented by `object` evaluated at the estimated coefficients; else, the restricted log-likelihood evaluated at the estimated coefficients is returned.

```
logLik(object, REML)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

REML: an optional logical value. If `TRUE` the restricted log-likelihood is returned, else, if `FALSE`, the log-likelihood is returned. Defaults to `FALSE`.

VALUE

the (restricted) log-likelihood of the linear mixed-effects model represented by `object` evaluated at the estimated coefficients.

REFERENCES

Harville, D.A. (1974) "Bayesian Inference for Variance Components Using Only Error Contrasts", *Biometrika*, 61, 383-385.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(distance ~ Sex * age, Orthodont, random = ~ age,
           method = "ML")
logLik(fml)
logLik(fml, REML = TRUE)
```

`Pars` is used to update the coefficients of the model components of `object` and the individual (restricted) log-likelihood contributions of each component are added together. The type of log-likelihood (restricted or not) is determined by the `settings` attribute of `object`.

```
logLik(object, Pars, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `lmeStruct`, representing a list of linear mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

`Pars`: the parameter values at which the (restricted) log-likelihood is to be evaluated.

`conLin`: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying lme model. Defaults to `attr(object, "conLin")`.

VALUE

the (restricted) log-likelihood for the linear mixed-effects model described by `object`, evaluated at `Pars`.

SEE ALSO

`lme`, `lmeStruct`

Calculates the log-likelihood, or restricted log-likelihood, of the Gaussian linear mixed-effects model represented by `object` and `conLin` (assuming spherical within-group covariance structure), evaluated at `coef(object)`. The `settings` attribute of `object` determines whether the log-likelihood, or the restricted log-likelihood, is to be calculated. The computational methods are described in Bates and Pinheiro (1998).

```
logLik(object, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

`conLin`: a condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying model.

VALUE

the log-likelihood, or restricted log-likelihood, of linear mixed-effects model represented by `object` and `conLin`, evaluated at `coef(object)`.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`reStruct`, `pdMat`

logLik.varFunc	varFunc Log-Likelihood	logLik.varFunc
-----------------------	-------------------------------	-----------------------

This method function extracts the component of a Gaussian log-likelihood associated with the variance function structure represented by `object`, which is equal to the sum of the logarithms of the corresponding weights.

```
logLik(object, data)
```

ARGUMENTS

`object`: an object inheriting from class `varFunc`, representing a variance function structure.

`data`: this argument is included to make this method function compatible with other `logLik` methods and will be ignored.

VALUE

the sum of the logarithms of the weights corresponding to the variance function structure represented by `object`.

EXAMPLE

```
vf1 <- varPower(form = ~ age)
vf1 <- initialize(vf1, Orthodont)
coef(vf1) <- 0.1
logLik(vf1)
```

matrix<-

Assign Matrix Values

matrix<-

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `pdMat`, `pdBlocked`, and `reStruct`.

```
matrix(object) <- value
```

ARGUMENTS

object: any object to which `as.matrix` can be applied.

value: a matrix, or list of matrices, with the same dimensions as `as.matrix(object)` with the new values to be assigned to the matrix associated with `object`.

VALUE

will depend on the method function; see the appropriate documentation.

SEE ALSO

`as.matrix`

EXAMPLE

```
## see the method function documentation
```

matrix<-pdMat

Assign Matrix to a pdMat Object

matrix<-pdMat

The positive-definite matrix represented by `object` is replaced by `value`. If the original matrix had row and/or column names, the corresponding names for `value` can either be `NULL`, or a permutation of the original names.

```
matrix(object) <- value
```

ARGUMENTS

object: an object inheriting from class `pdMat`, representing a positive definite matrix.

value: a matrix with the new values to be assigned to the positive-definite matrix represented by `object`. Must have the same dimensions as `as.matrix(object)`.

VALUE

a `pdMat` object similar to `object`, but with its coefficients modified to produce the matrix in `value`.

SEE ALSO

`pdMat`

EXAMPLE

```
pd1 <- pdSymm(diag(3))
matrix(pd1) <- diag(1:3)
pd1
```

matrix<-,.pdKron

Assign Matrix to a pdKron Object

matrix<-,.pdKron

If `value` has the same dimensions as the basic pd matrix used in the Kronecker product represented by `object`, it replaces the basic pd matrix. Otherwise, if `value` has the same dimensions as `as.matrix(object)`, the average of the block diagonal elements of `value` of dimensions equal to that of the basic pd matrix replaces the latter. If the original matrix represented by `object` (or the corresponding basic pd matrix) had row and/or column names, the corresponding names for `value` can either be `NULL`, or a permutation of the original names.

```
matrix(object) <- value
```

ARGUMENTS

object: an object inheriting from class `pdKron`, representing a Kronecker-product positive definite matrix.

value: a matrix with the new values to be assigned to the basic positive-definite matrix used in the Kronecker product represented by `object`. It can either have the same dimensions as the basic pd matrix, or the same dimensions as `as.matrix(object)`.

VALUE

a `pdKron` object similar to `object`, but with its coefficients modified to produce the matrix in `value`, or, depending on the dimensions of `value`, the average of the block diagonal elements of `value`.

SEE ALSO

`pdKron`

EXAMPLE

```
pd1 <- pdKron(~ day, data = Pixel, groups = ~ Side)
matrix(pd1) <- 3 * diag(2) + 1
pd1
matrix(pd1) <- 3 * diag(4) + 1
pd1
```

The individual matrices in `value` are assigned to each `pdMat` component of `object`, in the they are listed. The new matrices must have the same dimensions as the matrices they are meant to replace.

```
matrix(object) <- value
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

value: a matrix, or list of matrices, with the new values to be assigned to the matrices associated with the `pdMat` components of `object`.

VALUE

an `reStruct` object similar to `object`, but with the coefficients of the individual `pdMat` components modified to produce the matrices listed in `value`.

SEE ALSO

`reStruct`, `pdMat`

EXAMPLE

```
rs1 <- reStruct(list(Dog = ~ day, Side = ~ 1), data = Pixel)
matrix(rs1) <- list(diag(2), 3)
```

The model matrices for each element of `formula(object)`, calculated using data, are bound together column-wise. When multiple grouping levels are present (i.e. when `length(object)>1`), the individual model matrices are combined from innermost (at the leftmost position) to outermost (at the rightmost position).

```
model.matrix(object, data, contr)
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

data: a data frame in which to evaluate the variables defined in `formula(object)`.

contr: an optional named list specifying the contrasts to be used for representing the factor variables in `data`. The components names should match the names of the variables in `data` for which the contrasts are to be specified. The components of this list will be used as the `contrasts` attribute of the corresponding factor. If missing, the default contrast specification is used.

VALUE

a matrix obtained by binding together, column-wise, the model matrices for each element of `formula(object)`.

SEE ALSO

`model.matrix`, `contrasts`, `reStruct`, `formula.reStruct`

EXAMPLE

```
rs1 <- reStruct(list(Dog = ~ day, Side = ~ 1), data = Pixel)
model.matrix(rs1, Pixel)
```

Names	Names Associated with an Object	Names
-------	---------------------------------	-------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `formula`, `modelStruct`, `pdBlocked`, `pdMat`, and `reStruct`.

```
Names(object, ...)
Names(object, ...) <- value
```

ARGUMENTS

`object`: any object for which names can be extracted and/or assigned.

`...`: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SIDE EFFECTS

On the left side of an assignment, sets the names associated with `object` to `value`, which must have an appropriate length.

NOTE

If names were generic, there would be no need for this generic function.

SEE ALSO

`Names.formula`, `Names.pdMat`

EXAMPLE

```
## see the method function documentation
```

Names.formula

Extract Names from a formula

Names.formula

This method function returns the names of the terms corresponding to the right hand side of `object` (treated as a linear formula), obtained as the column names of the corresponding `model.matrix`.

```
Names(object, data, exclude)
```

ARGUMENTS

`object`: an object inheriting from class `formula`.

`data`: an optional data frame containing the variables specified in `object`. By default the variables are taken from the environment from which `Names.formula` is called.

`exclude`: an optional character vector with names to be excluded from the returned value. Default is `c("pi", ".")`.

VALUE

a character vector with the column names of the `model.matrix` corresponding to the right hand side of `object` which are not listed in `excluded`.

SEE ALSO

`model.matrix`, `terms`, `Names`

EXAMPLE

```
Names(distance ~ Sex * age, data = Orthodont)
```

Names.pdBlocked

Names of a pdBlocked Object

Names.pdBlocked

This method function extracts the first element of the `Dimnames` attribute, which contains the column names, for each block diagonal element in the matrix represented by `object`.

```
Names(object, asList)
```

ARGUMENTS

`object`: an object inheriting from class `pdBlocked` representing a positive-definite matrix with block diagonal structure

`asList`: a logical value. If `TRUE` a list with the names for each block diagonal element is returned. If `FALSE` a character vector with all column names is returned. Defaults to `FALSE`.

VALUE

if `asList` is `FALSE`, a character vector with column names of the matrix represented by `object`; otherwise, if `asList` is `TRUE`, a list with components given by the column names of the individual block diagonal elements in the matrix represented by `object`.

SEE ALSO

`Names`, `Names.pdMat`

EXAMPLE

```
pd1 <- pdBlocked(list(~ Sex - 1, ~ age - 1), data = Orthodont)
Names(pd1)
```

Names.pdMat	Names of a pdMat Object	Names.pdMat
--------------------	-------------------------	--------------------

This method function returns the first element of the `Dimnames` attribute of `object`, which contains the column names of the matrix represented by `object`.

```
Names(object)
Names(object) <- value
```

ARGUMENTS

object: an object inheriting from class `pdMat`, representing a positive-definite matrix.

value: a character vector with the replacement values for the column and row names of the matrix represented by `object`. It must have length equal to the dimension of the matrix represented by `object` and, if names have been previously assigned to `object`, it must correspond to a permutation of the original names.

VALUE

if `object` has a `Dimnames` attribute then the first element of this attribute is returned; otherwise `NULL`.

SIDE EFFECTS

On the left side of an assignment, sets the `Dimnames` attribute of `object` to `list(value, value)`.

SEE ALSO

`Names`, `Names.pdBlocked`

EXAMPLE

```
pd1 <- pdSymm(~ age, data = Orthodont)
Names(pd1)
```

Names.reStruct

Names of an reStruct Object

Names.reStruct

This method function extracts the column names of each of the positive-definite matrices represented the `pdMat` elements of `object`.

```
Names(object)
Names(object) <- value
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

`value`: a list of character vectors with the replacement values for the names of the individual `pdMat` objects that form `object`. It must have the same length as `object`.

VALUE

a list containing the column names of each of the positive-definite matrices represented by the `pdMat` elements of `object`.

SIDE EFFECTS

On the left side of an assignment, sets the `Names` of the `pdMat` elements of `object` to the corresponding element of `value`.

SEE ALSO

`reStruct`, `pdMat`, `Names.pdMat`

EXAMPLE

```
rs1 <- reStruct(list(Dog = ~ day, Side = ~ 1), data = Pixel)
Names(rs1)
```

needUpdate

Check if Update is Needed

needUpdate

This function is generic; method functions can be written to handle specific classes of objects. By default, it tries to extract a `needUpdate` attribute of `object`. If this is `NULL` or `FALSE` it returns `FALSE`; else it returns `TRUE`. Updating of objects usually takes place in iterative algorithms in which auxiliary quantities associated with the object, and not being optimized over, may change.

```
needUpdate(object)
```

ARGUMENTS

`object`: any object

VALUE

a logical value indicating whether `object` needs to be updated.

EXAMPLE

```
vf1 <- varExp()
vf1 <- initialize(vf1, data = Orthodont)
needUpdate(vf1)
```

needUpdate.modelStruct	Check modelStruct Updating	needUpdate.modelStruct
-------------------------------	----------------------------	-------------------------------

This method function checks if any of the elements of `object` needs to be updated. Updating of objects usually takes place in iterative algorithms in which auxiliary quantities associated with the object, and not being optimized over, may change.

```
needUpdate(object)
```

ARGUMENTS

object: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

VALUE

a logical value indicating whether any element of `object` needs to be updated.

EXAMPLE

```
lms1 <- lmeStruct(reStruct = reStruct(pdDiag(diag(2), ~ age)),
  varStruct = varPower(form = ~ age))
needUpdate(lms1)
```

nlme	Nonlinear Mixed-Effects Models	nlme
-------------	--------------------------------	-------------

This generic function fits a nonlinear mixed-effects model in the formulation described in Lindstrom and Bates (1990) but allowing for nested random effects. The within-group errors are allowed to be correlated and/or have unequal variances.

```
nlme(model, data, fixed, random, groups, start, correlation,
  weights, subset, method, na.action, naPattern, control,
  verbose)
```

ARGUMENTS

model: a nonlinear model formula, with the response on the left of a `~` operator and an expression involving parameters and covariates on the right, or an `nlsList` object. If `data` is given, all names used in the formula should be defined as parameters or variables in the data frame. The method function `nlme.nlsList` is documented separately.

- fixed:** a two-sided linear formula of the form $f_1 + \dots + f_n \sim x_1 + \dots + x_m$, or a list of two-sided formulas of the form $f_1 \sim x_1 + \dots + x_m$, with possibly different models for different parameters. The f_1, \dots, f_n are the names of the parameters included on the right hand side of `model` and the $x_1 + \dots + x_m$ expressions define linear models for these parameters (when the left hand side of the formula contains several parameters, they all are assumed to follow the same linear model, described by the right hand side expression). A 1 on the right hand side of the formula(s) indicates a single fixed effects for the corresponding parameter(s).
- data:** an optional data frame containing the variables named in `model`, `fixed`, `random`, `correlation`, `weights`, `subset`, and `naPattern`. By default the variables are taken from the environment from which `nlme` is called.
- random:** optionally, any of the following: (i) a two-sided formula of the form $r_1 + \dots + r_n \sim x_1 + \dots + x_m$ | $g_1 / \dots / g_Q$, with r_1, \dots, r_n naming parameters included on the right hand side of `model`, $x_1 + \dots + x_m$ specifying the random-effects model for these random effects and $g_1 / \dots / g_Q$ the grouping structure (Q may be equal to 1, in which case no / is required). The random effects formula will be repeated for all levels of grouping, in the case of multiple levels of grouping; (ii) a two-sided formula of the form $r_1 + \dots + r_n \sim x_1 + \dots + x_m$, a list of two-sided formulas of the form $r_1 \sim x_1 + \dots + x_m$, with possibly different random-effects models for different parameters, a `pdMat` object with a two-sided formula, or list of two-sided formulas (i.e. a non-NULL value for `formula(random)`), or a list of `pdMat` objects with two-sided formulas, or lists of two-sided formulas. In this case, the grouping structure formula will be given in `groups`, or derived from the data used to fit the nonlinear mixed-effects model, which should inherit from class `groupedData`; (iii) a named list of formulas, lists of formulas, or `pdMat` objects as in (ii), with the grouping factors as names. The order of nesting will be assumed the same as the order of the elements in the list; (iv) an `reStruct` object. See the documentation on `pdClasses` for a description of the available `pdMat` classes. Defaults to `fixed`, resulting in all fixed effects having also random effects.
- groups:** an optional one-sided formula of the form $\sim g_1$ (single level of nesting) or $\sim g_1 / \dots / g_Q$ (multiple levels of nesting), specifying the partitions of the data over which the random effects vary. g_1, \dots, g_Q must evaluate to factors in `data`. The order of nesting, when multiple levels are present, is taken from left to right (i.e. g_1 is the first level, g_2 the second, etc.).
- start:** an optional numeric vector, or list of initial estimates for the fixed effects and random effects. If declared as a numeric vector, it is converted internally to a list with a single component `fixed`, given by the vector. The `fixed` component is required, unless the model function inherits from class `selfStart`, in which case initial values will be derived from a call to `nlsList`. An optional `random` component is used to specify initial values for the random effects and should consist of a matrix, or a list of matrices with length equal to the number of grouping levels. Each matrix should have as many rows as the number of groups

at the corresponding level and as many columns as the number of random effects in that level.

`correlation`: an optional `corStruct` object describing the within-group correlation structure. See the documentation of `corClasses` for a description of the available `corStruct` classes. Defaults to `NULL`, corresponding to no within-group correlations.

`weights`: an optional `varFunc` object or one-sided formula describing the within-group heteroscedasticity structure. If given as a formula, it is used as the argument to `varFixed`, corresponding to fixed variance weights. See the documentation on `varClasses` for a description of the available `varFunc` classes. Defaults to `NULL`, corresponding to homoscedastic within-group errors.

`subset`: an optional expression indicating the subset of the rows of data that should be used in the fit. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

`method`: a character string. If `"REML"` the model is fit by maximizing the restricted log-likelihood. If `"ML"` the log-likelihood is maximized. Defaults to `"REML"`.

`na.action`: a function that indicates what should happen when the data contain `NA`s. The default action (`na.fail`) causes `nlme` to print an error message and terminate if there are any incomplete observations.

`naPattern`: an expression or formula object, specifying which returned values are to be regarded as missing.

`control`: a list of control values for the estimation algorithm to replace the default values returned by the function `nlmeControl`. Defaults to an empty list.

`verbose`: an optional logical value. If `TRUE` information on the evolution of the iterative algorithm is printed. Default is `FALSE`.

VALUE

an object of class `nlme` representing the nonlinear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `nlmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixef`, and `ranef` can be used to extract some of its components.

REFERENCES

The model formulation and computational methods are described in Lindstrom, M.J. and Bates, D.M. (1990). The variance-covariance parametrizations are described in Pinheiro, J.C. and Bates., D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997).

The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).
 Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.
 Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.
 Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", Biometrics, 38, 963-974.
 Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.
 Lindstrom, M.J. and Bates, D.M. (1990) "Nonlinear Mixed Effects Models for Repeated Measures Data", Biometrics, 46, 673-687.
 Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", Statistics and Computing, 6, 289-296.
 Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`nlmeControl`, `nlme.nlsList`, `nlmeObject`, `nlsList`, `reStruct`, `pdClasses`, `corClasses`, `varClasses`

EXAMPLE

```
## all parameters as fixed and random effects
fm1 <- nlme(weight ~ SSlogis(Time, Asym, xmid, scal),
            data = Soybean, fixed = Asym + xmid + scal ~ 1,
            start = c(18, 52, 7.5))
## only Asym and xmid as random, with a diagonal covariance
fm2 <- nlme(weight ~ SSlogis(Time, Asym, xmid, scal),
            data = Soybean, fixed = Asym + xmid + scal ~ 1,
            random = pdDiag(Asym + xmid ~ 1),
            start = c(18, 52, 7.5))
```

If the random effects names defined in `random` are a subset of the `lmList` object coefficient names, initial estimates for the covariance matrix of the random effects are obtained (overwriting any values given in `random`). `formula(fixed)` and the `data` argument in the calling sequence used to obtain `fixed` are passed as the `fixed` and `data` arguments to `nlme.formula`, together with any other additional arguments in the function call. See the documentation on `nlme.formula` for a description of that function.

```
nlme(model, data, fixed, random, groups, start, correlation, weights,
      subset, method, na.action, naPattern, control, verbose)
```

ARGUMENTS

`model`: an object inheriting from class `nlsList`, representing a list of `nls` fits with a common model.

`data`: this argument is included for consistency with the generic function. It is ignored in this method function.

`random`: an optional one-sided linear formula with no conditioning expression, or a `pdMat` object with a `formula` attribute. Multiple levels of grouping are not allowed with this method function. Defaults to a formula consisting of the right hand side of `formula(fixed)`.

`other arguments`: identical to the arguments in the generic function call. See the documentation on `nlme`.

VALUE

an object of class `nlme` representing the linear mixed-effects model fit. Generic functions such as `print`, `plot` and `summary` have methods to show the results of the fit. See `nlmeObject` for the components of the fit. The functions `resid`, `coef`, `fitted`, `fixef`, and `ranef` can be used to extract some of its components.

REFERENCES

The computational methods are described in Bates, D.M. and Pinheiro, J.C. (1998) and follow on the general framework of Lindstrom, M.J. and Bates, D.M. (1988). The model formulation is described in Laird, N.M. and Ware, J.H. (1982). The variance-covariance parametrizations are described in Pinheiro, J.C. and Bates., D.M. (1996). The different correlation structures available for the `correlation` argument are described in Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994), Littell, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996), and Venables, W.N. and Ripley, B.D. (1997). The use of variance functions for linear and nonlinear mixed effects models is presented in detail in Davidian, M. and Giltinan, D.M. (1995).

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

Box, G.E.P., Jenkins, G.M., and Reinsel G.C. (1994) "Time Series Analysis: Forecasting and Control", 3rd Edition, Holden-Day.

Davidian, M. and Giltinan, D.M. (1995) "Nonlinear Mixed Effects Models for Repeated Measurement Data", Chapman and Hall.

Laird, N.M. and Ware, J.H. (1982) "Random-Effects Models for Longitudinal Data", Biometrics, 38, 963-974.

Lindstrom, M.J. and Bates, D.M. (1988) "Newton-Raphson and EM Algorithms for Linear Mixed-Effects Models for Repeated-Measures Data", Journal of the American Statistical Association, 83, 1014-1022.

Littel, R.C., Milliken, G.A., Stroup, W.W., and Wolfinger, R.D. (1996) "SAS Systems for Mixed Models", SAS Institute.

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", Statistics and Computing, 6, 289-296.

Venables, W.N. and Ripley, B.D. (1997) "Modern Applied Statistics with S-plus", 2nd Edition, Springer-Verlag.

SEE ALSO

`nlme`, `lmList`, `nlmeObject`

EXAMPLE

```
fml <- nlsList(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean)
fm2 <- nlme(fml)
```

nlmeControl	Control Values for nlme Fit	nlmeControl
--------------------	-----------------------------	--------------------

The values supplied in the function call replace the defaults and a list with all possible arguments is returned. The returned list is used as the `control` argument to the `nlme` function.

```
nlmeControl(maxIter, pnlsMaxIter, msMaxIter, minScale, tolerance,
            niterEM, pnlsTol, msTol, msScale, returnObject,
            msVerbose, gradHess, apVar, .relStep, natural,
            natUnconstrained, sigma)
```

ARGUMENTS

maxIter: maximum number of iterations for the `nlme` optimization algorithm. Default is 50.

pnlsMaxIter: maximum number of iterations for the PNLs optimization step inside the `nlme` optimization. Default is 7.

msMaxIter: maximum number of iterations for the `ms` optimization step inside the `nlme` optimization. Default is 50.

`minScale`: minimum factor by which to shrink the default step size in an attempt to decrease the sum of squares in the PNLs step. Default 0.001.

`tolerance`: tolerance for the convergence criterion in the nlme algorithm. Default is 1e-6.

`niterEM`: number of iterations for the EM algorithm used to refine the initial estimates of the random effects variance-covariance coefficients. Default is 25.

`pnlsTol`: tolerance for the convergence criterion in PNLs step. Default is 1e-3.

`msTol`: tolerance for the convergence criterion in `ms`, passed as the `rel.tolerance` argument to the function (see documentation on `ms`). Default is 1e-7.

`msScale`: scale function passed as the `scale` argument to the `ms` function (see documentation on that function). Default is `lmeScale`.

`returnObject`: a logical value indicating whether the fitted object should be returned when the maximum number of iterations is reached without convergence of the algorithm. Default is `FALSE`.

`msVerbose`: a logical value passed as the `trace` argument to `ms` (see documentation on that function). Default is `FALSE`.

`gradHess`: a logical value indicating whether numerical gradient vectors and Hessian matrices of the log-likelihood function should be used in the `ms` optimization. This option is only available when the correlation structure (`corStruct`) and the variance function structure (`varFunc`) have no "varying" parameters and the `pdMat` classes used in the random effects structure are `pdSymm` (general positive-definite), `pdDiag` (diagonal), `pdIdent` (multiple of the identity), or `pdCompSymm` (compound symmetry). Default is `TRUE`.

`apVar`: a logical value indicating whether the approximate covariance matrix of the variance-covariance parameters should be calculated. Default is `TRUE`.

`.relStep`: relative step for numerical derivatives calculations. Default is `.Machine$double.eps1/3`.

`natural`: a logical value, or a named list of logical values, indicating whether a natural parameterization should be used for the model structures, when the approximate covariance matrix of the estimators is calculated. If given as a single logical value, it is used for all model structures (`pdMat`, `corStruct`, and `varFunc` objects) used in the fit. If given as a list, it must have names `reStruct`, `corStruct`, and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`natUnconstrained`: a logical value, or a named list of logical values, indicating whether an unconstrained parameterization should be used for the natural parameters of the model structures. If given as a single logical value, it is used for all model structures (`pdMat`, `corStruct` and `varFunc` objects) used in the fit. If given as

a list, it must have names `reStruct`, `corStruct` and `varStruct` corresponding to the model structures used in the fit. Default is `TRUE`.

`sigma`: a numeric value indicating the value at which the within-group standard error should be kept fixed during the optimization of the objective function. Defaults to `NULL`, in which case the within-group standard error is estimated together with the other model parameters. Must be a non-negative numeric value - setting it to zero has the same effect as the default (`NULL`).

VALUE
a list with components for each of the possible arguments.

SEE ALSO
`nlme`, `ms`, `nlmeScale`

EXAMPLE

```
# decrease the maximum number iterations in the ms call and
# request that information on the evolution of the ms iterations be printed
nlmeControl(msMaxIter = 20, msVerbose = TRUE)
```

nlmeObject	Fitted nlme Object	nlmeObject
-------------------	--------------------	-------------------

An object returned by the `nlme` function, inheriting from class `nlme`, also inheriting from class `lme`, and representing a fitted nonlinear mixed-effects model. Objects of this class have methods for the generic functions `anova`, `coef`, `fitted`, `fixef`, `formula`, `getGroups`, `getResponse`, `intervals`, `logLik`, `pairs`, `plot`, `predict`, `print`, `ranef`, `residuals`, `summary`, and `update`.

VALUE
The following components must be included in a legitimate `nlme` object.

COMPONENTS

`apVar`: an approximate covariance matrix for the variance-covariance coefficients. If `apVar = FALSE` in the list of control values used in the call to `nlme`, this component is equal to `NULL`.

`call`: a list containing an image of the `nlme` call that produced the object.

`coefficients`: a list with two components, `fixed` and `random`, where the first is a vector containing the estimated fixed effects and the second is a list of matrices with the estimated random effects for each level of grouping. For each matrix in the `random` list, the columns refer to the random effects and the rows to the groups.

`contrasts`: a list with the contrasts used to represent factors in the fixed effects formula and/or random effects formula. This information is important for making predictions from a new data frame in which not all levels of the original factors are observed. If no factors are used in the nlme model, this component will be an empty list.

`dims`: a list with basic dimensions used in the nlme fit, including the components `N` - the number of observations in the data, `Q` - the number of grouping levels, `qvec` - the number of random effects at each level from innermost to outermost (last two values are equal to zero and correspond to the fixed effects and the response), `ngrps` - the number of groups at each level from innermost to outermost (last two values are one and correspond to the fixed effects and the response), and `ncol` - the number of columns in the model matrix for each level of grouping from innermost to outermost (last two values are equal to the number of fixed effects and one).

`fitted`: a data frame with the fitted values as columns. The leftmost column corresponds to the population fixed effects (corresponding to the fixed effects only) and successive columns from left to right correspond to increasing levels of grouping.

`fixDF`: a list with components `X` and `terms` specifying the denominator degrees of freedom for, respectively, t-tests for the individual fixed effects and F-tests for the fixed-effects terms in the models.

`groups`: a data frame with the grouping factors as columns. The grouping level increases from left to right.

`logLik`: the (restricted) log-likelihood at convergence.

`map`: a list with components `fmap`, `rmap`, `rmapRel`, and `bmap`, specifying various mappings for the fixed and random effects, used to generate predictions from the fitted object.

`method`: the estimation method: either "ML" for maximum likelihood, or "REML" for restricted maximum likelihood.

`modelStruct`: an object inheriting from class `nlmeStruct`, representing a list of mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

`numIter`: the number of iterations used in the iterative algorithm.

`residuals`: a data frame with the residuals as columns. The leftmost column corresponds to the population residuals and successive columns from left to right correspond to increasing levels of grouping.

`sigma`: the estimated within-group error standard deviation.

`varFix`: an approximate covariance matrix of the fixed effects estimates.

SEE ALSO

`nlme`, `nlmeStruct`

nlmeStruct	Nonlinear Mixed-Effects Structure	nlmeStruct
-------------------	-----------------------------------	-------------------

A nonlinear mixed-effects structure is a list of model components representing different sets of parameters in the nonlinear mixed-effects model. An `nlmeStruct` list must contain at least a `reStruct` object, but may also contain `corStruct` and `varFunc` objects. `NULL` arguments are not included in the `nlmeStruct` list.

```
nlmeStruct(reStruct, corStruct, varStruct)
```

ARGUMENTS

`reStruct`: a `reStruct` representing a random effects structure.

`corStruct`: an optional `corStruct` object, representing a correlation structure. Default is `NULL`.

`varStruct`: an optional `varFunc` object, representing a variance function structure. Default is `NULL`.

VALUE

a list of model components determining the parameters to be estimated for the associated nonlinear mixed-effects model.

SEE ALSO

`nlme`, `reStruct`, `corClasses`, `varClasses`

EXAMPLE

```
nlms1 <- nlmeStruct(reStruct(~ age), corAR1(), varPower())
```

Data is partitioned according to the levels of the grouping factor defined in `model` and individual `nls` fits are obtained for each data partition, using the model defined in `model`.

```
nlsList(model, data, start, control, level, na.action, pool)
```

ARGUMENTS

model: either a nonlinear model formula, with the response on the left of a `~` operator and an expression involving parameters, covariates, and a grouping factor separated by the `|` operator on the right, or a `selfStart` function. The method function `nlsList.selfStart` is documented separately.

data: a data frame in which to interpret the variables named in `model`.

start: an optional named list with initial values for the parameters to be estimated in `model`. It is passed as the `start` argument to each `nls` call and is required when the nonlinear function in `model` does not inherit from class `selfStart`.

control: a list of control values passed as the `control` argument to `nls`. Defaults to an empty list.

level: an optional integer specifying the level of grouping to be used when multiple nested levels of grouping are present.

na.action: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes `nlsList` to print an error message and terminate if there are any incomplete observations.

pool: an optional logical value that is preserved as an attribute of the returned value. This will be used as the default for `pool` in calculations of standard deviations or standard errors for summaries.

VALUE

a list of `nls` objects with as many components as the number of groups defined by the grouping factor. Generic functions such as `coef`, `fixef`, `lme`, `pairs`, `plot`, `predict`, `ranef`, `summary`, and `update` have methods that can be applied to an `nlsList` object.

SEE ALSO

`nls`, `nlme.nlsList`.

EXAMPLE

```
fml <- nlsList(weight ~ SSlogis(Time, Asym, xmid, scal) | Plot,
               Soybean)
```

nlsList.selfStart**nlsList Fit from a selfStart Function****nlsList.selfStart**

The response variable and primary covariate in `formula(data)` are used together with `model` to construct the nonlinear model formula. This is used in the `nls` calls and, because a selfStarting model function can calculate initial estimates for its parameters from the data, no starting estimates need to be provided.

```
nlsList(model, data, start, control, level, na.action, pool)
```

ARGUMENTS

model: a `selfStart` model function, which calculates initial estimates for the model parameters from data.

data: a data frame in which to interpret the variables in `model`. Because no grouping factor can be specified in `model`, data must inherit from class `groupedData`.

other arguments: identical to the arguments in the generic function call. See the documentation on `nlsList`.

VALUE

a list of `nls` objects with as many components as the number of groups defined by the grouping factor. A `NULL` value is assigned to the components corresponding to clusters for which the `nls` algorithm failed to converge. Generic functions such as `coef`, `fixef`, `lme`, `pairs`, `plot`, `predict`, `ranef`, `summary`, and `update` have methods that can be applied to an `nlsList` object.

SEE ALSO

`selfStart`, `groupedData`, `nls`, `nlme.nlsList`, `nlsList.formula`

EXAMPLE

```
fml <- nlsList(SSlogis, Soybean)
```

NLSstClosestX**Inverse Interpolation****NLSstClosestX**

Use inverse linear interpolation to approximate the x value at which the function represented by xy is equal to $yval$.

```
NLSstClosestX(xy, yval)
```

ARGUMENTS

xy: a `sortedXyData` object

yval: the numeric value on the y scale to get close to

VALUE

A single numeric value on the x scale.

EXAMPLE

```
DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density),
NLSstClosestX( DN.srt, 1.0 )
```

NLSstLfAsymptote	Horizontal Asymptote on the Left Side	NLSstLfAsymptote
-------------------------	---------------------------------------	-------------------------

Provide an initial guess at the horizontal asymptote on the left side (i.e. small values of x) of the graph of y versus x from the xy object. Primarily used within initial functions for self-starting nonlinear regression models.

```
NLSstLfAsymptote(xy)
```

ARGUMENTS

xy: a sortedXyData object

VALUE

A single numeric estimating the horizontal asymptote for small x .

EXAMPLE

```
DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density),
NLSstLfAsymptote( DN.srt )
```

NLSstRtAsymptote	Horizontal Asymptote on the Right Side	NLSstRtAsymptote
-------------------------	--	-------------------------

Provide an initial guess at the horizontal asymptote on the right side (i.e. large values of x) of the graph of y versus x from the xy object. Primarily used within initial functions for self-starting nonlinear regression models.

```
NLSstRtAsymptote(xy)
```

ARGUMENTS

xy: a sortedXyData object

VALUE

A single numeric estimating the horizontal asymptote for large x .

EXAMPLE

```
DNase.2 <- DNase[ DNase$Run == "2", ]
DN.srt <- sortedXyData( expression(log(conc)), expression(density),
NLSstRtAsymptote( DN.srt )
```

Scatter plots of the values being compared are generated for each pair of coefficients in `object`. Different symbols (colors) are used for each object being compared and values corresponding to the same group are joined by a line, to facilitate comparison of fits. If only two coefficients are present, the `trellis` function `xyplot` is used; otherwise the `trellis` function `splo` is used.

```
pairs(object, subset, key, ...)
```

ARGUMENTS

`object`: an object of class `compareFits`.

`subset`: an optional logical or integer vector specifying which rows of `object` should be used in the plots. If missing, all rows are used.

`key`: an optional logical value, or list. If `TRUE`, a legend is included at the top of the plot indicating which symbols (colors) correspond to which objects being compared. If `FALSE`, no legend is included. If given as a list, `key` is passed down as an argument to the `trellis` function generating the plots (`splo` or `xyplot`). Defaults to `TRUE`.

`...`: optional arguments passed down to the `trellis` function generating the plots.

VALUE

Pairwise scatter plots of the values being compared, with different symbols (colors) used for each object under comparison.

SEE ALSO

`compareFits`, `plot.compareFits`, `xyplot`, `splo`

EXAMPLE

```
fm1 <- lmList(Orthodont)
fm2 <- lme(Orthodont)
pairs(compareFits(coef(fm1), coef(fm2)))
```


Diagnostic plots for the linear model fits corresponding to the `object` components are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display. The expression on the right hand side of the formula, before a `|` operator, must evaluate to a data frame with at least two columns. If the data frame has two columns, a scatter plot of the two variables is displayed (the Trellis function `xypoint` is used). Otherwise, if more than two columns are present, a scatter plot matrix with pairwise scatter plots of the columns in the data frame is displayed (the Trellis function `spoint` is used).

```
pairs(object, form, label, id, idLabels, grid, subset, ...)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

form: an optional one-sided formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `"."`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. The expression on the right hand side of `form`, and to the left of the `|` operator, must evaluate to a data frame with at least two columns. Default is `~coef(.)`, corresponding to a pairs plot of the coefficients of `object`.

id: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for an outlier test based on the Mahalanobis distances of the estimated random effects. Groups with random effects distances greater than the 1-value percentile of the appropriate chi-square distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify points in the plot. If missing, no points are identified.

idLabels: an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the points identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified points. Default is the innermost grouping factor.

grid: an optional logical value indicating whether a grid should be added to plot. Default is `FALSE`.

subset: an optional expression indicating the subset of the observations that should be used in the plot. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

...: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot.

SEE ALSO

`lmList`, `xyplot`, `splom`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
# scatter plot of coefficients by gender, identifying
# unusual subjects
pairs(fml, ~ coef(.) | Sex, id = 0.1, adj = -0.5)
# scatter plot of estimated random effects
pairs(fml, ~ ranef(.))
```

pairs.lme

Pairs Plot of an lme Object

pairs.lme

Diagnostic plots for the linear mixed-effects fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display. The expression on the right hand side of the formula, before a `|` operator, must evaluate to a data frame with at least two columns. If the data frame has two columns, a scatter plot of the two variables is displayed (the Trellis function `xyplot` is used). Otherwise, if more than two columns are present, a scatter plot matrix with pairwise scatter plots of the columns in the data frame is displayed (the Trellis function `splom` is used).

```
pairs(object, form, label, id, idLabels, grid, subset, ...)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

form: an optional one-sided formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `"."`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. The expression on the right hand side of `form`, and to the left of the `|` operator, must evaluate to a data frame with at least two columns. Default is `~coef(.)`, corresponding to a pairs plot of the coefficients evaluated at the innermost level of nesting.

`id`: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for an outlier test based on the Mahalanobis distances of the estimated random effects. Groups with random effects distances greater than the 1-value percentile of the appropriate chi-square distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify points in the plot. If missing, no points are identified.

`idLabels`: an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the points identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified points. Default is the innermost grouping factor.

`grid`: an optional logical value indicating whether a grid should be added to plot. Default is `FALSE`.

`...`: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot.

SEE ALSO

`lme`, `xyplot`, `splo`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
# scatter plot of coefficients by gender, identifying
# unusual subjects
pairs(fml, ~ coef(., augFrame = T) | Sex, id = 0.1, adj = -
0.5)
# scatter plot of estimated random effects
pairs(fml, ~ ranef(.))
```

This function is a constructor for the `pdBand` class, representing a banded positive-definite matrix. If the matrix associated with `object` is of dimension n and order r , it is represented by $r(2n - r + 1)/2$ unrestricted parameters, using the log-Cholesky parameterization described in Pinheiro and Bates (1996). When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdBand` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the unrestricted coefficients of the log-Cholesky parameterization of the underlying positive-definite matrix.

```
pdBand(value, form, nam, data, ord)
```

ARGUMENTS

- `value`: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- `nam`: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.
- `ord`: an optional integer specifying the order of the banded matrix represented by the returned `pdBand` object, defined as the number of nonzero diagonals in the upper-triangular part of the matrix. Defaults to 2.

VALUE

a `pdBand` object representing a banded positive-definite matrix, also inheriting from class `pdMat`.

REFERENCES

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

SEE ALSO

```
as.matrix.pdMat, coef.pdMat, matrix<- .pdMat
```

EXAMPLE

```
pd1 <- pdBand(diag(1:3), nam = c("A", "B", "C"))
pd1
```

pdBandNat Banded Positive-Definite Matrix in Natural Parameterization **pdBandNat**

This function is a constructor for the `pdBandNat` class, representing a banded positive-definite matrix, using a natural parameterization. If the matrix associated with `object` is of dimension n and order r , it is represented by $r(2n - r + 1)/2$ parameters. Letting Σ_{ij} denote the ij th element of the underlying positive definite matrix ($|i - j| < r$) and $\rho_{ij} = \Sigma_{ij} / \sqrt{\Sigma_{ii}\Sigma_{jj}}$, $i \neq j$ denote the associated "correlations", if `unconstrained = TRUE`, the "natural" parameters are given by $\log(\Sigma_{ii})$, $i = 1, \dots, n$ and $\log((1 + \rho_{ij})/(1 - \rho_{ij}))$, $|i - j| < r$. Note that, in this case, all natural parameters are individually unrestricted, but not jointly unrestricted (meaning that not all unrestricted vectors would give positive-definite matrices). If `unconstrained = FALSE`, the Σ_{ij} are used to parameterize the returned object. This class does not define an unrestricted parameterization and, therefore, should NOT be used for optimization. It is mostly used for deriving approximate confidence intervals on parameters following the optimization of an objective function. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdBand` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the natural parameters of the underlying positive-definite matrix.

```
pdBandNat(value, form, nam, data, ord, unconstrained)
```

ARGUMENTS

- value**: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form**: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.

- `nam`: an optional vector of character strings specifying the row/column names for the matrix represented by object. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.
- `ord`: an optional integer specifying the order of the banded matrix represented by the returned `pdBandNat` object, defined as the number of nonzero diagonals in the upper-triangular part of the matrix. Defaults to 2.
- `unconstrained`: an optional logical value indicating whether an unconstrained natural parameterization should be used. Defaults to `TRUE`.

VALUE

a `pdBandNat` object representing a banded positive-definite matrix in natural parameterization, also inheriting from class `pdMat`.

SEE ALSO

`asNatural.pdBand`, `pdBand`

EXAMPLE

```
pdBandNat(diag(1:3))
```

pdBlocked	Positive-Definite Block Diagonal Matrix	pdBlocked
------------------	---	------------------

This function is a constructor for the `pdBlocked` class, representing a positive-definite block-diagonal matrix. Each block-diagonal element of the underlying matrix is itself a positive-definite matrix and is represented internally as an individual `pdMat` object. When `value` is `numeric(0)`, a list of uninitialized `pdMat` objects, a list of one-sided formulas, or a list of vectors of character strings, object is returned as an uninitialized `pdBlocked` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is a list of initialized `pdMat` objects, object will be constructed from the list obtained by applying `as.matrix` to each of the `pdMat` elements of `value`. Finally, if `value` is a list of numeric vectors, they are assumed to represent the unrestricted coefficients of the block-diagonal elements of the underlying positive-definite matrix.

```
pdBlocked(value, form, nam, data, pdClass)
```

ARGUMENTS

- `value`: an optional list with elements to be used as the `value` argument to other `pdMat` constructors. These include: `pdMat` objects, positive-definite matrices, one-sided linear formulas, vectors of character strings, or numeric vectors. All elements in the list must be similar (e.g. all one-sided formulas, or all numeric vectors). Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional list of one-sided linear formulas specifying the row/column names for the block-diagonal elements of the matrix represented by `object`. Because factors may be present in `form`, the formulas needs to be evaluated on a `data.frame` to resolve the names they define. This argument is ignored when `value` is a list of one-sided formulas. Defaults to `NULL`.
- `nam`: an optional list of vector of character strings specifying the row/column names for the block-diagonal elements of the matrix represented by `object`. Each of its components must have length equal to the dimension of the corresponding block-diagonal element and unreplicated elements. This argument is ignored when `value` is a list of vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on any `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.
- `pdClass`: an optional vector of character strings naming the `pdMat` classes to be assigned to the individual blocks in the underlying matrix. If a single class is specified, it is used for all block-diagonal elements. This argument will only be used when `value` is missing, or its elements are not `pdMat` objects. Defaults to `"pdSymm"`.

VALUE

a `pdBlocked` object representing a positive-definite block-diagonal matrix, also inheriting from class `pdMat`.

SEE ALSO

`as.matrix.pdMat`, `coef.pdMat`, `matrix<-pdMat`

EXAMPLE

```
pd1 <- pdBlocked(list(diag(1:2), diag(c(0.1, 0.2, 0.3))),
  nam = list(c("A","B"), c("a1", "a2", "a3")))
```

Standard classes of positive-definite matrices (`pdMat`) structures available in the `nlme` library.

STANDARD CLASSES

`pdSymm`: general positive-definite matrix, with no additional structure

`pdDiag`: diagonal

`pdIdent`: multiple of an identity

`pdCompSymm`: compound symmetry structure (constant diagonal and constant off-diagonal elements)

`pdBand`: banded positive-definite matrix

`pdBand`: banded positive-definite matrix

`pdBlocked`: block-diagonal matrix, with diagonal blocks of any "atomic" `pdMat` class

NOTE

Users may define their own `pdMat` classes by specifying a constructor function and, at a minimum, methods for the functions `pdConstruct`, `pdMatrix` and `coef`. For examples of these functions, see the methods for classes `pdSymm` and `pdDiag`.

SEE ALSO

`pdBand`, `pdCompSymm`, `pdDiag`, `pdIdent`, `pdSymm`, `pdBand`, `pdBlocked`

This function is a constructor for the `pdCompSymm` class, representing a positive-definite matrix with compound symmetry structure (constant diagonal and constant off-diagonal elements). The underlying matrix is represented by 2 unrestricted parameters. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdCompSymm` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector of length 2, it is assumed to represent the unrestricted coefficients of the underlying positive-definite matrix.

```
pdCompSymm(value, form, nam, data)
```

ARGUMENTS

- `value`: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector of length 2. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- `nam`: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

a `pdCompSymm` object representing a positive-definite matrix with compound symmetry structure, also inheriting from class `pdMat`.

SEE ALSO

```
as.matrix.pdMat, coef.pdMat, matrix<-.pdMat
```

EXAMPLE

```
pd1 <- pdCompSymm(diag(3) + 1, nam = c("A", "B", "C"))
pd1
```

pdConstruct	Construct pdMat Objects	pdConstruct
--------------------	-------------------------	--------------------

This function is an alternative constructor for the `pdMat` class associated with `object` and is mostly used internally in other functions. See the documentation on the principal constructor function, generally with the same name as the `pdMat` class of `object`.

```
pdConstruct(object, value, form, nam, data)
```

ARGUMENTS

- object**: an object inheriting from class `pdMat`, representing a positive definite matrix.
- value**: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form**: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- nam**: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- data**: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

a `pdMat` object representing a positive-definite matrix, inheriting from the same classes as `object`.

SEE ALSO

```
pdCompSymm, pdDiag, pdIdent, pdSymm
```

EXAMPLE

```
pd1 <- pdSymm()
pdConstruct(pd1, diag(1:4))
```

This function give an alternative constructor for the `pdBlocked` class, representing a positive-definite block-diagonal matrix. Each block-diagonal element of the underlying matrix is itself a positive-definite matrix and is represented internally as an individual `pdMat` object. When `value` is `numeric(0)`, a list of uninitialized `pdMat` objects, a list of one-sided formulas, or a list of vectors of character strings, `object` is returned as an uninitialized `pdBlocked` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is a list of initialized `pdMat` objects, `object` will be constructed from the list obtained by applying `as.matrix` to each of the `pdMat` elements of `value`. Finally, if `value` is a list of numeric vectors, they are assumed to represent the unrestricted coefficients of the block-diagonal elements of the underlying positive-definite matrix.

```
pdConstruct(object, value, form, nam, data, pdClass)
```

ARGUMENTS

- `object`: an object inheriting from class `pdBlocked`, representing a positive definite block-diagonal matrix.
- `value`: an optional list with elements to be used as the `value` argument to other `pdMat` constructors. These include: `pdMat` objects, positive-definite matrices, one-sided linear formulas, vectors of character strings, or numeric vectors. All elements in the list must be similar (e.g. all one-sided formulas, or all numeric vectors). Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional list of one-sided linear formula specifying the row/column names for the block-diagonal elements of the matrix represented by `object`. Because factors may be present in `form`, the formulas needs to be evaluated on a `data.frame` to resolve the names they defines. This argument is ignored when `value` is a list of one-sided formulas. Defaults to `NULL`.
- `nam`: an optional list of vector of character strings specifying the row/column names for the block-diagonal elements of the matrix represented by `object`. Each of its components must have length equal to the dimension of the corresponding block-diagonal element and unreplicated elements. This argument is ignored when `value` is a list of vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

pdClass: an optional vector of character strings naming the `pdMat` classes to be assigned to the individual blocks in the underlying matrix. If a single class is specified, it is used for all block-diagonal elements. This argument will only be used when `value` is missing, or its elements are not `pdMat` objects. Defaults to `"pdSymm"`.

VALUE

a `pdBlocked` object representing a positive-definite block-diagonal matrix, also inheriting from class `pdMat`.

SEE ALSO

`as.matrix.pdMat`, `coef.pdMat`, `matrix<-.pdMat`

EXAMPLE

```
pd1 <- pdBlocked(list(c("A", "B"), c("a1", "a2", "a3")))
pdConstruct(pd1, list(diag(1:2), diag(c(0.1, 0.2, 0.3))))
```

pdDiag	Diagonal Positive-Definite Matrix	pdDiag
---------------	-----------------------------------	---------------

This function is a constructor for the `pdDiag` class, representing a diagonal positive-definite matrix. If the matrix associated with `object` is of dimension `n`, it is represented by `n` unrestricted parameters, given by the logarithm of the square-root of the diagonal values. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdDiag` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the unrestricted coefficients of the underlying positive-definite matrix.

```
pdDiag(value, form, nam, data)
```

ARGUMENTS

- value:** an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector of length equal to the dimension of the underlying positive-definite matrix. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form:** an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- nam:** an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.

data: an optional data frame in which to evaluate the variables named in **value** and **form**. It is used to obtain the levels for **factors**, which affect the dimensions and the row/column names of the underlying matrix. If **NULL**, no attempt is made to obtain information on **factors** appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

a **pdDiag** object representing a diagonal positive-definite matrix, also inheriting from class **pdMat**.

SEE ALSO

as.matrix.pdMat, **coef.pdMat**, **matrix<-**.**pdMat**

EXAMPLE

```
pd1 <- pdDiag(diag(1:3), nam = c("A", "B", "C"))
pd1
```

pdFactor	Square-Root Factor of a Positive-Definite Matrix	pdFactor
-----------------	--	-----------------

A square-root factor of the positive-definite matrix represented by **object** is obtained. Letting Σ denote a positive-definite matrix, a square-root factor of Σ is any square matrix L such that $\Sigma = L^t L$. This function extracts L .

```
pdFactor(object, ...)
```

ARGUMENTS

object: an object inheriting from class **pdMat**, representing a positive definite matrix, which must have been initialized (i.e. `length(coef(object)) > 0`).

...: some methods for the generic function may require additional arguments.

VALUE

a vector with a square-root factor of the positive-definite matrix associated with **object** stacked column-wise.

NOTE

This function is used intensively in optimization algorithms and its value is returned as a vector for efficiency reasons. The **pdMatrix** function can be used to obtain square-root factors in matrix form.

SEE ALSO

pdMatrix

EXAMPLE

```
pd1 <- pdCompSymm(4 * diag(3) + 1)
pdFactor(pd1)
```

This method function extracts square-root factors of the positive-definite matrices corresponding to the `pdMat` elements of `object`.

```
pdFactor(object)
```

ARGUMENTS

`object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

VALUE

a vector with square-root factors of the positive-definite matrices corresponding to the elements of `object` stacked column-wise.

NOTE

This function is used intensively in optimization algorithms and its value is returned as a vector for efficiency reasons. The `pdMatrix` function can be used to obtain square-root factors in matrix form.

SEE ALSO

`pdMatrix.reStruct`, `pdFactor.pdMat`

EXAMPLE

```
rsl <- reStruct(pdSymm(diag(3), form=~ Sex+age, data=Orthodont))
pdFactor(rsl)
```

This function is a constructor for the `pdIdent` class, representing a multiple of the identity positive-definite matrix. The matrix associated with `object` is represented by 1 unrestricted parameter, given by the logarithm of the square-root of the diagonal value. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdIdent` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric value, it is assumed to represent the unrestricted coefficient of the underlying positive-definite matrix.

```
pdIdent(value, form, nam, data)
```

ARGUMENTS

- `value`: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric value. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- `nam`: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

a `pdIdent` object representing a multiple of the identity positive-definite matrix, also inheriting from class `pdMat`.

SEE ALSO

`as.matrix.pdMat`, `coef.pdMat`, `matrix<-`.`pdMat`

EXAMPLE

```
pd1 <- pdIdent(4 * diag(3), nam = c("A", "B", "C"))
pd1
```

This function is a constructor for the `pdKron` class, representing a positive-definite matrix obtained as the Kronecker product between an identity matrix and a basic positive-definite matrix. Any `pdMat` object can be used to define the basic `pd` matrix. The order of the identity matrix used in the Kronecker product is determined by the number of levels of the factor declared in `groups`. The number of coefficients associated with this class is equal to the number of coefficients used to represent the basic `pdMat` object. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdKron` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, the basic `pdMat` object will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the unrestricted coefficients of the basic `pdMat` object.

```
pdKron(value, form, nam, data, pdClass, groups)
```

ARGUMENTS

- `value`: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional one-sided linear formula specifying the row/column names for the basic matrix used to form the Kronecker-product matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- `nam`: an optional vector of character strings specifying the row/column names for the basic matrix used to form the Kronecker-product matrix represented by `object`. It must have length equal to the dimension of the underlying basic positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value`, `form`, and `groups`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.
- `pdClass`: an optional character string naming the `pdMat` class to be assigned to the basic element used to form the Kronecker product represented by the returned object. This argument will only be used when `value` is not a `pdMat` object. Defaults to `"pdSymm"`.

groups: a one-sided formula specifying the grouping variable used in forming the Kronecker-product matrix represented by `object`. Its right hand side must evaluate to a factor in `data`; the number of levels of the corresponding factor determines the number of identical block diagonal elements in the matrix represented by `object`.

VALUE

a `pdKron` object representing a positive-definite matrix obtained as the Kronecker product between an identity matrix and a basic positive-definite matrix, also inheriting from class `pdMat`.

SEE ALSO

`matrix<-.pdKron`

EXAMPLE

```
pd1 <- pdKron(3*diag(2)+1, form = ~ day, data = Pixel, groups = ~ S
pd1
```

pdMat	Positive-Definite Matrix	pdMat
--------------	--------------------------	--------------

This function gives an alternative way of constructing an object inheriting from the `pdMat` class named in `pdClass`, or from `data.class(object)` if `object` inherits from `pdMat`, and is mostly used internally in other functions. See the documentation on the principal constructor function, generally with the same name as the `pdMat` class of `object`.

```
pdMat(value, form, nam, data, pdClass)
```

ARGUMENTS

- value:** an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form:** an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- nam:** an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- data:** an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is

made to obtain information on `factors` appearing the formulas. Defaults to parent frame from which the function was called.

`pdClass`: an optional character string naming the `pdMat` class to be assigned to the returned object. This argument will only be used when `value` is not a `pdMat` object. Defaults to `"pdSymm"`.

VALUE

a `pdMat` object representing a positive-definite matrix, inheriting from the class named in `pdClass`, or from `class(object)`, if `object` inherits from `pdMat`.

SEE ALSO

`pdCompSymm`, `pdDiag`, `pdIdent`, `pdSymm`

EXAMPLE

```
pd1 <- pdMat(diag(1:4), pdClass = "pdDiag")
pd1
```

pdMatrix	pdMat Matrix or Square-Root Factor	pdMatrix
-----------------	------------------------------------	-----------------

The positive-definite matrix represented by `object`, or a square-root factor of it is obtained. Letting Σ denote a positive-definite matrix, a square-root factor of Σ is any square matrix L such that $\Sigma = L^t L$. This function extracts Σ or L .

```
pdMatrix(object, fact, ...)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive definite matrix.

`fact`: an optional logical value. If `TRUE`, a square-root factor of the positive-definite matrix represented by `object` is returned; else, if `FALSE`, the positive-definite matrix is returned. Defaults to `FALSE`.

`...`: some methods for the generic function may require additional arguments.

VALUE

if `fact` is `FALSE` the positive-definite matrix represented by `object` is returned; else a square-root of the positive-definite matrix is returned.

SEE ALSO

`as.matrix.pdMat`, `pdFactor`, `corMatrix`

EXAMPLE

```
pd1 <- pdSymm(diag(1:4))
pdMatrix(pd1)
```

pdMatrix.reStruct	reStruct Matrix or Square-Root Factor	pdMatrix.reStruct
--------------------------	---------------------------------------	--------------------------

This method function extracts the positive-definite matrices corresponding to the `pdMat` elements of `object`, or square-root factors of the positive-definite matrices.

```
pdMatrix(object, fact)
```

ARGUMENTS

object: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

fact: an optional logical value. If `TRUE`, square-root factors of the positive-definite matrices represented by the elements of `object` are returned; else, if `FALSE`, the positive-definite matrices are returned. Defaults to `FALSE`.

VALUE

a list with components given by the positive-definite matrices corresponding to the elements of `object`, or square-root factors of the positive-definite matrices.

SEE ALSO

`as.matrix.reStruct`, `reStruct`, `pdMatrix.pdMat`

EXAMPLE

```
rsl <- reStruct(pdSymm(diag(3), form=~ Sex+age, data=Orthodont))
pdMatrix(rsl)
```

pdStrat	Stratified Positive-Definite Matrix	pdStrat
----------------	-------------------------------------	----------------

This function is a constructor for the `pdStrat` class, representing a set of positive-definite matrices corresponding to different strata, defined by `strata`. Different `pdMat` classes can be used for different strata, but they must all represent matrices of equal dimensions and equal row/column names. The number of coefficients associated with this class is equal to the sum of the number of coefficients of each `pdMat` object it includes. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdStrat` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions.

```
pdStrat(value, form, nam, data, strata)
```

ARGUMENTS

- value:** an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, a numeric vector, or a list of `pdMat` objects, positive-definite matrices, or numeric vectors. If given as a list, all of its elements must represent positive-definite matrices of equal dimensions and equal row/column names, when converted to `pdMat` objects (in particular, all `pdMat` objects must have the same formula). When not given as a list, `value` is repeated for all strata. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form:** an optional one-sided linear formula specifying the row/column names for all `pdMat` objects represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- nam:** an optional vector of character strings specifying the row/column names for the matrices represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrices and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- data:** an optional data frame in which to evaluate the variables named in `value`, `form`, and `strata`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.
- strata:** a one-sided formula specifying the stratification variable for the different `pdMat` objects represented by `object`. Its right hand side must evaluate to a factor or an integer in `data`.

VALUE

a `pdStrat` object representing a set of positive-definite matrices corresponding to different strata.

SEE ALSO

`pdMat`

EXAMPLE

```
pd1 <- pdStrat(list(pdSymm(diag(2)), pdDiag(c(1,0))), form = ~ age,
  data = Orthodont, strata = ~ Sex)
pd1
```

This function is a constructor for the `pdSymm` class, representing a general positive-definite matrix. If the matrix associated with `object` is of order n , it is represented by $n(n + 1)/2$ unrestricted parameters, using the matrix-logarithm parametrization described in Pinheiro and Bates (1996). When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdSymm` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the unrestricted coefficients of the matrix-logarithm parametrization of the underlying positive-definite matrix.

```
pdSymm(value, form, nam, data)
```

ARGUMENTS

- `value`: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- `form`: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- `nam`: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.
- `data`: an optional data frame in which to evaluate the variables named in `value` and `form`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying matrix. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

a `pdSymm` object representing a general positive-definite matrix, also inheriting from class `pdMat`.

REFERENCES

Pinheiro, J.C. and Bates., D.M. (1996) "Unconstrained Parametrizations for Variance-Covariance Matrices", *Statistics and Computing*, 6, 289-296.

SEE ALSO

```
as.matrix.pdMat, coef.pdMat, matrix<- .pdMat
```

EXAMPLE

```
pd1 <- pdSymm(diag(1:3), nam = c("A", "B", "C"))
pd1
```

pdSymmNat	General PD Matrix in Natural Parametrization	pdSymmNat
------------------	--	------------------

This function is a constructor for the `pdSymmNat` class, representing a general positive-definite matrix, using a natural parametrization. If the matrix associated with `object` is of order n , it is represented by $n(n + 1)/2$ parameters. Letting Σ_{ij} denote the ij -th element of the underlying positive definite matrix and $\rho_{ij} = \Sigma_{ij} / \sqrt{\Sigma_{ii}\Sigma_{jj}}$, $i \neq j$ denote the associated *correlations*, if `unconstrained = TRUE`, the *natural* parameters are given by $\log(\Sigma_{ii})/2$, $i = 1, \dots, n$ and $\log((1 + \rho_{ij})/(1 - \rho_{ij}))$, $i \neq j$. Note that, in this case, all natural parameters are individually unrestricted, but not jointly unrestricted (meaning that not all unrestricted vectors would give positive-definite matrices). If `unconstrained = FALSE`, the Σ_{ij} are used to parameterize the returned object. This class does not define an unrestricted parameterization and, therefore, should NOT be used for optimization. It is mostly used for deriving approximate confidence intervals on parameters following the optimization of an objective function. When `value` is `numeric(0)`, an uninitialized `pdMat` object, a one-sided formula, or a vector of character strings, `object` is returned as an uninitialized `pdSymm` object (with just some of its attributes and its class defined) and needs to have its coefficients assigned later, generally using the `coef` or `matrix` replacement functions. If `value` is an initialized `pdMat` object, `object` will be constructed from `as.matrix(value)`. Finally, if `value` is a numeric vector, it is assumed to represent the natural parameters of the underlying positive-definite matrix.

```
pdSymmNat(value, form, nam, data, unconstrained)
```

ARGUMENTS

- value**: an optional initialization value, which can be any of the following: a `pdMat` object, a positive-definite matrix, a one-sided linear formula (with variables separated by `+`), a vector of character strings, or a numeric vector. Defaults to `numeric(0)`, corresponding to an uninitialized object.
- form**: an optional one-sided linear formula specifying the row/column names for the matrix represented by `object`. Because factors may be present in `form`, the formula needs to be evaluated on a `data.frame` to resolve the names it defines. This argument is ignored when `value` is a one-sided formula. Defaults to `NULL`.
- nam**: an optional vector of character strings specifying the row/column names for the matrix represented by `object`. It must have length equal to the dimension of the underlying positive-definite matrix and unreplicated elements. This argument is ignored when `value` is a vector of character strings. Defaults to `NULL`.

data: an optional data frame in which to evaluate the variables named in **value** and **form**. It is used to obtain the levels for **factors**, which affect the dimensions and the row/column names of the underlying matrix. If **NULL**, no attempt is made to obtain information on **factors** appearing in the formulas. Defaults to the parent frame from which the function was called.

unconstrained: an optional logical value indicating whether an unconstrained natural parameterization should be used. Defaults to **TRUE**.

VALUE

a **pdSymmNat** object representing a general positive-definite matrix in natural parametrization, also inheriting from class **pdMat**.

SEE ALSO

`as.matrix.pdMat`, `coef.pdMat`, `matrix<-.pdMat`

EXAMPLE

```
pdSymmNat(diag(1:3))
```

plot.ACF	Plot an ACF Object	plot.ACF
-----------------	--------------------	-----------------

an **xyplot** of the autocorrelations versus the lags, with **type = "h"**, is produced. If **alpha > 0**, curves representing the critical limits for a two-sided test of level **alpha** for the autocorrelations are added to the plot.

```
plot(object, alpha, xlab, ylab, grid, ...)
```

ARGUMENTS

object: an object inheriting from class **ACF**, consisting of a data frame with two columns named **lag** and **ACF**, representing the autocorrelation values and the corresponding lags.

alpha: an optional numeric value with the significance level for testing if the autocorrelations are zero. Lines corresponding to the lower and upper critical values for a test of level **alpha** are added to the plot. Default is 0, in which case no lines are plotted.

xlab,ylab: optional character strings with the x- and y-axis labels. Default respectively to "Lag" and "Autocorrelation".

grid: an optional logical value indicating whether a grid should be added to plot. Defaults to **FALSE**.

...: optional arguments passed to the **xyplot** function.

VALUE

an **xyplot** Trellis plot.

SEE ALSO

ACF, **xyplot**

EXAMPLE

```
fml <- lme(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary)
plot(ACF(fml, maxLag = 10), alpha = 0.01)
```

plot.augPred

Plot an augPred Object

plot.augPred

A Trellis `xyplot` of predictions versus the primary covariate is generated, with a different panel for each value of the grouping factor. Predicted values are joined by lines, with different line types (colors) being used for each level of grouping. Original observations are represented by circles.

```
plot(x, key, grid, ...)
```

ARGUMENTS

x: an object of class `augPred`.

key: an optional logical value, or list. If `TRUE`, a legend is included at the top of the plot indicating which symbols (colors) correspond to which prediction levels. If `FALSE`, no legend is included. If given as a list, `key` is passed down as an argument to the `trellis` function generating the plots (`xyplot`). Defaults to `TRUE`.

grid: an optional logical value indicating whether a grid should be added to plot. Defaults to `FALSE`.

...: optional arguments passed down to the `trellis` function generating the plots.

VALUE

A Trellis plot of predictions versus the primary covariate, with panels determined by the grouping factor.

SEE ALSO

`augPred`, `xyplot`

EXAMPLE

```
fml <- lme(Orthodont)
plot(augPred(fml, level = 0:1, length.out = 2))
```


A Trellis `dotplot` of the values being compared, with different rows per group, is generated, with a different panel for each coefficient. Different symbols (colors) are used for each object being compared.

```
plot(object, subset, key, mark, ...)
```

ARGUMENTS

object: an object of class `compareFits`.

subset: an optional logical or integer vector specifying which rows of `object` should be used in the plots. If missing, all rows are used.

key: an optional logical value, or list. If `TRUE`, a legend is included at the top of the plot indicating which symbols (colors) correspond to which objects being compared. If `FALSE`, no legend is included. If given as a list, `key` is passed down as an argument to the `trellis` function generating the plots (`dotplot`). Defaults to `TRUE`.

mark: an optional numeric vector, of length equal to the number of coefficients being compared, indicating where vertical lines should be drawn in the plots. If missing, no lines are drawn.

...: optional arguments passed down to the `trellis` function generating the plots.

VALUE

A Trellis `dotplot` of the values being compared, with rows determined by the groups and panels by the coefficients.

SEE ALSO

```
compareFits, pairs.compareFits, dotplot
```

EXAMPLE

```
fm1 <- lmList(Orthodont)
fm2 <- lme(Orthodont)
plot(compareFits(coef(fm1), coef(fm2)))
```

Diagnostic plots for the linear model fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display. If `form` is a one-sided formula, histograms of the variable on the right hand side of the formula, before a `|` operator, are displayed (the Trellis function `histogram` is used). If `form` is two-sided and both its left and right hand side variables are numeric, scatter plots are displayed (the Trellis function `xyplot` is used). Finally, if `form` is two-sided and its left hand side variable is a factor, box-plots of the right hand side variable by the levels of the left hand side variable are displayed (the Trellis function `bwplot` is used).

```
plot(object, form, abline, id, idLabels, idResType, grid, subset, ...)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

form: an optional formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `". "`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. Default is `resid(., type = "p") ~fitted(.,` corresponding to a plot of the standardized residuals versus fitted values, both evaluated at the innermost level of nesting.

abline: an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.

id: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized residuals. Observations with absolute standardized residuals greater than the $1-id/2$ quantile of the standard normal distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.

idLabels: an optional vector, or one-sided formula. If given as a vector, it is converted to character mode and used to label the observations identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character mode and used to label the identified observations. Default is the innermost grouping factor.

`idResType`: an optional character string specifying the type of residuals to be used in identifying outliers, when `id` is a numeric value. If "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "pearson". `.AG` subset an optional expression indicating the subset of the observations that should be used in the plot. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

`grid`: an optional logical value indicating whether a grid should be added to plot. Default depends on the type of Trellis plot used: if `xyplot` defaults to TRUE, else defaults to FALSE.

...: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot.

SEE ALSO

`gls`, `xyplot`, `bwplot`, `histogram`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          correlation = corAR1(form = ~ 1 | Mare))
# standardized residuals versus fitted values by Mare
plot(fml, resid(., type = "p") ~ fitted(.) | Mare, abline = 0)
# box-plots of residuals by Mare
plot(fml, Mare ~ resid(.))
# observed versus fitted values by Mare
plot(fml, follicles ~ fitted(.) | Mare, abline = c(0,1))
```

A Trellis dot-plot of the confidence intervals on the linear model coefficients is generated, with a different panel for each coefficient. Rows in the dot-plot correspond to the names of the `lm` components of the `lmList` object used to produce `object`. The lower and upper confidence limits are connected by a line segment and the estimated coefficients are marked with a "+". The Trellis function `dotplot` is used in this method function.

```
plot(object, ...)
```

ARGUMENTS

`object`: an object inheriting from class `intervals.lmList`, representing confidence intervals and estimates for the coefficients in the `lm` components of the `lmList` object used to produce `object`.

`...`: optional arguments passed to the Trellis `dotplot` function.

VALUE

a Trellis plot with the confidence intervals on the coefficients of the individual `lm` components of the `lmList` that generated `object`.

SEE ALSO

`intervals.lmList`, `lmList`, `dotplot`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
plot(intervals(fml))
```

Diagnostic plots for the linear model fits corresponding to the `object` components are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display. If `form` is a one-sided formula, histograms of the variable on the right hand side of the formula, before a `|` operator, are displayed (the Trellis function `histogram` is used). If `form` is two-sided and both its left and right hand side variables are numeric, scatter plots are displayed (the Trellis function `xypplot` is used). Finally, if `form` is two-sided and its left hand side variable is a factor, box-plots of the right hand side variable by the levels of the left hand side variable are displayed (the Trellis function `bwplot` is used).

```
plot(object, form, abline, id, idLabels, grid, subset, ...)
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

form: an optional formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `". "`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. Default is `resid(., type = "pool") ~fitted(.,` corresponding to a plot of the standardized residuals (using a pooled estimate for the residual standard error) versus fitted values.

abline: an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.

id: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized residuals. Observations with absolute standardized residuals greater than the $1 - \text{value}/2$ quantile of the standard normal distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.

idLabels: an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the observations identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified observations. Default is `getGroups(object)`. `.AG` subset an optional expression indicating the subset

of the observations that should be used in the plot. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.

grid: an optional logical value indicating whether a grid should be added to plot. Default depends on the type of Trellis plot used: if `xyplot` defaults to `TRUE`, else defaults to `FALSE`.

...: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot.

SEE ALSO

`lmList`, `xyplot`, `bwplot`, `histogram`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
# standardized residuals versus fitted values by gender
plot(fml, resid(., type = "pool") ~ fitted(.) | Sex,
      abline = 0, id = 0.05)
# box-plots of residuals by Subject
plot(fml, Subject ~ resid(.))
# observed versus fitted values by Subject
plot(fml, distance ~ fitted(.) | Subject, abline = c(0,1))
```

plot.lme

Plot an lme Object

plot.lme

Diagnostic plots for the linear mixed-effects fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display. If `form` is a one-sided formula, histograms of the variable on the right hand side of the formula, before a `|` operator, are displayed (the Trellis function `histogram` is used). If `form` is two-sided and both its left and right hand side variables are numeric, scatter plots are displayed (the Trellis function `xyplot` is used). Finally, if `form` is two-sided and its left hand side variable is a factor, box-plots of the right hand side variable by the levels of the left hand side variable are displayed (the Trellis function `bwplot` is used).

```
plot(object, form, abline, id, idLabels, idResType, grid, sub-
set, ...)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

- form:** an optional formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `"."`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. Default is `resid(., type = "p") ~fitted(.)`, corresponding to a plot of the standardized residuals versus fitted values, both evaluated at the innermost level of nesting.
- abline:** an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.
- id:** an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized residuals. Observations with absolute standardized residuals greater than the $1-id/2$ quantile of the standard normal distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.
- idLabels:** an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the observations identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified observations. Default is the innermost grouping factor.
- idResType:** an optional character string specifying the type of residuals to be used in identifying outliers, when `id` is a numeric value. If `"pearson"`, the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if `"normalized"`, the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to `"pearson"`.
- grid:** an optional logical value indicating whether a grid should be added to plot. Default depends on the type of Trellis plot used: if `xyplot` defaults to `TRUE`, else defaults to `FALSE`.
- ...:** optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot.

SEE ALSO

`lme`, `xyplot`, `bwplot`, `histogram`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
# standardized residuals versus fitted values by gender
plot(fml, resid(., type = "p") ~ fitted(.) | Sex, abline = 0)
# box-plots of residuals by Subject
plot(fml, Subject ~ resid(.))
# observed versus fitted values by Subject
plot(fml, distance ~ fitted(.) | Subject, abline = c(0,1))
```

plot.nffGroupedData

Plot nffGroupedData Object

plot.nffGroupedData

A Trellis dot-plot of the response by group is generated. If outer variables are specified, the combination of their levels are used to determine the panels of the Trellis display. The Trellis function `dotplot` is used.

```
plot(x, outer, inner, innerGroups, xlab, ylab, strip, panel,
      subset, grid, key, ...)
```

ARGUMENTS

- x**: an object inheriting from class `nffGroupedData`, representing a `groupedData` object with a factor primary covariate and a single grouping level.
- outer**: an optional logical value or one-sided formula, indicating covariates that are outer to the grouping factor, which are used to determine the panels of the Trellis plot. If equal to `TRUE`, `attr(object, "outer")` is used to indicate the outer covariates. An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. Defaults to `NULL`, meaning that no outer covariates are to be used.
- inner**: an optional logical value or one-sided formula, indicating a covariate that is inner to the grouping factor, which is used to associate points within each panel of the Trellis plot. If equal to `TRUE`, `attr(object, "inner")` is used to indicate the inner covariate. An inner covariate can change within the sets of rows defined by the grouping factor. Defaults to `NULL`, meaning that no inner covariate is present.
- innerGroups**: an optional one-sided formula specifying a factor to be used for grouping the levels of the inner covariate. Different colors, or symbols, are used for each level of the `innerGroups` factor. Default is `NULL`, meaning that no `innerGroups` covariate is present.
- xlab**: an optional character string with the label for the horizontal axis. Default is the `y` elements of `attr(object, "labels")` and `attr(object, "units")` pasted together.
- ylab**: an optional character string with the label for the vertical axis. Default is the grouping factor name.

- `strip`: an optional function passed as the `strip` argument to the `dotplot` function. Default is `strip.default(..., style = 1)` (see `trellis.args`).
- `panel`: an optional function used to generate the individual panels in the Trellis display, passed as the `panel` argument to the `dotplot` function. `.AG` subset an optional expression indicating the subset of the observations that should be used in the plot. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
- `key`: an optional logical function or function. If `TRUE` and either `inner` or `innerGroups` are non-NULL, a legend for the different inner (`innerGroups`) levels is included at the top of the plot. If given as a function, it is passed as the `key` argument to the `dotplot` function. Default is `TRUE` if either `inner` or `innerGroups` are non-NULL and `FALSE` otherwise.
- `grid`: this argument is included for consistency with the `plot.nfnGroupedData` method calling sequence. It is ignored in this method function.
- `...`: optional arguments passed to the `dotplot` function.

VALUE

a Trellis dot-plot of the response by group.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1997), "Software Design for Longitudinal Data", in "Modeling Longitudinal and Spatially Correlated Data: Methods, Applications and Future Directions", T.G. Gregoire (ed.), Springer-Verlag, New York.
 Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at <http://nlme.stat.wisc.edu>.

SEE ALSO

`groupedData`, `dotplot`

EXAMPLE

```
plot(Machines)
plot(Machines, inner = TRUE)
```

A Trellis plot of the response versus the primary covariate is generated. If outer variables are specified, the combination of their levels are used to determine the panels of the Trellis display. Otherwise, the levels of the grouping variable determine the panels. A scatter plot of the response versus the primary covariate is displayed in each panel, with observations corresponding to same inner group joined by line segments. The Trellis function `xyplot` is used.

```
plot(x, outer, inner, innerGroups, xlab, ylab, strip, aspect,
      panel, subset, key, grid, ...)
```

ARGUMENTS

- x**: an object inheriting from class `nfnGroupedData`, representing a `groupedData` object with a numeric primary covariate and a single grouping level.
- outer**: an optional logical value or one-sided formula, indicating covariates that are outer to the grouping factor, which are used to determine the panels of the Trellis plot. If equal to `TRUE`, `attr(object, "outer")` is used to indicate the outer covariates. An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. Defaults to `NULL`, meaning that no outer covariates are to be used.
- inner**: an optional logical value or one-sided formula, indicating a covariate that is inner to the grouping factor, which is used to associate points within each panel of the Trellis plot. If equal to `TRUE`, `attr(object, "inner")` is used to indicate the inner covariate. An inner covariate can change within the sets of rows defined by the grouping factor. Defaults to `NULL`, meaning that no inner covariate is present.
- innerGroups**: an optional one-sided formula specifying a factor to be used for grouping the levels of the `inner` covariate. Different colors, or line types, are used for each level of the `innerGroups` factor. Default is `NULL`, meaning that no `innerGroups` covariate is present.
- xlab, ylab**: optional character strings with the labels for the plot. Default is the corresponding elements of `attr(object, "labels")` and `attr(object, "units")` pasted together.
- strip**: an optional function passed as the `strip` argument to the `xyplot` function. Default is `strip.default(..., style = 1)` (see `trellis.args`).
- aspect**: an optional character string indicating the aspect ratio for the plot passed as the `aspect` argument to the `xyplot` function. Default is `"xy"` (see `trellis.args`).

- `panel`: an optional function used to generate the individual panels in the Trellis display, passed as the `panel` argument to the `xyplot` function. `.AG` subset an optional expression indicating the subset of the observations that should be used in the plot. This can be a logical vector, or a numeric vector indicating which observation numbers are to be included, or a character vector of the row names to be included. All observations are included by default.
- `key`: an optional logical function or function. If `TRUE` and `innerGroups` is non-`NULL`, a legend for the different `innerGroups` levels is included at the top of the plot. If given as a function, it is passed as the `key` argument to the `xyplot` function. Default is `TRUE` if `innerGroups` is non-`NULL` and `FALSE` otherwise.
- `grid`: an optional logical value indicating whether a grid should be added to plot. Defaults to `TRUE`.
- `...`: optional arguments passed to the `xyplot` function.

VALUE

a Trellis plot of the response versus the primary covariate.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1997), "Software Design for Longitudinal Data", in "Modeling Longitudinal and Spatially Correlated Data: Methods, Applications and Future Directions", T.G. Gregoire (ed.), Springer-Verlag, New York.
 Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at <http://nlme.stat.wisc.edu>.

SEE ALSO

`groupedData`, `xyplot`

EXAMPLE

```
# different panels per Subject
plot(Orthodont)
# different panels per gender
plot(Orthodont, outer = TRUE)
```

The `groupedData` object is summarized by the values of the `displayLevel` grouping factor (or the combination of its values and the values of the covariate indicated in `preserve`, if any is present). The collapsed data is used to produce a new `groupedData` object, with grouping factor given by the `displayLevel` factor, which is plotted using the appropriate `plot` method for `groupedData` objects with single level of grouping.

```
plot(x, collapseLevel, displayLevel, outer, inner, preserve,  
     FUN, subset, grid, ...)
```

ARGUMENTS

- `x`: an object inheriting from class `nmGroupedData`, representing a `groupedData` object with multiple grouping factors.
- `collapseLevel`: an optional positive integer or character string indicating the grouping level to use when collapsing the data. Level values increase from outermost to innermost grouping. Default is the highest or innermost level of grouping.
- `displayLevel`: an optional positive integer or character string indicating the grouping level to use for determining the panels in the Trellis display, when `outer` is missing. Default is `collapseLevel`.
- `outer`: an optional logical value or one-sided formula, indicating covariates that are outer to the `displayLevel` grouping factor, which are used to determine the panels of the Trellis plot. If equal to `TRUE`, the `displayLevel` element `attr(object, "outer")` is used to indicate the outer covariates. An outer covariate is invariant within the sets of rows defined by the grouping factor. Ordering of the groups is done in such a way as to preserve adjacency of groups with the same value of the outer variables. Defaults to `NULL`, meaning that no outer covariates are to be used.
- `inner`: an optional logical value or one-sided formula, indicating a covariate that is inner to the `displayLevel` grouping factor, which is used to associate points within each panel of the Trellis plot. If equal to `TRUE`, `attr(object, "inner")` is used to indicate the inner covariate. An inner covariate can change within the sets of rows defined by the grouping factor. Defaults to `NULL`, meaning that no inner covariate is present.
- `preserve`: an optional one-sided formula indicating a covariate whose levels should be preserved when collapsing the data according to the `collapseLevel` grouping factor. The collapsing factor is obtained by pasting together the levels of the `collapseLevel` grouping factor and the values of the covariate to be preserved. Default is `NULL`, meaning that no covariates need to be preserved.

- FUN**: an optional summary function or a list of summary functions to be used for collapsing the data. The function or functions are applied only to variables in object that vary within the groups defined by `collapseLevel`. Invariant variables are always summarized by group using the unique value that they assume within that group. If **FUN** is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If **FUN** is a list of functions, the names in the list should designate classes of variables in the data such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any non-invariant variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.
- subset**: an optional named list. Names can be either positive integers representing grouping levels, or names of grouping factors. Each element in the list is a vector indicating the levels of the corresponding grouping factor to be used for plotting the data. Default is `NULL`, meaning that all levels are used.
- grid**: an optional logical value indicating whether a grid should be added to plot. Defaults to `TRUE`.
- ...**: optional arguments passed to the Trellis plot function.

VALUE

a Trellis display of the data collapsed over the values of the `collapseLevel` grouping factor and grouped according to the `displayLevel` grouping factor.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1997), "Software Design for Longitudinal Data", in "Modeling Longitudinal and Spatially Correlated Data: Methods, Applications and Future Directions", T.G. Gregoire (ed.), Springer-Verlag, New York.
 Pinheiro, J.C. and Bates, D.M. (1997) "Future Directions in Mixed-Effects Software: Design of NLME 3.0" available at <http://nlme.stat.wisc.edu>.

SEE ALSO

`groupedData`, `collapse.groupedData`, `plot.nfnGroupedData`, `plot.nffGroupedData`

EXAMPLE

```
# no collapsing, panels by Dog
plot(Pixel, display = "Dog", inner = ~ Side)
# collapsing by Dog, preserving day
plot(Pixel, collapse = "Dog", preserve = ~ day)
```

If `form` is missing, or is given as a one-sided formula, a Trellis dot-plot of the random effects is generated, with a different panel for each random effect (coefficient). Rows in the dot-plot are determined by the `form` argument (if not missing) or by the row names of the random effects (coefficients). If a single factor is specified in `form`, its levels determine the dot-plot rows (with possibly multiple dots per row); otherwise, if `form` specifies a crossing of factors, the dot-plot rows are determined by all combinations of the levels of the individual factors in the formula. The Trellis function `dotplot` is used in this method function.

If `form` is a two-sided formula, a Trellis display is generated, with a different panel for each variable listed in the right hand side of `form`. Scatter plots are generated for numeric variables and boxplots are generated for categorical (factor or ordered) variables.

```
plot(object, form, omitFixed, level, grid, control, ...)
```

ARGUMENTS

object: an object inheriting from class `ranef.lme`, representing the estimated coefficients or estimated random effects for the `lme` object from which it was produced.

form: an optional formula specifying the desired type of plot. If given as a one-sided formula, a `dotplot` of the estimated random effects (coefficients) grouped according to all combinations of the levels of the factors named in `form` is returned. Single factors (`~g`) or crossed factors (`~g1*g2`) are allowed. If given as a two-sided formula, the left hand side must be a single random effects (coefficient) and the right hand side is formed by covariates in `object` separated by `+`. A Trellis display of the random effect (coefficient) versus the named covariates is returned in this case. Default is `NULL`, in which case the row names of the random effects (coefficients) are used.

omitFixed: an optional logical value indicating whether columns with values that are constant across groups should be omitted. Default is `TRUE`.

level: an optional integer value giving the level of grouping to be used for `object`. Only used when `object` is a list with different components for each grouping level. Defaults to the highest or innermost level of grouping.

grid: an optional logical value indicating whether a grid should be added to plot. Only applies to plots associated with two-sided formulas in `form`. Default is `FALSE`.

control: an optional list with control values for the plot, when `form` is given as a two-sided formula. The control values are referenced by name in the `control` list and only the ones to be modified from the default need to be specified. Available

values include: `drawLine`, a logical value indicating whether a loess smoother should be added to the scatter plots and a line connecting the medians should be added to the boxplots (default is `TRUE`); `span.loess`, used as the `span` argument in the call to `panel.loess` (default is `2/3`); `degree.loess`, used as the `degree` argument in the call to `panel.loess` (default is `1`); `cex.axis`, the character expansion factor for the x-axis (default is `0.8`); `srt.axis`, the rotation factor for the x-axis (default is `0`); and `mgp.axis`, the margin parameters for the x-axis (default is `c(2, 0.5, 0)`).

...: optional arguments passed to the Trellis `dotplot` function.

VALUE

a Trellis plot of the estimated random-effects (coefficients) versus covariates, or groups.

SEE ALSO

`ranef.lme`, `lme`, `dotplot`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
plot(ranef(fml))
fmlRE <- ranef(fml, aug = TRUE)
plot(fmlRE, form = ~ Sex)
plot(fmlRE, form = age ~ Sex)
```

plot.ranef.lmList	Plot a ranef.lmList Object	plot.ranef.lmList
--------------------------	----------------------------	--------------------------

If `form` is missing, or is given as a one-sided formula, a Trellis dot-plot of the random effects is generated, with a different panel for each random effect (coefficient). Rows in the dot-plot are determined by the `form` argument (if not missing) or by the row names of the random effects (coefficients). If a single factor is specified in `form`, its levels determine the dot-plot rows (with possibly multiple dots per row); otherwise, if `form` specifies a crossing of factors, the dot-plot rows are determined by all combinations of the levels of the individual factors in the formula. The Trellis function `dotplot` is used in this method function.

If `form` is a two-sided formula, a Trellis display is generated, with a different panel for each variable listed in the right hand side of `form`. Scatter plots are generated for numeric variables and boxplots are generated for categorical (factor or ordered) variables.

```
plot(object, form, grid, control, ...)
```

ARGUMENTS

object: an object inheriting from class `ranef.lmList`, representing the estimated coefficients or estimated random effects for the `lmList` object from which it was produced.

- form:** an optional formula specifying the desired type of plot. If given as a one-sided formula, a `dotplot` of the estimated random effects (coefficients) grouped according to all combinations of the levels of the factors named in `form` is returned. Single factors (`~g`) or crossed factors (`~g1*g2`) are allowed. If given as a two-sided formula, the left hand side must be a single random effects (coefficient) and the right hand side is formed by covariates in `object` separated by `+`. A Trellis display of the random effect (coefficient) versus the named covariates is returned in this case. Default is `NULL`, in which case the row names of the random effects (coefficients) are used.
- grid:** an optional logical value indicating whether a grid should be added to plot. Only applies to plots associated with two-sided formulas in `form`. Default is `FALSE`.
- control:** an optional list with control values for the plot, when `form` is given as a two-sided formula. The control values are referenced by name in the `control` list and only the ones to be modified from the default need to be specified. Available values include: `drawLine`, a logical value indicating whether a loess smoother should be added to the scatter plots and a line connecting the medians should be added to the boxplots (default is `TRUE`); `span.loess`, used as the `span` argument in the call to `panel.loess` (default is `2/3`); `degree.loess`, used as the `degree` argument in the call to `panel.loess` (default is `1`); `cex.axis`, the character expansion factor for the x-axis (default is `0.8`); `srt.axis`, the rotation factor for the x-axis (default is `0`); and `mgp.axis`, the margin parameters for the x-axis (default is `c(2, 0.5, 0)`).
- ...:** optional arguments passed to the Trellis `dotplot` function.

VALUE

a Trellis plot of the estimated random-effects (coefficients) versus covariates, or groups.

SEE ALSO

`lmList`, `dotplot`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
plot(ranef(fml))
fmlRE <- ranef(fml, aug = TRUE)
plot(fmlRE, form = ~ Sex)
plot(fmlRE, form = age ~ Sex)
```


plot.Variogram

Plot a Variogram Object

plot.Variogram

an `xyplot` of the semi-variogram versus the distances is produced. If `smooth = TRUE`, a loess smoother is added to the plot. If `showModel = TRUE` and object includes an "modelVariog" attribute, the corresponding semi-variogram is added to the plot.

```
plot(object, smooth, showModel, sigma, span, xlab, ylab, type, ylim,
```

ARGUMENTS

object: an object inheriting from class `Variogram`, consisting of a data frame with two columns named `variog` and `dist`, representing the semi-variogram values and the corresponding distances.

smooth: an optional logical value controlling whether a loess smoother should be added to the plot. Defaults to `TRUE`, when `showModel` is `FALSE`.

showModel: an optional logical value controlling whether the semi-variogram corresponding to an "modelVariog" attribute of `object`, if any is present, should be added to the plot. Defaults to `TRUE`, when the "modelVariog" attribute is present.

sigma: an optional numeric value used as the height of a horizontal line displayed in the plot. Can be used to represent the process standard deviation. Default is `NULL`, implying that no horizontal line is drawn.

span: an optional numeric value with the smoothing parameter for the loess fit. Default is 0.6.

xlab,ylab: optional character strings with the x- and y-axis labels. Default respectively to "Distance" and "Semivariogram".

type: an optional character indicating the type of plot. Defaults to "p".

ylim: an optional numeric vector with the limits for the y-axis. Defaults to `c(0, max(object$variog))`.

...: optional arguments passed to the Trellis `xyplot` function.

VALUE

an `xyplot` Trellis plot.

SEE ALSO

`Variogram`, `xyplot`, `loess`

EXAMPLE

```
fml <- lme(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary)
plot(Variogram(fml, form = ~ Time | Mare, maxDist = 0.7))
```

pooledSD

Extract Pooled Standard Deviation

pooledSD

The pooled estimated standard deviation is obtained by adding together the residual sum of squares for each non-null element of `object`, dividing by the sum of the corresponding residual degrees-of-freedom, and taking the square-root.

```
pooledSD(object)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`.

VALUE

the pooled standard deviation for the non-null elements of `object`, with an attribute `df` with the number of degrees-of-freedom used in the estimation.

SEE ALSO

`lmList`, `lm`

EXAMPLE

```
fml <- lmList(Orthodont)
pooledSD(fml)
```

predict.glsPredictions from a `glS` Object**predict.gls**

The predictions for the linear model represented by `object` are obtained at the covariate values defined in `newdata`.

```
predict(object, newdata, na.action)
```

ARGUMENTS

`object`: an object inheriting from class `glS`, representing a generalized least squares fitted linear model.

`newdata`: an optional data frame to be used for obtaining the predictions. All variables used in the linear model must be present in the data frame. If missing, the fitted values are returned.

`na.action`: a function that indicates what should happen when `newdata` contains NAs. The default action (`na.fail`) causes the function to print an error message and terminate if there are any incomplete observations.

VALUE

a vector with the predicted values.

SEE ALSO

`glS`, `fitted.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
newOvary <- data.frame(Time = c(-0.75, -0.5, 0, 0.5, 0.75))
predict(fml, newOvary)
```

predict.gnls	Predictions from a gnls Object	predict.gnls
---------------------	--------------------------------	---------------------

The predictions for the nonlinear model represented by `object` are obtained at the covariate values defined in `newdata`.

```
predict(object, newdata, na.action, naPattern)
```

ARGUMENTS

`object`: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

`newdata`: an optional data frame to be used for obtaining the predictions. All variables used in the nonlinear model must be present in the data frame. If missing, the fitted values are returned.

`na.action`: a function that indicates what should happen when `newdata` contains NAs. The default action (`na.fail`) causes the function to print an error message and terminate if there are any incomplete observations.

`naPattern`: an expression or formula object, specifying which returned values are to be regarded as missing.

VALUE

a vector with the predicted values.

SEE ALSO

`gnls`, `fitted.gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
           weights = varPower())
newSoybean <- data.frame(Time = c(10,30,50,80,100))
predict(fml, newSoybean)
```

If the grouping factor corresponding to `object` is included in `newdata`, the data frame is partitioned according to the grouping factor levels; else, `newdata` is repeated for all `lm` components. The predictions and, optionally, the standard errors for the predictions, are obtained for each `lm` component of `object`, using the corresponding element of the partitioned `newdata`, and arranged into a list with as many components as `object`, or combined into a single vector or data frame (if `se.fit=TRUE`).

```
predict(object, newdata, subset, pool, asList, se.fit)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

`newdata`: an optional data frame to be used for obtaining the predictions. All variables used in the `object` model formula must be present in the data frame. If missing, the same data frame used to produce `object` is used.

`subset`: an optional character or integer vector naming the `lm` components of `object` from which the predictions are to be extracted. Default is `NULL`, in which case all components are used.

`asList`: an optional logical value. If `TRUE`, the returned object is a list with the predictions split by groups; else the returned value is a vector. Defaults to `FALSE`.

`pool`: an optional logical value indicating whether a pooled estimate of the residual standard error should be used. Default is `attr(object, "pool")`.

`se.fit`: an optional logical value indicating whether pointwise standard errors should be computed along with the predictions. Default is `FALSE`.

VALUE

a list with components given by the predictions (and, optionally, the standard errors for the predictions) from each `lm` component of `object`, a vector with the predictions from all `lm` components of `object`, or a data frame with columns given by the predictions and their corresponding standard errors.

SEE ALSO

```
lmList, predict.lm
```

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
predict(fml, se.fit = TRUE)
```

The predictions at level *i* are obtained by adding together the population predictions (based only on the fixed effects estimates) and the estimated contributions of the random effects to the predictions at grouping levels less or equal to *i*. The resulting values estimate the best linear unbiased predictions (BLUPs) at level *i*. If group values not included in the original grouping factors are present in *newdata*, the corresponding predictions will be set to NA for levels greater or equal to the level at which the unknown groups occur.

```
predict(object, newdata, level, asList, na.action)
```

ARGUMENTS

- object**: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.
- newdata**: an optional data frame to be used for obtaining the predictions. All variables used in the fixed and random effects models, as well as the grouping factors, must be present in the data frame. If missing, the fitted values are returned.
- level**: an optional integer vector giving the level(s) of grouping to be used in obtaining the predictions. Level values increase from outermost to innermost grouping, with level zero corresponding to the population predictions. Defaults to the highest or innermost level of grouping.
- asList**: an optional logical value. If `TRUE` and a single value is given in *level*, the returned object is a list with the predictions split by groups; else the returned value is either a vector or a data frame, according to the length of *level*.
- na.action**: a function that indicates what should happen when *newdata* contains NAs. The default action (`na.fail`) causes the function to print an error message and terminate if there are any incomplete observations.

VALUE

if a single level of grouping is specified in *level*, the returned value is either a list with the predictions split by groups (`asList = TRUE`) or a vector with the predictions (`asList = FALSE`); else, when multiple grouping levels are specified in *level*, the returned object is a data frame with columns given by the predictions at different levels and the grouping factors.

SEE ALSO

`lme`, `fitted.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
newOrth <- data.frame(Sex = c("Male", "Male", "Female", "Female", "Male"),
                      age = c(15, 20, 10, 12, 2, 4),
                      Subject = c("M01", "M01", "F30", "F30", "M04", "M04"))
predict(fml, newOrth, level = 0:1)
```

predict.nlme	Predictions from an nlme Object	predict.nlme
---------------------	---------------------------------	---------------------

The predictions at level *i* are obtained by adding together the contributions from the estimated fixed effects and the estimated random effects at levels less or equal to *i* and evaluating the model function at the resulting estimated parameters. If group values not included in the original grouping factors are present in *newdata*, the corresponding predictions will be set to NA for levels greater or equal to the level at which the unknown groups occur.

```
predict(object, newdata, level, asList, na.action, naPattern)
```

ARGUMENTS

object: an object inheriting from class *nlme*, representing a fitted nonlinear mixed-effects model.

newdata: an optional data frame to be used for obtaining the predictions. All variables used in the nonlinear model, the fixed and the random effects models, as well as the grouping factors, must be present in the data frame. If missing, the fitted values are returned.

level: an optional integer vector giving the level(s) of grouping to be used in obtaining the predictions. Level values increase from outermost to innermost grouping, with level zero corresponding to the population predictions. Defaults to the highest or innermost level of grouping.

asList: an optional logical value. If *TRUE* and a single value is given in *level*, the returned object is a list with the predictions split by groups; else the returned value is either a vector or a data frame, according to the length of *level*.

na.action: a function that indicates what should happen when *newdata* contains NAs. The default action (*na.fail*) causes the function to print an error message and terminate if there are any incomplete observations.

naPattern: an expression or formula object, specifying which returned values are to be regarded as missing.

VALUE

if a single level of grouping is specified in *level*, the returned value is either a list with the predictions split by groups (*asList* = *TRUE*) or a vector with the

predictions (`asList = FALSE`); else, when multiple grouping levels are specified in `level`, the returned object is a data frame with columns given by the predictions at different levels and the grouping factors.

SEE ALSO

`nlme`, `fitted.nlme`

EXAMPLE

```
fml <- nlme(weight ~ SSlogis(Time, Asym, xmid, scal),
            data = Soybean, fixed = Asym + xmid + scal ~ 1,
            start = c(18, 52, 7.5))
newSoybean <- data.frame(Time = c(10,30,50,80,100),
                          Plot = c("1988F1", "1988F1", "1988F1", "1988F1", "1988F1"))
predict(fml, newSoybean, level = 0:1)
```

print.anova.lme

Print an anova.lme Object

print.anova.lme

When only one fitted model object is used in the call to `anova.lme`, a data frame with the estimated values, the approximate standard errors, the z-ratios, and the p-values for the fixed effects is printed. Otherwise, when multiple fitted objects are being compared, a data frame with the degrees of freedom, the (restricted) log-likelihood, the Akaike Information Criterion (AIC), and the Bayesian Information Criterion (BIC) of each fitted model object is printed. If included in `x`, likelihood ratio statistics, with associated p-values, are included in the output.

```
print(x, verbose)
```

ARGUMENTS

`x`: an object inheriting from class `anova.lme`, generally obtained by applying the `anova.lme` method to an `lme` object.

`verbose`: an optional logical value. If `TRUE`, the calling sequences for each fitted model object are printed with the rest of the output, being omitted if `verbose = FALSE`. Defaults to `attr(x, "verbose")`.

SEE ALSO

`anova.lme`, `lme`

EXAMPLE

```
fml <- lme(distance ~ age * Sex, Orthodont, random = ~ age | Subject)
print(anova(fml))           # single argument form
fm2 <- update(fml, random = ~ 1 | Subject)
print(anova(fml, fm2))      # multiple argument form
```

print.corStruct

Print a corStruct Object

print.corStruct

If `x` has been initialized, its coefficients are printed in constrained form.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `corStruct`, representing a correlation structure.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

VALUE

the printed coefficients of `x` in constrained form.

SEE ALSO

```
print.default, coef.corStruct
```

EXAMPLE

```
cs1 <- corAR1(0.3)
print(cs1)
```

print.gls

Print a gls Object

print.gls

Information describing the fitted linear model represented by `x` is printed. This includes the coefficients, correlation and variance function parameters, if any are present, and the residual standard error.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

```
gls, print.summary.gls
```

EXAMPLE

```
fm1 <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
print(fm1)
```


print.groupedData

Print a groupedData Object

print.groupedData

Prints the display formula and the data frame associated with `object`.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `groupedData`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`groupedData`

EXAMPLE

```
print(Orthodont)
```

print.intervals.gls

Print an intervals.gls Object

print.intervals.gls

The individual components of `x` are printed.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `intervals.gls`, representing a list of data frames with confidence intervals and estimates for the coefficients in the `gl`s object that produced `x`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`intervals.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
print(intervals(fml))
```

print.intervals.lme

Print an intervals.lme Object

print.intervals.lme

The individual components of `x` are printed.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `intervals.lme`, representing a list of data frames with confidence intervals and estimates for the coefficients in the `lme` object that produced `x`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`intervals.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
print(intervals(fml))
```

print.lmList

Print an lmList Object

print.lmList

Information describing the individual `lm` fits corresponding to `object` is printed. This includes the estimated coefficients and the residual standard error.

```
print(x, pool, ...)
```

ARGUMENTS

`x`: an object inheriting from class `lmList`, representing a list of fitted `lm` objects.

`pool`: an optional logical value indicating whether a pooled estimate of the residual standard error should be used. Default is `attr(object, "pool")`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`lmList`

EXAMPLE

```
fml <- lmList(Orthodont)
print(fml)
```

print.lme

Print an lme Object

print.lme

Information describing the fitted linear mixed-effects model represented by `x` is printed. This includes the fixed effects, the standard deviations and correlations for the random effects, the within-group correlation and variance function parameters, if any are present, and the within-group standard deviation.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

```
lme, print.summary.lme
```

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
print(fml)
```

print.modelStruct

Print a modelStruct Object

print.modelStruct

This method function applies `print` to each element of `object`.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

VALUE

the printed elements of `object`.

SEE ALSO

```
print
```

EXAMPLE

```
lms1 <- lmeStruct(reStruct = reStruct(pdDiag(diag(2), ~ age)),
  corStruct = corAR1(0.3))
print(lms1)
```

print.pdMat

Print a pdMat Object

print.pdMat

Print the standard deviations and correlations (if any) associated the positive-definite matrix represented by `x` (considered as a variance-covariance matrix).

```
print(x, ...)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive definite matrix.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

```
print.summary.pdMat
```

EXAMPLE

```
pd1 <- pdSymm(diag(1:3), nam = c("A", "B", "C"))
print(pd1)
```

print.reStruct

Print an reStruct Object

print.reStruct

Each `pdMat` component of `object` is printed, together with its formula and the associated grouping level.

```
print(x, sigma, reEstimates, verbose=F, ...)
```

ARGUMENTS

`x`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.

`sigma`: an optional numeric value used as a multiplier for the square-root factors of the `pdMat` components (usually the estimated within-group standard deviation from a mixed-effects model). Defaults to 1.

`reEstimates`: an optional list with the random effects estimates for each level of grouping. Only used when `verbose = TRUE`.

`verbose`: an optional logical value determining if the random effects estimates should be printed. Defaults to `FALSE`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

```
reStruct
```

EXAMPLE

```
rs1 <- reStruct(list(Dog = ~ day, Side = ~ 1), data = Pixel)
matrix(rs1) <- list(diag(2), 3)
print(rs1)
```

print.summary.corStruct

Print summary.corStruct

print.summary.corStruct

This method function prints the constrained coefficients of an initialized `corStruct` object, with a header specifying the type of correlation structure associated with the object.

```
print(x, ...)
```

ARGUMENTS

x: an object inheriting from class `summary.corStruct`, generally resulting from applying `summary` to an object inheriting from class `corStruct`.

...: optional arguments passed to `print.default`; see the documentation on that method function.

VALUE

the printed coefficients of **x** in constrained form, with a header specifying the associated correlation structure type.

SEE ALSO

`summary.corStruct`

EXAMPLE

```
cs1 <- corAR1(0.3)
print(summary(cs1))
```

print.summary.gls

Print a summary.gls Object

print.summary.gls

Information summarizing the fitted linear model represented by `x` is printed. This includes the AIC, BIC, and log-likelihood at convergence, the coefficient estimates and their respective standard errors, correlation and variance function parameters, if any are present, and the residual standard error.

```
print(x, verbose, ...)
```

ARGUMENTS

`x`: an object inheriting from class `summary.gls`, representing a summarized `gl`s object.

`verbose`: an optional logical value used to control the amount of printed output. Defaults to `FALSE`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`summary.gls`, `gl`s

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
print(summary(fml))
```

print.summary.lmList

Print a summary.lmList Object

print.summary.lmList

Information summarizing the individual `lm` fitted objects corresponding to `x` is printed. This includes the estimated coefficients and their respective standard errors, t-values, and p-values.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `summary.lmList`, representing a summarized `lme` object.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`summary.lmList`, `lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
print(summary(fml))
```

print.summary.lme

Print a summary.lme Object

print.summary.lme

Information summarizing the fitted linear mixed-effects model represented by `x` is printed. This includes the AIC, BIC, and log-likelihood at convergence, the fixed effects estimates and respective standard errors, the standard deviations and correlations for the random effects, the within-group correlation and variance function parameters, if any are present, and the within-group standard deviation.

```
print(x, verbose, ...)
```

ARGUMENTS

`x`: an object inheriting from class `summary.lme`, representing a summarized `lme` object.

`verbose`: an optional logical value used to control the amount of printed output. Defaults to `FALSE`.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`summary.lme`, `lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
print(summary((fml)))
```

print.summary.modelStruct	Print summary.modelStruct	print.summary.modelStruct
----------------------------------	---------------------------	----------------------------------

This method function prints the constrained coefficients of an initialized `modelStruct` object, with a header specifying the type of correlation structure associated with the object.

```
print(x, ...)
```

ARGUMENTS

- `x`: an object inheriting from class `summary.modelStruct`, generally resulting from applying `summary` to an object inheriting from class `modelStruct`.
- `...`: optional arguments passed to `print.default`; see the documentation on that method function.

VALUE

the printed coefficients of `x` in constrained form, with a header specifying the associated correlation structure type.

SEE ALSO

`summary.modelStruct`

EXAMPLE

```
cs1 <- corAR1(0.3)
print(summary(cs1))
```

print.summary.pdMat	Print summary.pdMat	print.summary.pdMat
----------------------------	---------------------	----------------------------

The standard deviations and correlations associated with the positive-definite matrix represented by `object` (considered as a variance-covariance matrix) are printed, together with the formula and the grouping level associated `object`, if any are present.

```
print(x, sigma, rdig, Level, resid, ...)
```

ARGUMENTS

- `x`: an object inheriting from class `summary.pdMat`, generally resulting from applying `summary` to an object inheriting from class `pdMat`.
- `sigma`: an optional numeric value used as a multiplier for the square-root factor of the positive-definite matrix represented by `object` (usually the estimated within-group standard deviation from a mixed-effects model). Defaults to 1.
- `rdig`: an optional integer value with the number of significant digits to be used in printing correlations. Defaults to 3.

Level: an optional character string with a description of the grouping level associated with **object** (generally corresponding to levels of grouping in a mixed-effects model). Defaults to **NULL**.

resid: an optional logical value. If **TRUE** an extra row with the "residual" standard deviation given in **sigma** will be included in the output. Defaults to **FALSE**.

...: optional arguments passed to **print.default**; see the documentation on that method function.

SEE ALSO

`summary.pdMat`, `pdMat`

EXAMPLE

```
pd1 <- pdCompSymm(3 * diag(3) + 1, form = ~ age + age^2,
  data = Orthodont)
print(summary(pd1), sigma = 1.2, resid = TRUE)
```

print.summary.varFunc

Print `summary.varFunc`

print.summary.varFunc

The variance function structure description, the formula and the coefficients associated with **x** are printed.

```
print(x, header, ...)
```

ARGUMENTS

x: an object inheriting from class `varFunc`, representing a variance function structure.

header: an optional logical value controlling whether a header should be included with the rest of the output. Defaults to **TRUE**.

...: optional arguments passed to **print.default**; see the documentation on that method function.

SEE ALSO

`summary.varFunc`

EXAMPLE

```
vf1 <- varPower(0.3, form = ~ age)
vf1 <- initialize(vf1, Orthodont)
print(summary(vf1))
```

print.varFunc

Print a varFunc Object

print.varFunc

The class and the coefficients associated with `x` are printed.

```
print(x, ...)
```

ARGUMENTS

`x`: an object inheriting from class `varFunc`, representing a variance function structure.

`...`: optional arguments passed to `print.default`; see the documentation on that method function.

SEE ALSO

`summary.varFunc`, `print.summary.varFunc`

EXAMPLE

```
vfl <- varPower(0.3, form = ~ age)
vfl <- initialize(vfl, Orthodont)
print(vfl)
```

pruneLevels

Prune Factor Levels

pruneLevels

The `levels` attribute of `object` are pruned to contain only the levels occurring in the factor.

```
pruneLevels(object)
```

ARGUMENTS

`object`: an object inheriting from class `factor`.

VALUE

an object identical to `object`, but with the `levels` attribute containing only value occurring in the factor.

SEE ALSO

`factor`, `ordered`

EXAMPLE

```
f1 <- factor(c(1,1,2,3,3,4,5))
levels(f1)
f2 <- f1[4:7]
levels(f2)
levels(pruneLevels(f2))
```

Diagnostic plots for assessing the normality of residuals the generalized least squares fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display.

```
qqnorm(object, form, abline, id, idLabels, grid, ...)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted model.

form: an optional one-sided formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `"."`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. The expression on the right hand side of `form` and to the left of a `|` operator must evaluate to a residuals vector. Default is `~resid(., type = "p")`, corresponding to a normal plot of the standardized residuals.

abline: an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.

id: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized residuals (random effects). Observations with absolute standardized residuals (random effects) greater than the $1 - \text{value}/2$ quantile of the standard normal distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.

idLabels: an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the observations identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified observations. Default is the innermost grouping factor.

grid: an optional logical value indicating whether a grid should be added to plot. Defaults to `FALSE`.

...: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot for assessing normality of residuals.

SEE ALSO

`gls`, `plot.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
qqnorm(fml, abline = c(0,1))
```

qqnorm.lme	Normal Plot of lme Residuals or Random Effects	qqnorm.lme
-------------------	--	-------------------

Diagnostic plots for assessing the normality of residuals and random effects in the linear mixed-effects fit are obtained. The `form` argument gives considerable flexibility in the type of plot specification. A conditioning expression (on the right side of a `|` operator) always implies that different panels are used for each level of the conditioning factor, according to a Trellis display.

```
qqnorm(object, form, abline, id, idLabels, grid, ...)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

form: an optional one-sided formula specifying the desired type of plot. Any variable present in the original data frame used to obtain `object` can be referenced. In addition, `object` itself can be referenced in the formula using the symbol `"."`. Conditional expressions on the right of a `|` operator can be used to define separate panels in a Trellis display. The expression on the right hand side of `form` and to the left of a `|` operator must evaluate to a residuals vector, or a random effects matrix. Default is `~resid(., type = "p")`, corresponding to a normal plot of the standardized residuals evaluated at the innermost level of nesting.

abline: an optional numeric value, or numeric vector of length two. If given as a single value, a horizontal line will be added to the plot at that coordinate; else, if given as a vector, its values are used as the intercept and slope for a line added to the plot. If missing, no lines are added to the plot.

id: an optional numeric value, or one-sided formula. If given as a value, it is used as a significance level for a two-sided outlier test for the standardized residuals (random effects). Observations with absolute standardized residuals (random effects) greater than the $1 - \text{value}/2$ quantile of the standard normal distribution are identified in the plot using `idLabels`. If given as a one-sided formula, its right hand side must evaluate to a logical, integer, or character vector which is used to identify observations in the plot. If missing, no observations are identified.

`idLabels`: an optional vector, or one-sided formula. If given as a vector, it is converted to character and used to label the observations identified according to `id`. If given as a one-sided formula, its right hand side must evaluate to a vector which is converted to character and used to label the identified observations. Default is the innermost grouping factor.

`grid`: an optional logical value indicating whether a grid should be added to plot. Defaults to `FALSE`.

`...`: optional arguments passed to the Trellis plot function.

VALUE

a diagnostic Trellis plot for assessing normality of residuals or random effects.

SEE ALSO

`lme`, `plot.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
# normal plot of standardized residuals by gender
qqnorm(fml, ~ resid(., type = "p") | Sex, abline = c(0, 1))
# normal plots of random effects
qqnorm(fml, ~ ranef(.))
```

random.effects

Extract Random Effects

random.effects

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `lmList` and `lme`.

```
random.effects(object, ...)
ranef(object, ...)
```

ARGUMENTS

`object`: any fitted model object from which random effects estimates can be extracted.

`...`: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

`ranef.lmList`, `ranef.lme`

EXAMPLE

```
## see the method function documentation
```

ranef

Extract Random Effects

ranef

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `lmList` and `lme`.

```
ranef(object, ...)
```

ARGUMENTS

object: any fitted model object from which random effects estimates can be extracted.

...: some methods for this generic function require additional arguments.

VALUE

will depend on the method function used; see the appropriate documentation.

SEE ALSO

```
ranef.lmList, ranef.lme
```

EXAMPLE

```
## see the method function documentation
```

ranef.lmList

Extract lmList Random Effects

ranef.lmList

A data frame containing the differences between the coefficients of the individual `lm` fits and the average coefficients.

```
ranef(object, augFrame, data, which, FUN, standard, omitGroupingFact
```

ARGUMENTS

object: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

augFrame: an optional logical value. If `TRUE`, the returned data frame is augmented with variables defined in the data frame used to produce `object`; else, if `FALSE`, only the random effects are returned. Defaults to `FALSE`.

data: an optional data frame with the variables to be used for augmenting the returned data frame when `augFrame = TRUE`. Defaults to the data frame used to fit `object`.

which: an optional positive integer or character vector specifying which columns of the data frame used to produce `object` should be used in the augmentation of the returned data frame. Defaults to all variables in the data.

FUN: an optional summary function or a list of summary functions to be applied to group-varying variables, when collapsing the data by groups. Group-invariant variables are always summarized by the unique value that they assume within that group. If **FUN** is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If **FUN** is a list of functions, the names in the list should designate classes of variables in the frame such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any group-varying variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

standard: an optional logical value indicating whether the estimated random effects should be "standardized" (i.e. divided by the corresponding estimated standard error). Defaults to `FALSE`.

omitGroupingFactor: an optional logical value. When `TRUE` the grouping factor itself will be omitted from the group-wise summary of data but the levels of the grouping factor will continue to be used as the row names for the returned data frame. Defaults to `FALSE`.

VALUE

a data frame with the differences between the individual `lm` coefficients in object and their average. Optionally, the returned data frame may be augmented with covariates summarized over groups or the differences may be standardized.

SEE ALSO

`lmList`, `fixef.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
ranef(fml)
ranef(fml, standard = TRUE)
ranef(fml, augFrame = TRUE)
```

The estimated random effects at level *i* are represented as a data frame with rows given by the different groups at that level and columns given by the random effects. If a single level of grouping is specified, the returned object is a data frame; else, the returned object is a list of such data frames. Optionally, the returned data frame(s) may be augmented with covariates summarized over groups.

```
ranef(object, augFrame, level, data, which, FUN, standard,
      omitGroupingFactor)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

augFrame: an optional logical value. If `TRUE`, the returned data frame is augmented with variables defined in `data`; else, if `FALSE`, only the coefficients are returned. Defaults to `FALSE`.

level: an optional vector of positive integers giving the levels of grouping to be used in extracting the random effects from an object with multiple nested grouping levels. Defaults to all levels of grouping.

data: an optional data frame with the variables to be used for augmenting the returned data frame when `augFrame = TRUE`. Defaults to the data frame used to fit `object`.

which: an optional positive integer vector specifying which columns of `data` should be used in the augmentation of the returned data frame. Defaults to all columns in `data`.

FUN: an optional summary function or a list of summary functions to be applied to group-varying variables, when collapsing `data` by groups. Group-invariant variables are always summarized by the unique value that they assume within that group. If `FUN` is a single function it will be applied to each non-invariant variable by group to produce the summary for that variable. If `FUN` is a list of functions, the names in the list should designate classes of variables in the frame such as `ordered`, `factor`, or `numeric`. The indicated function will be applied to any group-varying variables of that class. The default functions to be used are `mean` for numeric factors, and `Mode` for both `factor` and `ordered`. The `Mode` function, defined internally in `gsummary`, returns the modal or most popular value of the variable. It is different from the `mode` function that returns the S-language mode of the variable.

standard: an optional logical value indicating whether the estimated random effects should be "standardized" (i.e. divided by the corresponding estimated standard error). Defaults to `FALSE`.

`omitGroupingFactor`: an optional logical value. When `TRUE` the grouping factor itself will be omitted from the group-wise summary of data but the levels of the grouping factor will continue to be used as the row names for the returned data frame. Defaults to `FALSE`.

VALUE

a data frame, or list of data frames, with the estimated random effects at the grouping level(s) specified in `level` and, optionally, other covariates summarized over groups. The returned object inherits from classes `ranef.lme` and `data.frame`.

SEE ALSO

`lme`, `fixef.lme`, `coef.lme`, `plot.ranef.lme`, `gsummary`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
ranef(fml)
ranef(fml, augFrame = TRUE)
```

recalc	Recalculate Condensed Linear Model Object	recalc
---------------	---	---------------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include: `corStruct`, `modelStruct`, `reStruct`, and `varFunc`.

```
recalc(object, conLin)
```

ARGUMENTS

`object`: any object which induces a recalculation of the condensed linear model object `conLin`.

`conLin`: a condensed linear model object, consisting of a list with components `"Xy"`, corresponding to a regression matrix (`x`) combined with a response vector (`y`), and `"logLik"`, corresponding to the log-likelihood of the underlying model.

...: some methods for this generic function may require additional arguments.

VALUE

the recalculated condensed linear model object.

NOTE

This function is only used inside model fitting functions, such as `lme` and `gls`, that require recalculation of a condensed linear model object.

EXAMPLE

```
## see the method function documentation
```

This method function pre-multiplies the "Xy" component of `conLin` by the transpose square-root factor(s) of the correlation matrix (matrices) associated with `object` and adds the log-likelihood contribution of `object`, given by `logLik(object)`, to the "logLik" component of `conLin`.

```
recalc(object, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `corStruct`, representing a correlation structure.

`conLin`: a condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (x) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying model.

VALUE

the recalculated condensed linear model object.

NOTE

This method function is only used inside model fitting functions, such as `lme` and `gls`, that allow correlated error terms.

SEE ALSO

`corFactor`, `logLik.corStruct`

This method function recalculates the condensed linear model object using each element of `object` sequentially from last to first.

```
recalc(object, conLin)
```

ARGUMENTS

`object`: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

`conLin`: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (x) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying model. Defaults to `attr(object, "conLin")`.

VALUE

the recalculated condensed linear model object.

NOTE

This method function is primarily used inside model fitting functions, such as `lme` and `gls`, that allow model components, such as correlated error terms and variance functions.

SEE ALSO

`recalc.corStruct`, `recalc.reStruct`, `recalc.varFunc`

recalc.reStruct	Recalculate for reStruct Object	recalc.reStruct
------------------------	---------------------------------	------------------------

The log-likelihood, or restricted log-likelihood, of the Gaussian linear mixed-effects model represented by `object` and `conLin` (assuming spherical within-group covariance structure), evaluated at `coef(object)` is calculated and added to the `logLik` component of `conLin`. The `settings` attribute of `object` determines whether the log-likelihood, or the restricted log-likelihood, is to be calculated. The computational methods for the (restricted) log-likelihood calculations are described in Bates and Pinheiro (1998).

`recalc(object, conLin)`

ARGUMENTS

- `object`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.
- `conLin`: a condensed linear model object, consisting of a list with components "XY", corresponding to a regression matrix (X) combined with a response vector (Y), and "logLik", corresponding to the log-likelihood of the underlying model.

VALUE

the condensed linear model with its `logLik` component updated.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`reStruct`, `logLik`, `lme`

This method function pre-multiplies the "XY" component of `conLin` by a diagonal matrix with diagonal elements given by the weights corresponding to the variance structure represented by `object` and adds the log-likelihood contribution of `object`, given by `logLik(object)`, to the "logLik" component of `conLin`.

```
recalc(object, conLin)
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

conLin: a condensed linear model object, consisting of a list with components "XY", corresponding to a regression matrix (X) combined with a response vector (Y), and "logLik", corresponding to the log-likelihood of the underlying model.

VALUE

the recalculated condensed linear model object.

NOTE

This method function is only used inside model fitting functions, such as `lme` and `gls`, that allow heteroscedastic error terms.

SEE ALSO

`varWeights`, `logLik.varFunc`

The residuals for the linear model represented by `object` are extracted.

```
residuals(object, type)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

type: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "response".

VALUE

a vector with the residuals for the linear model represented by `object`.

SEE ALSO

`gls`, `fitted.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
residuals(fml)
```

residuals.glsStruct

Calculate glsStruct Residuals

residuals.glsStruct

The residuals for the linear model represented by `object` are extracted.

```
residuals(object, glsFit)
```

ARGUMENTS

`object`: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

`glsFit`: an optional list with components `logLik` (log-likelihood), `beta` (coefficients), `sigma` (standard deviation for error term), `varBeta` (coefficients' covariance matrix), `fitted` (fitted values), and `residuals` (residuals). Defaults to `attr(object, "glsFit")`.

VALUE

a vector with the residuals for the linear model represented by `object`.

NOTE

This method function is primarily used inside `gls` and `residuals.gls`.

SEE ALSO

`gls`, `residuals.gls`, `fitted.glsStruct`

The residuals for the linear model represented by `object` are extracted.

```
residuals(object, type)
```

ARGUMENTS

object: an object inheriting from class `gnls`, representing a generalized least squares fitted linear model.

type: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "response".

VALUE

a vector with the residuals for the linear model represented by `object`.

SEE ALSO

`gnls`, `fitted.gnls`

EXAMPLE

```
fml <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
             weights = varPower())
fitted(fml)
```

The residuals for the nonlinear model represented by `object` are extracted.

```
fitted(object)
```

ARGUMENTS

object: an object inheriting from class `gnlsStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects, and attributes specifying the underlying nonlinear model and the response variable.

VALUE

a vector with the residuals for the nonlinear model represented by `object`.

NOTE

This method function is primarily used inside `gnls` and `residuals.gnls`.

SEE ALSO

`gnls`, `residuals.gnlsStruct`, `fitted.gnlsStruct`

The residuals are extracted from each `lm` component of `object` and arranged into a list with as many components as `object`, or combined into a single vector.

```
residuals(object, type, subset, asList)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` objects with a common model.

`subset`: an optional character or integer vector naming the `lm` components of `object` from which the residuals are to be extracted. Default is `NULL`, in which case all components are used.

`type`: an optional character string specifying the type of residuals to be extracted. Options include `"response"` for the "raw" residuals (observed - fitted), `"pearson"` for the standardized residuals (raw residuals divided by the estimated residual standard error) using different standard errors for each `lm` fit, and `"pooled.pearson"` for the standardized residuals using a pooled estimate of the residual standard error. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to `"response"`.

`asList`: an optional logical value. If `TRUE`, the returned object is a list with the residuals split by groups; else the returned value is a vector. Defaults to `FALSE`.

VALUE

a list with components given by the residuals of each `lm` component of `object`, or a vector with the residuals for all `lm` components of `object`.

SEE ALSO

`lmList`, `fitted.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
residuals(fml)
```

The residuals at level *i* are obtained by subtracting the fitted levels at that level from the response vector (and dividing by the estimated within-group standard error, if `type="pearson"`). The fitted values at level *i* are obtained by adding together the population fitted values (based only on the fixed effects estimates) and the estimated contributions of the random effects to the fitted values at grouping levels less or equal to *i*.

```
residuals(object, level, type, asList)
```

ARGUMENTS

object: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

level: an optional integer vector giving the level(s) of grouping to be used in extracting the residuals from **object**. Level values increase from outermost to innermost grouping, with level zero corresponding to the population residuals. Defaults to the highest or innermost level of grouping.

type: an optional character string specifying the type of residuals to be used. If `"response"`, the "raw" residuals (observed - fitted) are used; else, if `"pearson"`, the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if `"normalized"`, the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to `"response"`.

asList: an optional logical value. If `TRUE` and a single value is given in **level**, the returned object is a list with the residuals split by groups; else the returned value is either a vector or a data frame, according to the length of **level**. Defaults to `FALSE`.

VALUE

if a single level of grouping is specified in **level**, the returned value is either a list with the residuals split by groups (`asList = TRUE`) or a vector with the residuals (`asList = FALSE`); else, when multiple grouping levels are specified in **level**, the returned object is a data frame with columns given by the residuals at different levels and the grouping factors.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`lme`, `fitted.lme`

EXAMPLE

```
fml <- lme(distance ~ age + Sex, data = Orthodont, random = ~ 1)
residuals(fml, level = 0:1)
```

residuals.lmeStruct

Calculate lmeStruct Residuals

residuals.lmeStruct

The residuals at level *i* are obtained by subtracting the fitted values at that level from the response vector. The fitted values at level *i* are obtained by adding together the population fitted values (based only on the fixed effects estimates) and the estimated contributions of the random effects to the fitted values at grouping levels less or equal to *i*.

```
residuals(object, levels, lmeFit, conLin)
```

ARGUMENTS

object: an object inheriting from class `lmeStruct`, representing a list of linear mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

level: an optional integer vector giving the level(s) of grouping to be used in extracting the residuals from `object`. Level values increase from outermost to innermost grouping, with level zero corresponding to the population fitted values. Defaults to the highest or innermost level of grouping.

lmeFit: an optional list with components `beta` and `b` containing respectively the fixed effects estimates and the random effects estimates to be used to calculate the residuals. Defaults to `attr(object, "lmeFit")`.

conLin: an optional condensed linear model object, consisting of a list with components `"Xy"`, corresponding to a regression matrix (*X*) combined with a response vector (*y*), and `"logLik"`, corresponding to the log-likelihood of the underlying lme model. Defaults to `attr(object, "conLin")`.

VALUE

if a single level of grouping is specified in `level`, the returned value is a vector with the residuals at the desired level; else, when multiple grouping levels are specified in `level`, the returned object is a matrix with columns given by the residuals at different levels.

NOTE

This method function is primarily used inside the `lme` function.

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`lme`, `residuals.lme`, `fitted.lmeStruct`

The residuals at level *i* are obtained by subtracting the fitted values at that level from the response vector. The fitted values at level *i* are obtained by adding together the contributions from the estimated fixed effects and the estimated random effects at levels less or equal to *i* and evaluating the model function at the resulting estimated parameters.

```
residuals(object, levels, conLin)
```

ARGUMENTS

object: an object inheriting from class `nlmeStruct`, representing a list of mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

level: an optional integer vector giving the level(s) of grouping to be used in extracting the residuals from `object`. Level values increase from outermost to innermost grouping, with level zero corresponding to the population fitted values. Defaults to the highest or innermost level of grouping.

conLin: an optional condensed linear model object, consisting of a list with components "Xy", corresponding to a regression matrix (X) combined with a response vector (y), and "logLik", corresponding to the log-likelihood of the underlying nlme model. Defaults to `attr(object, "conLin")`.

VALUE

if a single level of grouping is specified in `level`, the returned value is a vector with the residuals at the desired level; else, when multiple grouping levels are specified in `level`, the returned object is a matrix with columns given by the residuals at different levels.

NOTE

This method function is primarily used inside the `nlme` function

REFERENCES

Bates, D.M. and Pinheiro, J.C. (1998) "Computational methods for multilevel models" available in PostScript or PDF formats at <http://nlme.stat.wisc.edu>

SEE ALSO

`nlme`, `fitted.nlmeStruct`

This function is a constructor for the `reStruct` class, representing a random effects structure and consisting of a list of `pdMat` objects, plus a `settings` attribute containing information for the optimization algorithm used to fit the associated mixed-effects model.

```
reStruct(object, pdClass, REML, data)
```

ARGUMENTS

object: any of the following: (i) a one-sided formula of the form $\sim x_1 + \dots + x_n \mid g_1 / \dots / g_m$, with $x_1 + \dots + x_n$ specifying the model for the random effects and $g_1 / \dots / g_m$ the grouping structure (m may be equal to 1, in which case no `/` is required). The random effects formula will be repeated for all levels of grouping, in the case of multiple levels of grouping; (ii) a list of one-sided formulas of the form $\sim x_1 + \dots + x_n \mid g$, with possibly different random effects models for each grouping level. The order of nesting will be assumed the same as the order of the elements in the list; (iii) a one-sided formula of the form $\sim x_1 + \dots + x_n$, or a `pdMat` object with a formula (i.e. a non-NULL value for `formula(object)`), or a list of such formulas or `pdMat` objects. In this case, the grouping structure formula will be derived from the data used to fit the mixed-effects model, which should inherit from class `groupedData`; (iv) a named list of formulas or `pdMat` objects as in (iii), with the grouping factors as names. The order of nesting will be assumed the same as the order of the elements in the list; (v) an `reStruct` object.

pdClass: an optional character string with the name of the `pdMat` class to be used for the formulas in `object`. Defaults to `"pdSymm"` which corresponds to a general positive-definite matrix.

REML: an optional logical value. If `TRUE`, the associated mixed-effects model will be fitted using restricted maximum likelihood; else, if `FALSE`, maximum likelihood will be used. Defaults to `FALSE`.

data: an optional data frame in which to evaluate the variables used in the random effects formulas in `object`. It is used to obtain the levels for `factors`, which affect the dimensions and the row/column names of the underlying `pdMat` objects. If `NULL`, no attempt is made to obtain information on `factors` appearing in the formulas. Defaults to the parent frame from which the function was called.

VALUE

an object inheriting from class `reStruct`, representing a random effects structure.

SEE ALSO

`pdMat`, `lme`, `groupedData`

EXAMPLE

```
rsl <- reStruct(list(Dog = ~ day, Side = ~ 1), data = Pixel)
rsl
```

selfStart

Construct Self-starting Nonlinear Models

selfStart

This function is generic; methods functions can be written to handle specific classes of objects. Available methods include `selfStart.default` and `selfStart.formula`. See the documentation on the appropriate method function.

```
selfStart(model, initial, parameters, template)
```

VALUE

a function object of the `selfStart` class.

SEE ALSO

```
selfStart.default, selfStart.formula  
## see documentation for the methods
```

selfStart.default

Construct Self-starting Nonlinear Models

selfStart.default

A method for the generic function `selfStart` for function objects.

```
selfStart(model, initial, parameters, template)
```

ARGUMENTS

model: a function object defining a nonlinear model.

initial: a function object taking three arguments: `mCall`, `data`, and `LHS`, representing, respectively, a matched call to the function `model`, a data frame in which to interpret the variables in `mCall`, and the expression from the left hand side of the model formula in the call to `nls`. This function should return initial values for the parameters in `model`.

parameters, template: these arguments are included for consistency with the call to the generic function, but are not used in the default method. See the documentation on `selfStart.formula`.

VALUE

a function object of class `selfStart`, corresponding to a self-starting nonlinear model function. An `initial` attribute (defined by the `initial` argument) is added to the function to calculate starting estimates for the parameters in the model automatically.

SEE ALSO

```
selfStart.formula
```

EXAMPLE

```
# first.order.log.model is a function object defining a
# first order compartment model
# first.order.log.initial is a function object which calculates
# initial values for the parameters in first.order.log.model
# self-starting first order compartment model
SSfol <- selfStart(first.order.log.model, first.order.log.initial)
```

selfStart.formula

Construct Self-starting Nonlinear Models

selfStart.formula

A method for the generic function `selfStart` for formula objects.

```
selfStart(model, initial, parameters, template)
```

ARGUMENTS

model: a nonlinear formula object of the form `~expression`.

initial: a function object taking three arguments: `mCall`, `data`, and `LHS`, representing, respectively, a matched call to the function `model`, a data frame in which to interpret the variables in `mCall`, and the expression from the left hand side of the model formula in the call to `nls`. This function should return initial values for the parameters in `model`.

parameters: a character vector specifying the terms on the right hand side of `model` for which initial estimates should be calculated. Passed as the `namevec` argument to the `deriv` function.

template: an optional prototype for the calling sequence of the returned object, passed as the `function.arg` argument to the `deriv` function. By default, a template is generated with the covariates in `model` coming first and the parameters in `model` coming last in the calling sequence.

VALUE

a function object of class `selfStart`, obtained by applying `deriv` to the right hand side of the model formula. An `initial` attribute (defined by the `initial` argument) is added to the function to calculate starting estimates for the parameters in the model automatically.

SEE ALSO

```
selfStart.default, deriv
```

EXAMPLE

```
## self-starting logistic model
SSlogis <- selfStart(~ Asym/(1 + exp((xmid - x)/scal)),
  function(mCall, data, LHS)
  {
    xy <- sortedXyData(mCall[["x"]], LHS, data)
```

```

if(nrow(xy) < 4) {
  stop("Too few distinct x values to fit a logistic")
}
z <- xy[["y"]]
if (min(z) <= 0) { z <- z + 0.05 * max(z) } # avoid zeroes
z <- z/(1.05 * max(z)) # scale to within unit height
xy[["z"]] <- log(z/(1 - z)) # logit transformation
aux <- coef(lm(x ~ z, xy))
parameters(xy) <- list(xmid = aux[1], scal = aux[2])
pars <- as.vector(coef(nls(y ~ 1/(1 + exp((xmid - x)/scal)),
  data = xy, algorithm = "plinear")))
value <- c(pars[3], pars[1], pars[2])
names(value) <- mCall[c("Asym", "xmid", "scal")]
value
}, c("Asym", "xmid", "scal"))

```

simulate.lme

simulate lme models

simulate.lme

The model `m1` is fit to the data. Using the fitted values of the parameters, `nsim` new data vectors from this model are simulated. Both `m1` and `m2` are fit by maximum likelihood (ML) and/or by restricted maximum likelihood (REML) to each of the simulated data vectors.

```
simulate.lme(m1, m2, Random.seed, method, nsim, niterEM, useGen)
```

ARGUMENTS

- `m1`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model, or a list containing an lme model specification. If given as a list, it should contain components `fixed`, `data`, and `random` with values suitable for a call to `lme`. This argument defines the null model.
- `m2`: an `lme` object, or a list, like `m1` containing a second lme model specification. This argument defines the alternative model. If given as a list, only those parts of the specification that change between model `m1` and `m2` need to be specified.
- `Random.seed`: an optional vector to seed the random number generator so as to reproduce a simulation. This vector should be the same form as the `.Random.seed` object.
- `method`: an optional character array. If it includes "REML" the models are fit by maximizing the restricted log-likelihood. If it includes "ML" the log-likelihood is maximized. Defaults to `c("REML", "ML")`, in which case both methods are used.
- `nsim`: an optional positive integer specifying the number of simulations to perform. Defaults to 1000.

niterEM: an optional integer vector of length 2 giving the number of iterations of the EM algorithm to apply when fitting the `m1` and `m2` to each simulated set of data. Defaults to `c(40, 200)`.

useGen: an optional logical value. If `TRUE`, numerical derivatives are used to obtain the gradient and the Hessian of the log-likelihood in the optimization algorithm in the `ms` function. If `FALSE`, the default algorithm in `ms` for functions that do not incorporate gradient and Hessian attributes is used. Default depends on the `pdMat` classes used in `m1` and `m2`: if both are standard classes (see `pdClasses`) then defaults to `TRUE`, otherwise defaults to `FALSE`.

VALUE

an object of class `simulate.lme` with components `null` and `alt`. Each of these has components `ML` and/or `REML` which are matrices. An attribute called `Random.seed` contains the seed that was used for the random number generator.

SEE ALSO

`lme`

EXAMPLE

```
orthSim <-
  simulate.lme(m1 = list(fixed = distance ~ age, data = Orthodont,
                        random = ~ 1 | Subject),
              m2 = list(random = ~ age | Subject))
```

solve.pdMat	Calculate Inverse of a Positive-Definite Matrix	solve.pdMat
--------------------	---	--------------------

The positive-definite matrix represented by `a` is inverted and assigned to `a`.

```
solve(a, b, tol)
```

ARGUMENTS

a: an object inheriting from class `pdMat`, representing a positive definite matrix.

b: this argument is only included for consistency with the generic function and is not used in this method function.

tol: an optional numeric value for the tolerance used in the numerical algorithm. Defaults to `1e-7`.

VALUE

a `pdMat` object similar to `a`, but with coefficients corresponding to the inverse of the positive-definite matrix represented by `a`.

SEE ALSO

`pdMat`

EXAMPLE

```
pd1 <- pdCompSymm(3 * diag(3) + 1)
solve(pd1)
```

solve.reStruct

Apply Solve to an reStruct Object

solve.reStruct

Solve is applied to each `pdMat` component of `a`, which results in inverting the positive-definite matrices they represent.

```
solve(a, b, tol)
```

ARGUMENTS

- `a`: an object inheriting from class `reStruct`, representing a random effects structure and consisting of a list of `pdMat` objects.
- `b`: this argument is only included for consistency with the generic function and is not used in this method function.
- `tol`: an optional numeric value for the tolerance used in the numerical algorithm. Defaults to `1e-7`.

VALUE

an `reStruct` object similar to `a`, but with the `pdMat` components representing the inverses of the matrices represented by the components of `a`.

SEE ALSO

`solve.pdMat`, `reStruct`

EXAMPLE

```
rs1 <- reStruct(list(A = pdSymm(diag(1:3), form = ~ Score),
  B = pdDiag(2 * diag(4), form = ~ Educ)))
solve(rs1)
```


sortedXyData

Create a sortedXyData object

sortedXyData

This is a constructor function for the class of `sortedXyData` objects. These objects are mostly used in the `initial` function for a self-starting nonlinear regression model, which will be of the `selfStart` class.

```
sortedXyData(x, y, data)
```

ARGUMENTS

x: a numeric vector or an expression that will evaluate in `data` to a numeric vector

y: a numeric vector or an expression that will evaluate in `data` to a numeric vector

data: an optional data frame in which to evaluate expressions for `x` and `y`, if they are given as expressions

VALUE

A `sortedXyData` object. This is a data frame with exactly two numeric columns, named `x` and `y`. The rows are sorted so the `x` column is in increasing order. Duplicate `x` values are eliminated by averaging the corresponding `y` values.

SEE ALSO

```
selfStart, NLSstClosestX, NLSstLfAsymptote, NLSstRtAsymptote
```

EXAMPLE

```
DNase.2 <- DNase[ DNase$Run == "2", ]
sortedXyData( expression(log(conc)), expression(density), DNase.2 )
```

splitFormula

Split a Formula

splitFormula

Splits the right hand side of `form` into a list of subformulas according to the presence of `sep`. The left hand side of `form`, if present, will be ignored. The length of the returned list will be equal to the number of occurrences of `sep` in `form` plus one.

```
splitFormula(frm, sep)
```

ARGUMENTS

form: a formula object.

sep: an optional character string specifying the separator to be used for splitting the formula. Defaults to `" / "`.

VALUE

a list of formulas, corresponding to the split of `form` according to `sep`.

SEE ALSO

```
formula
```

EXAMPLE

```
splitFormula(~ g1/g2/g3)
```

SSasymp

Asymptotic regression model

SSasymp

This `selfStart` model evaluates the asymptotic regression function and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters `Asym`, `R0`, and `lrc` for a given set of data.

```
SSasymp(input, Asym, R0, lrc)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model

`Asym`: a numeric parameter representing the horizontal asymptote on the right side (very large values of `input`)

`R0`: a numeric parameter representing the response when `input` is zero.

`lrc`: a numeric parameter representing the natural logarithm of the rate constant

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $Asym + (R0 - Asym) \exp(-\exp(lrc)input)$. If all of the arguments `Asym`, `R0`, and `lrc` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

```
nls, selfStart
```

EXAMPLE

```
Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
SSasymp( Lob.329$age, 100, -8.5, -3.2 ) # response only
Asym <- 100
resp0 <- -8.5
lrc <- -3.2
SSasymp( Lob.329$age, Asym, resp0, lrc ) # response and gradient
```

This `selfStart` model evaluates an alternative parameterization of the asymptotic regression function and the gradient with respect to those parameters. It has an `initial` attribute that creates initial estimates of the parameters `Asym`, `lrc`, and `c0`.

```
SSasympOff(input, Asym, lrc, c0)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`Asym`: a numeric parameter representing the horizontal asymptote on the right side (very large values of `input`).

`lrc`: a numeric parameter representing the natural logarithm of the rate constant.

`c0`: a numeric parameter representing the `input` for which the response is zero.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $Asym\{1 - \exp[-\exp(lrc)(input - c0)]\}$. If all of the arguments `Asym`, `lrc`, and `c0` are names of objects, as opposed to expressions or explicit numerical values, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
CO2.Qn1 <- CO2[CO2$Plant == "Qn1", ]
SSasympOff( CO2.Qn1$conc, 32, 43, -4 ) # response only
Asym <- 32
lrc <- -4
c0 <- 43
SSasympOff( CO2.Qn1$conc, Asym, lrc, c0 ) # response and gradient
```

This `selfStart` model evaluates the asymptotic regression function through the origin and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters `Asym` and `lrc` for a given set of data.

```
SSasympOrig(input, Asym, lrc)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`Asym`: a numeric parameter representing the horizontal asymptote.

`lrc`: a numeric parameter representing the natural logarithm of the rate constant.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $Asym\{1 - \exp[-\exp(lrc) \cdot input]\}$. If all of the arguments `Asym` and `lrc` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
Lob.329 <- Loblolly[ Loblolly$Seed == "329", ]
SSasympOrig( Lob.329$age, 100, -3.2 ) # response only
Asym <- 100
lrc <- -3.2
SSasympOrig( Lob.329$age, Asym, lrc ) # response and gradient
```

This `selfStart` model evaluates the biexponential model function and its gradient. It has an `initial` attribute that creates evaluate initial estimates of the parameters `A1`, `lrc1`, `A2`, and `lrc2`.

```
SSbiexp(input, A1, lrc1, A2, lrc2)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`A1`: a numeric parameter representing the multiplier of the first exponential.

`lrc1`: a numeric parameter representing the natural logarithm of the rate constant of the first exponential.

`A2`: a numeric parameter representing the multiplier of the second exponential.

`lrc2`: a numeric parameter representing the natural logarithm of the rate constant of the second exponential.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $A1 \exp[-\exp(lrc1) \cdot input] + A2 \exp[-\exp(lrc2) \cdot input]$. If all of the arguments `A1`, `lrc1`, `A2`, and `lrc2` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
Indo.1 <- Indometh[Indometh$Subject == 1, ]
SSbiexp( Indo.1$time, 3, 1, 0.6, -1.3 ) # response only
A1 <- 3
lrc1 <- 1
A2 <- 0.6
lrc2 <- -1.3
SSbiexp( Indo.1$time, A1, lrc1, A2, lrc2 ) # response and gradient
```

This `selfStart` model evaluates the first-order compartment function and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters `lKe`, `lKa`, and `lCl` for a given set of data.

```
SSfol(Dose, input, lKe, lKa, lCl)
```

ARGUMENTS

`Dose`: a numeric value representing the initial dose.

`input`: a numeric vector at which to evaluate the model.

`lKe`: a numeric parameter representing the natural logarithm of the elimination rate constant.

`lKa`: a numeric parameter representing the natural logarithm of the absorption rate constant.

`lCl`: a numeric parameter representing the natural logarithm of the clearance.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression

$$\frac{Dose \exp(lKe + lKa - lCl)}{\exp(lKa) - \exp(lKe)} \{ \exp[-\exp(lKe) \cdot input] - \exp[-\exp(lKa) \cdot input] \}$$

. If all of the arguments `lKe`, `lKa`, and `lCl` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
Theoph.1 <- Theoph[ Theoph$Subject == 1, ]
# response only
SSfol( Theoph.1$Dose, Theoph.1$Time, -2.5, 0.5, -3 )
lKe <- -2.5
lKa <- 0.5
lCl <- -3
# response and gradient
SSfol( Theoph.1$Dose, Theoph.1$Time, lKe, lKa, lCl )
```

This `selfStart` model evaluates the four-parameter logistic function and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters `A`, `B`, `xmid`, and `scal` for a given set of data.

```
SSfpl(input, A, B, xmid, scal)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`A`: a numeric parameter representing the horizontal asymptote on the left side (very small values of `input`).

`B`: a numeric parameter representing the horizontal asymptote on the right side (very large values of `input`).

`xmid`: a numeric parameter representing the `input` value at the inflection point of the curve. The value of `SSfpl` will be midway between `A` and `B` at `xmid`.

`scal`: a numeric scale parameter on the `input` axis.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $A + (B - A) / \{1 + \exp[(xmid - input) / scal]\}$. If all of the arguments `A`, `B`, `xmid`, and `scal` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
Chick.1 <- ChickWeight[ChickWeight$Chick == 1, ]
SSfpl( Chick.1$Time, 13, 368, 14, 6 ) # response only
A <- 13
B <- 368
xmid <- 14
scal <- 6
SSfpl( Chick.1$Time, A, B, xmid, scal ) # response and gradient
```

This `selfStart` model evaluates the logistic function and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters `Asym`, `xmid`, and `scal` for a given set of data.

```
SSlogis(input, Asym, xmid, scal)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`Asym`: a numeric parameter representing the asymptote.

`xmid`: a numeric parameter representing the x value at the inflection point of the curve. The value of `SSlogis` will be `Asym/2` at `xmid`.

`scal`: a numeric scale parameter on the input axis.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $Asym / \{1 + \exp[(xmid - input)/scal]\}$. If all of the arguments `Asym`, `xmid`, and `scal` are names of objects, as opposed to expressions or explicit numerical values, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
Chick.1 <- ChickWeight[ChickWeight$Chick == 1, ]
SSlogis( Chick.1$Time, 368, 14, 6 ) # response only
Asym <- 368
xmid <- 14
scal <- 6
SSlogis( Chick.1$Time, Asym, xmid, scal ) # response and gradient
```


This `selfStart` model evaluates the Michaelis-Menten model and its gradient. It has an `initial` attribute that will evaluate initial estimates of the parameters V_m and K

```
SSmicmen(input, Vm, K)
```

ARGUMENTS

`input`: a numeric vector of values at which to evaluate the model.

`Vm`: a numeric parameter representing the maximum value of the response.

`K`: a numeric parameter representing the `input` value at which half the maximum response is attained. In the field of enzyme kinetics this is called the Michaelis parameter.

VALUE

a numeric vector of the same length as `input`. It is the value of the expression $V_m \cdot \text{input} / (K + \text{input})$. If both the arguments `Vm` and `K` are names of objects, the gradient matrix with respect to these names is attached as an attribute named `gradient`.

SEE ALSO

`nls`, `selfStart`

EXAMPLE

```
PurTrt <- Puromycin[ Puromycin$state == "treated", ]
SSmicmen( PurTrt$conc, 200, 0.05 ) # response only
Vm <- 200
K <- 0.05
SSmicmen( PurTrt$conc, Vm, K ) # response and gradient
```

summary.corStruct

Summarize a corStruct Object

summary.corStruct

This method function prepares `object` to be printed using the `print.summary` method, by changing its class and adding a `structName` attribute to it.

```
summary(object, structName)
```

ARGUMENTS

`object`: an object inheriting from class `corStruct`, representing a correlation structure.

`structName`: an optional character string defining the type of correlation structure associated with `object`, to be used in the `print.summary` method. Defaults to `class(object)[1]`.

VALUE

an object identical to `object`, but with its class changed to `summary.corStruct` and an additional attribute `structName`. The returned value inherits from the same classes as `object`.

SEE ALSO

```
print.summary.corStruct
```

EXAMPLE

```
cs1 <- corAR1(0.2)
summary(cs1)
```

summary.gls

Summarize a gls Object

summary.gls

Additional information about the linear model fit represented by `object` is extracted and included as components of `object`. The returned object is suitable for printing with the `print.summary.gls` method.

```
summary(object, adjustSigma, verbose)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

`adjustSigma`: an optional logical value. If `TRUE` and the estimation method used to obtain `object` was maximum likelihood, the standard errors of the coefficients are multiplied by $\sqrt{n_{obs}/(n_{obs} - n_{par})}$, with n_{obs} and n_{par} denoting, respectively, the number of observations and the number of coefficients. This converts the standard errors to REML-like estimates. Default is `TRUE`.

`verbose`: an optional logical value used to control the amount of output in the `print.summary.gls` method. Defaults to `FALSE`.

VALUE

an object inheriting from class `summary.gls` with all components included in object (see `glObject` for a full description of the components) plus the following components:

`corBeta`: approximate correlation matrix for the coefficients estimates

`tTable`: a data frame with columns `Value`, `Std. Error`, `t-value`, and `p-value` representing respectively the coefficients estimates, their approximate standard errors, the ratios between the estimates and their standard errors, and the associated p-value under a t approximation. Rows correspond to the different coefficients.

`residuals`: if more than five observations are used in the `gl`s fit, a vector with the minimum, first quartile, median, third quartile, and maximum of the residuals distribution; else the residuals.

`AIC`: the Akaike Information Criterion corresponding to object.

`BIC`: the Bayesian Information Criterion corresponding to object.

SEE ALSO

`gl`s, `AIC`, `BIC`, `print.summary.gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
           correlation = corAR1(form = ~ 1 | Mare))
summary(fml)
```

summary.lmList

Summarize an `lmList` Object

summary.lmList

The `summary.lm` method is applied to each `lm` component of `object` to produce summary information on the individual fits, which is organized into a list of summary statistics. The returned object is suitable for printing with the `print.summary.lmList` method.

```
summary(object, pool)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` fitted objects.

`pool`: an optional logical value indicating whether a pooled estimate of the residual standard error should be used. Default is `attr(object, "pool")`.

VALUE

a list with summary statistics obtained by applying `summary.lm` to the elements of `object`, inheriting from class `summary.lmList`. The components of value are:

`call`: a list containing an image of the `lmList` call that produced object.

`coefficients`: a three dimensional array with summary information on the `lm` coefficients. The first dimension corresponds to the names of the object components, the second dimension is given by "Value", "Std. Error", "t value", and " $\Pr(>|t|)$ ", corresponding, respectively, to the coefficient estimates and their associated standard errors, t-values, and p-values. The third dimension is given by the coefficients names.

`correlation`: a three dimensional array with the correlations between the individual `lm` coefficient estimates. The first dimension corresponds to the names of the object components. The third dimension is given by the coefficients names. For each coefficient, the rows of the associated array give the correlations between that coefficient and the remaining coefficients, by `lm` component.

`cov.unscaled`: a three dimensional array with the unscaled variances/covariances for the individual `lm` coefficient estimates (giving the estimated variance/covariance for the coefficients, when multiplied by the estimated residual errors). The first dimension corresponds to the names of the object components. The third dimension is given by the coefficients names. For each coefficient, the rows of the associated array give the unscaled covariances between that coefficient and the remaining coefficients, by `lm` component.

`df`: an array with the number of degrees of freedom for the model and for residuals, for each `lm` component.

`df.residual`: the total number of degrees of freedom for residuals, corresponding to the sum of residuals `df` of all `lm` components.

`fstatistics`: an array with the F test statistics and corresponding degrees of freedom, for each `lm` component.

`pool`: the value of the `pool` argument to the function.

`r.squared`: a vector with the multiple R-squared statistics for each `lm` component.

`residuals`: a list with components given by the residuals from individual `lm` fits.

`RSE`: the pooled estimate of the residual standard error.

`sigma`: a vector with the residual standard error estimates for the individual `lm` fits.

`terms`: the terms object used in fitting the individual `lm` components.

SEE ALSO

`lmList`, `print.summary.lmList`

EXAMPLE

```
fml <- lmList(distance ~ age | Subject, Orthodont)
summary(fml)
```

Additional information about the linear mixed-effects fit represented by `object` is extracted and included as components of `object`. The returned object is suitable for printing with the `print.summary.lme` method.

```
summary(object, adjustSigma, verbose)
```

ARGUMENTS

`object`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

`adjustSigma`: an optional logical value. If `TRUE` and the estimation method used to obtain `object` was maximum likelihood, the standard errors of the fixed effects are multiplied by $\sqrt{n_{obs}/(n_{obs} - n_{par})}$, with n_{obs} and n_{par} denoting, respectively, the number of observations and the number of fixed effects. This converts the standard errors to REML-like estimates. Default is `TRUE`.

`verbose`: an optional logical value used to control the amount of output in the `print.summary.lme` method. Defaults to `FALSE`.

VALUE

an object inheriting from class `summary.lme` with all components included in `object` (see `lmeObject` for a full description of the components) plus the following components:

`corFixed`: approximate correlation matrix for the fixed effects estimates

`tTable`: a data frame with columns `Value`, `Std. Error`, `DF`, `t-value`, and `p-value` representing respectively the fixed effects estimates, their approximate standard errors, the denominator degrees of freedom, the ratios between the estimates and their standard errors, and the associated p-value from a t distribution. Rows correspond to the different fixed effects.

`residuals`: if more than five observations are used in the `lme` fit, a vector with the minimum, 25 quantile, and maximum of the innermost grouping level residuals distribution; else the innermost grouping level residuals.

`AIC`: the Akaike Information Criterion corresponding to `object`.

`BIC`: the Bayesian Information Criterion corresponding to `object`.

SEE ALSO

`lme`, `AIC`, `BIC`, `print.summary.lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
summary(fml)
```

summary.modelStruct

Summarize modelStruct

summary.modelStruct

This method function applies `summary` to each element of `object`.

```
summary(object)
```

ARGUMENTS

object: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

VALUE

a list with elements given by the summarized components of `object`. The returned value is of class `summary.modelStruct`, also inheriting from the same classes as `object`.

SEE ALSO

```
print.summary.modelStruct
```

EXAMPLE

```
lms1 <- lmeStruct(reStruct = reStruct(pdDiag(diag(2), ~ age)),  
  corStruct = corAR1(0.3))  
summary(lms1)
```

summary.nlsList

Summarize an nlsList Object

summary.nlsList

The `summary.nls` method is applied to each `nls` component of `object` to produce summary information on the individual fits, which is organized into a list of summary statistics. The returned object is suitable for printing with the `print.summary.nlsList` method.

```
summary(object, pool)
```

ARGUMENTS

object: an object inheriting from class `nlsList`, representing a list of `nls` fitted objects.

pool: an optional logical value indicating whether a pooled estimate of the residual standard error should be used. Default is `attr(object, "pool")`.

VALUE

a list with summary statistics obtained by applying `summary.nls` to the elements of `object`, inheriting from class `summary.nlsList`. The components of value are:

call: a list containing an image of the `nlsList` call that produced `object`.

`parameters`: a three dimensional array with summary information on the `nls` coefficients. The first dimension corresponds to the names of the `object` components, the second dimension is given by "Value", "Std. Error", "t value", and "Pr(>|t|)", corresponding, respectively, to the coefficient estimates and their associated standard errors, t-values, and p-values. The third dimension is given by the coefficients names.

`correlation`: a three dimensional array with the correlations between the individual `nls` coefficient estimates. The first dimension corresponds to the names of the `object` components. The third dimension is given by the coefficients names. For each coefficient, the rows of the associated array give the correlations between that coefficient and the remaining coefficients, by `nls` component.

`cov.unscaled`: a three dimensional array with the unscaled variances/covariances for the individual `lm` coefficient estimates (giving the estimated variance/covariance for the coefficients, when multiplied by the estimated residual errors). The first dimension corresponds to the names of the `object` components. The third dimension is given by the coefficients names. For each coefficient, the rows of the associated array give the unscaled covariances between that coefficient and the remaining coefficients, by `nls` component.

`df`: an array with the number of degrees of freedom for the model and for residuals, for each `nls` component.

`df.residual`: the total number of degrees of freedom for residuals, corresponding to the sum of residuals `df` of all `nls` components.

`pool`: the value of the `pool` argument to the function.

`RSE`: the pooled estimate of the residual standard error.

`sigma`: a vector with the residual standard error estimates for the individual `lm` fits.

SEE ALSO

`nlsList`, `summary.nls`

EXAMPLE

```
fml <- nlsList(weight ~ SSlogis(Time, Asym, xmid, scal) | Plot,
               Soybean)
summary(fml)
```

Attributes `structName` and `noCorrelation`, with the values of the corresponding arguments to the method function, are appended to `object` and its class is changed to `summary.pdMat`.

```
summary(object, structName, noCorrelation)
```

ARGUMENTS

`object`: an object inheriting from class `pdMat`, representing a positive definite matrix.

`structName`: an optional character string with a description of the `pdMat` class. Default depends on the method function: "Blocked" for `pdBlocked`, "Compound Symmetry" for `pdCompSymm`, "Diagonal" for `pdDiag`, "Multiple of an Identity" for `pdIdent`, "General Positive-Definite" for `pdSymm`, and `data.class(object)` for `pdMat`.

`noCorrelation`: an optional logical value indicating whether correlations are to be printed in `print.summary.pdMat`. Default depends on the method function: `FALSE` for `pdDiag` and `pdIdent`, and `TRUE` for all other classes.

VALUE

an object similar to `object`, with additional attributes `structName` and `noCorrelation`, inheriting from class `summary.pdMat`.

SEE ALSO

```
print.summary.pdMat, pdMat
```

EXAMPLE

```
summary(pdSymm(diag(4)))
```


A `structName` attribute, with the value of corresponding argument, is appended to `object` and its class is changed to `summary.varFunc`.

```
summary(object, structName)
```

ARGUMENTS

`object`: an object inheriting from class `varFunc`, representing a variance function structure.

`structName`: an optional character string with a description of the `varFunc` class. Default depends on the method function: "Combination of variance functions" for `varComb`, "Constant plus power of covariate" for `varConstPower`, "Exponential of variance covariate" for `varExp`, "Different standard deviations per stratum" for `varIdent`, "Power of variance covariate" for `varPower`, and `data.class(object)` for `varFunc`.

VALUE

an object similar to `object`, with an additional attribute `structName`, inheriting from class `summary.varFunc`.

SEE ALSO

```
print.summary.varFunc
```

EXAMPLE

```
vf1 <- varPower(0.3, form = ~ age)
vf1 <- initialize(vf1, Orthodont)
summary(vf1)
```

The non-missing arguments in the call to the `update.gls` method replace the corresponding arguments in the original call used to produce `object` and `gls` is used with the modified call to produce an updated fitted object.

```
update(object, model, data, correlation, weights, subset, method,
       na.action, control)
```

ARGUMENTS

`object`: an object inheriting from class `gls`, representing a generalized least squares fitted linear model.

`other arguments`: defined as in `gls`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `gls` object.

SEE ALSO

`gls`

EXAMPLE

```
fml <- gls(follicles ~ sin(2*pi*Time) + cos(2*pi*Time), Ovary,
          correlation = corAR1(form = ~ 1 | Mare))
fm2 <- update(fml, weights = varPower())
```

The non-missing arguments in the call to the `update.gnls` method replace the corresponding arguments in the original call used to produce `object` and `gnls` is used with the modified call to produce an updated fitted object.

```
update(object, model, data, params, start, correlation, weights,
       subset, na.action, naPattern, control, verbose)
```

ARGUMENTS

`object`: an object inheriting from class `gnls`, representing a generalized nonlinear least squares fitted model.

`other arguments`: defined as in `gnls`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `gnls` object.

SEE ALSO

`gnls`

EXAMPLE

```
fm1 <- gnls(weight ~ SSlogis(Time, Asym, xmid, scal), Soybean,
            weights = varPower())
fm2 <- update(fm1, correlation = corAR1(form = ~ 1|Plot))
```

update.groupedData

Update a groupedData Object

update.groupedData

The non-missing arguments in the call to the `update.groupedData` method replace the corresponding arguments in the original call used to produce `object` and `groupedData` is used with the modified call to produce an updated fitted object.

```
update(object, formula, data, order.groups, FUN, outer, inner,
       labels, units)
```

ARGUMENTS

`object`: an object inheriting from class `groupedData`.

`other arguments`: defined as in `groupedData`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `groupedData` object.

SEE ALSO

`groupedData`

EXAMPLE

```
Orthodont2 <- update(Orthodont, FUN = mean)
```

update.lmList

Update an lmList Object

update.lmList

The non-missing arguments in the call to the `update.lmList` method replace the corresponding arguments in the original call used to produce `object` and `lmList` is used with the modified call to produce an updated fitted object.

```
update(object, formula, data, level, subset, na.action,  
        control, pool)
```

ARGUMENTS

`object`: an object inheriting from class `lmList`, representing a list of `lm` fitted objects.

`formula`: a two-sided linear formula with the common model for the individuals `lm` fits.

`other arguments`: defined as in `lmList`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `lmList` object.

SEE ALSO

`lmList`

EXAMPLE

```
fm1 <- lmList(Orthodont)  
fm2 <- update(fm1, distance ~ I(age - 11))
```

update.lme

Update an lme Object

update.lme

The non-missing arguments in the call to the `update.lme` method replace the corresponding arguments in the original call used to produce `object` and `lme` is used with the modified call to produce an updated fitted object.

```
update(object, fixed, data, random, correlation, weights,  
        subset, method, na.action, control)
```

ARGUMENTS

`object`: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.

`other arguments`: defined as in `lme`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `lme` object.

SEE ALSO

`lme`

EXAMPLE

```
fml <- lme(distance ~ age, Orthodont, random = ~ age | Subject)
fm2 <- update(fml, distance ~ age * Sex)
```

update.modelStruct

Update a modelStruct Object

update.modelStruct

This method function updates each element of `object`, allowing the access to data.

```
update(object, data)
```

ARGUMENTS

object: an object inheriting from class `modelStruct`, representing a list of model components, such as `corStruct` and `varFunc` objects.

data: a data frame in which to evaluate the variables needed for updating the elements of `object`.

VALUE

an object similar to `object` (same class, length, and names), but with updated elements.

NOTE

This method function is primarily used within model fitting functions, such as `lme` and `gls`, that allow model components such as variance functions.

update.nlme

Update an nlme Object

update.nlme

The non-missing arguments in the call to the `update.nlme` method replace the corresponding arguments in the original call used to produce `object` and `nlme` is used with the modified call to produce an updated fitted object.

```
update(object, model, data, fixed, random, groups, start,
        correlation, weights, subset, method, na.action,
        naPattern, control, verbose)
```

ARGUMENTS

object: an object inheriting from class `nlme`, representing a fitted nonlinear mixed-effects model.

other arguments: defined as in `nlme`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `nlme` object.

SEE ALSO

`nlme`

EXAMPLE

```
fm1 <- nlme(weight ~ SSlogis(Time, Asym, xmid, scal),
             data = Soybean, fixed = Asym + xmid + scal ~ 1,
             start = c(18, 52, 7.5))
fm2 <- update(fm1, weights = varPower())
```

update.nlsList

Update an nlsList Object

update.nlsList

The non-missing arguments in the call to the `update.nlsList` method replace the corresponding arguments in the original call used to produce `object` and `nlsList` is used with the modified call to produce an updated fitted object.

```
update(object, model, data, start, control, level, subset,
        na.action, control, pool)
```

ARGUMENTS

`object`: an object inheriting from class `nlsList`, representing a list of fitted `nls` objects.

`other arguments`: defined as in `nlsList`. See that function's documentation for descriptions of and default values for these arguments.

VALUE

an updated `nlsList` object.

SEE ALSO

`nlsList`

EXAMPLE

```
fm1 <- nlsList(weight ~ SSlogis(Time, Asym, xmid, scal) | Plot,
               Soybean)
fm2 <- update(fm1, start = list(Asym = 23, xmid = 57, scal = 9))
```

If the `formula(object)` includes a "." term, representing a fitted object, the variance covariate needs to be updated upon completion of an optimization cycle (in which the variance function weights are kept fixed). This method function allows a reevaluation of the variance covariate using the current fitted object and, optionally, other variables in the original data.

```
update(object, data)
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

data: a list with a component named "." with the current version of the fitted object (from which fitted values, coefficients, and residuals can be extracted) and, if necessary, other variables used to evaluate the variance covariate(s).

VALUE

if `formula(object)` includes a "." term, an `varFunc` object similar to `object`, but with the variance covariate reevaluated at the current fitted object value; else `object` is returned unchanged.

SEE ALSO

```
needUpdate, covariate<-.varFunc
```

Standard classes of variance function structures (`varFunc`) available in the `nlme` library. Covariates included in the variance function, denoted by variance covariates, may involve functions of the fitted model object, such as the fitted values and the residuals. Different coefficients may be assigned to the levels of a classification factor.

STANDARD CLASSES

varExp: exponential of a variance covariate.

varPower: power of a variance covariate.

varConstPower: constant plus power of a variance covariate.

varIdent: constant variance(s), generally used to allow different variances according to the levels of a classification factor.

varFixed: fixed weights, determined by a variance covariate.

`varComb`: combination of variance functions.

NOTE

Users may define their own `varFunc` classes by specifying a constructor function and, at a minimum, methods for the functions `coef`, `coef<-`, and `initialize`. For examples of these functions, see the methods for class `varPower`.

SEE ALSO

`varExp`, `varPower`, `varConstPower`, `varIdent`, `varFixed`, `varComb`

varComb	Combination of Variance Functions	varComb
----------------	-----------------------------------	----------------

This function is a constructor for the `varComb` class, representing a combination of variance functions. The corresponding variance function is equal to the product of the variance functions of the `varFunc` objects listed in . . .

```
varComb(...)
```

ARGUMENTS

. . . : objects inheriting from class `varFunc` representing variance function structures.

VALUE

a `varComb` object representing a combination of variance functions, also inheriting from class `varFunc`.

SEE ALSO

`varWeights.varComb`, `coef.varComb`

EXAMPLE

```
vf1 <- varComb(varIdent(form = ~ 1|Sex), varPower())
```


This function is a constructor for the `varConstPower` class, representing a constant plus power variance function structure. Letting v denote the variance covariate and $\sigma^2(v)$ denote the variance function evaluated at v , the constant plus power variance function is defined as $\sigma^2(v) = (\theta_1 + |v|^{\theta_2})^2$, where θ_1, θ_2 are the variance function coefficients. When a grouping factor is present, different θ_1, θ_2 are used for each factor level.

```
varConstPower(const, power, form, fixed)
```

ARGUMENTS

- `const, power`: optional numeric vectors, or lists of numeric values, with, respectively, the coefficients for the constant and the power terms. Both arguments must have length one, unless a grouping factor is specified in `form`. If either argument has length greater than one, it must have names which identify its elements to the levels of the grouping factor defined in `form`. If a grouping factor is present in `form` and the argument has length one, its value will be assigned to all grouping levels. Only positive values are allowed for `const`. Default is `numeric(0)`, which results in a vector of zeros of appropriate length being assigned to the coefficients when object is initialized (corresponding to constant variance equal to one).
- `form`: an optional one-sided formula of the form $\sim v$, or $\sim v \mid g$, specifying a variance covariate v and, optionally, a grouping factor g for the coefficients. The variance covariate must evaluate to a numeric vector and may involve expressions using `"."`, representing a fitted model object from which fitted values (`fitted(.)`) and residuals (`resid(.)`) can be extracted (this allows the variance covariate to be updated during the optimization of an objective function). When a grouping factor is present in `form`, a different coefficient value is used for each of its levels. Several grouping variables may be simultaneously specified, separated by the `*` operator, like in $\sim v \mid g1 * g2 * g3$. In this case, the levels of each grouping variable are pasted together and the resulting factor is used to group the observations. Defaults to $\sim \text{fitted}(.)$ representing a variance covariate given by the fitted values of a fitted model object and no grouping factor.
- `fixed`: an optional list with components `const` and/or `power`, consisting of numeric vectors, or lists of numeric values, specifying the values at which some or all of the coefficients in the variance function should be fixed. If a grouping factor is specified in `form`, the components of `fixed` must have names identifying which coefficients are to be fixed. Coefficients included in `fixed` are not allowed to vary during the optimization of an objective function. Defaults to `NULL`, corresponding to no fixed coefficients.

VALUE

a `varConstPower` object representing a constant plus power variance function structure, also inheriting from class `varFunc`.

SEE ALSO

`varWeights.varFunc`, `coef.varConstPower`

EXAMPLE

```
vf1 <- varConstPower(1.2, 0.2, form = ~ age|Sex)
```

VarCorr	Extract variance and correlation components	VarCorr
----------------	---	----------------

This function calculates the estimated variances, standard deviations, and correlations between the random-effects terms in a linear mixed-effects model, of class `lme`, or a nonlinear mixed-effects model, of class `nlme`. The within-group error variance and standard deviation are also calculated.

```
VarCorr(object, sigma, rdig)
```

ARGUMENTS

object: a fitted model object, usually an object inheriting from class `lme`.

sigma: an optional numeric value used as a multiplier for the standard deviations. Default is 1.

rdig: an optional integer value specifying the number of digits used to represent correlation estimates. Default is 3.

VALUE

a matrix with the estimated variances, standard deviations, and correlations for the random effects. The first two columns, named `Variance` and `StdDev`, give, respectively, the variance and the standard deviations. If there are correlation components in the random effects model, the third column, named `Corr`, and the remaining unnamed columns give the estimated correlations among random effects within the same level of grouping. The within-group error variance and standard deviation are included as the last row in the matrix.

EXAMPLE

```
fml <- lme(distance ~ age, data = Orthodont, random = ~ age)
VarCorr(fml)
```

This function is a constructor for the `varExp` class, representing an exponential variance function structure. Letting v denote the variance covariate and $\sigma^2(v)$ denote the variance function evaluated at v , the exponential variance function is defined as $\sigma^2(v) = \exp(2\theta v)$, where θ is the variance function coefficient. When a grouping factor is present, a different θ is used for each factor level.

```
varExp(value, form, fixed)
```

ARGUMENTS

- value:** an optional numeric vector, or list of numeric values, with the variance function coefficients. `value` must have length one, unless a grouping factor is specified in `form`. If `value` has length greater than one, it must have names which identify its elements to the levels of the grouping factor defined in `form`. If a grouping factor is present in `form` and `value` has length one, its value will be assigned to all grouping levels. Default is `numeric(0)`, which results in a vector of zeros of appropriate length being assigned to the coefficients when object is initialized (corresponding to constant variance equal to one).
- form:** an optional one-sided formula of the form `~v`, or `~v | g`, specifying a variance covariate `v` and, optionally, a grouping factor `g` for the coefficients. The variance covariate must evaluate to a numeric vector and may involve expressions using `"."`, representing a fitted model object from which fitted values (`fitted(.)`) and residuals (`resid(.)`) can be extracted (this allows the variance covariate to be updated during the optimization of an objective function). When a grouping factor is present in `form`, a different coefficient value is used for each of its levels. Several grouping variables may be simultaneously specified, separated by the `*` operator, like in `~v | g1 * g2 * g3`. In this case, the levels of each grouping variable are pasted together and the resulting factor is used to group the observations. Defaults to `~fitted(.)` representing a variance covariate given by the fitted values of a fitted model object and no grouping factor.
- fixed:** an optional numeric vector, or list of numeric values, specifying the values at which some or all of the coefficients in the variance function should be fixed. If a grouping factor is specified in `form`, `fixed` must have names identifying which coefficients are to be fixed. Coefficients included in `fixed` are not allowed to vary during the optimization of an objective function. Defaults to `NULL`, corresponding to no fixed coefficients.

VALUE

a `varExp` object representing an exponential variance function structure, also inheriting from class `varFunc`.

SEE ALSO

```
varWeights.varFunc, coef.varExp
```

EXAMPLE

```
vf1 <- varExp(0.2, form = ~ age|Sex)
```

varFixed

Fixed Variance Function

varFixed

This function is a constructor for the `varFixed` class, representing a variance function with fixed variances. Letting v denote the variance covariate defined in `value`, the variance function $\sigma^2(v)$ for this class is $\sigma^2(v) = |v|$. The variance covariate v is evaluated once at initialization and remains fixed thereafter. No coefficients are required to represent this variance function.

```
varFixed(value)
```

ARGUMENTS

value: a one-sided formula of the form `~v` specifying a variance covariate `v`. Grouping factors are ignored.

VALUE

a `varFixed` object representing a fixed variance function structure, also inheriting from class `varFunc`.

SEE ALSO

```
varWeights.varFunc, varFunc
```

EXAMPLE

```
vf1 <- varFixed(~ age)
```

varFunc

Variance Function Structure

varFunc

If `object` is a one-sided formula, it is used as the argument to `varFixed` and the resulting object is returned. Else, if `object` inherits from class `varFunc`, it is returned unchanged.

```
varFunc(object)
```

ARGUMENTS

object: either an one-sided formula specifying a variance covariate, or an object inheriting from class `varFunc`, representing a variance function structure.

VALUE

an object from class `varFunc`, representing a variance function structure.

SEE ALSO

```
varFixed, varWeights.varFunc, coef.varFunc
```

EXAMPLE

```
vfl <- varFunc(~ age)
```

varIdent	Different Variances per Group	varIdent
-----------------	-------------------------------	-----------------

This function is a constructor for the `varIdent` class, representing a constant variance function structure. If no grouping factor is present in `form`, the variance function is constant and equal to one, and no coefficients required to represent it. When `form` includes a grouping factor with $M > 1$ levels, the variance function allows M different variances, one for each level of the factor. For identifiability reasons, the coefficients of the variance function represent the ratios between the variances and a reference variance (corresponding to a reference group level). Therefore, only $M - 1$ coefficients are needed to represent the variance function. By default, if the elements in `value` are unnamed, the first group level is taken as the reference level.

```
varIdent(value, form, fixed)
```

ARGUMENTS

- value:** an optional numeric vector, or list of numeric values, with the variance function coefficients. If no grouping factor is present in `form`, this argument is ignored, as the resulting variance function contains no coefficients. If `value` has length one, its value is repeated for all coefficients in the variance function. If `value` has length greater than one, it must have length equal to the number of grouping levels minus one and names which identify its elements to the levels of the grouping factor. Only positive values are allowed for this argument. Default is `numeric(0)`, which results in a vector of zeros of appropriate length being assigned to the coefficients when `object` is initialized (corresponding to constant variance equal to one).
- form:** an optional one-sided formula of the form `~v`, or `~v | g`, specifying a variance covariate `v` and, optionally, a grouping factor `g` for the coefficients. The variance covariate is ignored in this variance function. When a grouping factor is present in `form`, a different coefficient value is used for each of its levels less one reference level (see description section below). Several grouping variables may be simultaneously specified, separated by the `*` operator, like in `~v | g1 * g2 * g3`. In this case, the levels of each grouping variable are pasted together and the resulting factor is used to group the observations. Defaults to `~1`.
- fixed:** an optional numeric vector, or list of numeric values, specifying the values at which some or all of the coefficients in the variance function should be fixed. It must have names identifying which coefficients are to be fixed. Coefficients included in `fixed` are not allowed to vary during the optimization of an objective function. Defaults to `NULL`, corresponding to no fixed coefficients.

VALUE

a `varIdent` object representing a constant variance function structure, also inheriting from class `varFunc`.

SEE ALSO

`varWeights.varFunc`, `coef.varIdent`

EXAMPLE

```
vfl <- varIdent(c(F = 0.5), form = ~ 1 | Sex)
```

Variogram	Calculate Semi-Variogram	Variogram
-----------	--------------------------	-----------

This function is generic; method functions can be written to handle specific classes of objects. Classes which already have methods for this function include `default`, `gls` and `lme`. See the appropriate method documentation for a description of the arguments.

```
Variogram(object, distance, ...)
```

VALUE

will depend on the method function used; see the appropriate documentation.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
Diggle, P.J., Liang, K.Y. and Zeger, S. L. (1994) "Analysis of Longitudinal Data", Oxford University Press Inc.

SEE ALSO

`Variogram.default`, `Variogram.gls`, `Variogram.lme`, `plot.Variogram`

EXAMPLE

```
## see the method function documentation
```

Variogram.corExp Calculate Semi-Variogram for a corExp Object **Variogram.corExp**

This method function calculates the semi-variogram values corresponding to the Exponential correlation model, using the estimated coefficients corresponding to object, at the distances defined by distance.

```
Variogram(object, distance, sig2, length.out)
```

ARGUMENTS

object: an object inheriting from class `corExp`, representing an exponential spatial correlation structure.

distance: an optional numeric vector with the distances at which the semi-variogram is to be calculated. Defaults to `NULL`, in which case a sequence of length `length.out` between the minimum and maximum values of `getCovariate(object)` is used.

sig2: an optional numeric value representing the process variance. Defaults to 1.

length.out: an optional integer specifying the length of the sequence of distances to be used for calculating the semi-variogram, when `distance = NULL`. Defaults to 50.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

SEE ALSO

`corExp`, `plot.Variogram`

EXAMPLE

```
cs1 <- corExp(3, form = ~ Time | Rat)
cs1 <- initialize(cs1, BodyWeight)
Variogram(cs1)[1:10,]
```

Variogram.corGaus Calculate Semi-Variogram for a corGaus Object Variogram.corGaus

This method function calculates the semi-variogram values corresponding to the Gaussian correlation model, using the estimated coefficients corresponding to object, at the distances defined by distance.

```
Variogram(object, distance, sig2, length.out)
```

ARGUMENTS

object: an object inheriting from class `corGaus`, representing an Gaussian spatial correlation structure.

distance: an optional numeric vector with the distances at which the semi-variogram is to be calculated. Defaults to `NULL`, in which case a sequence of length `length.out` between the minimum and maximum values of `getCovariate(object)` is used.

sig2: an optional numeric value representing the process variance. Defaults to 1.

length.out: an optional integer specifying the length of the sequence of distances to be used for calculating the semi-variogram, when `distance = NULL`. Defaults to 50.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

SEE ALSO

`corGaus`, `plot.Variogram`

EXAMPLE

```
cs1 <- corGaus(3, form = ~ Time | Rat)
cs1 <- initialize(cs1, BodyWeight)
Variogram(cs1)[1:10,]
```


Variogram.corLin Calculate Semi-Variogram for a corLin Object Variogram.corLin
--

This method function calculates the semi-variogram values corresponding to the Linear correlation model, using the estimated coefficients corresponding to object, at the distances defined by distance.

```
Variogram(object, distance, sig2, length.out)
```

ARGUMENTS

object: an object inheriting from class `corLin`, representing an Linear spatial correlation structure.

distance: an optional numeric vector with the distances at which the semi-variogram is to be calculated. Defaults to `NULL`, in which case a sequence of length `length.out` between the minimum and maximum values of `getCovariate(object)` is used.

sig2: an optional numeric value representing the process variance. Defaults to 1.

length.out: an optional integer specifying the length of the sequence of distances to be used for calculating the semi-variogram, when `distance = NULL`. Defaults to 50.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

SEE ALSO

`corLin`, `plot.Variogram`

EXAMPLE

```
cs1 <- corLin(15, form = ~ Time | Rat)
cs1 <- initialize(cs1, BodyWeight)
Variogram(cs1)[1:10,]
```

Variogram.corRatio Calculate Semi-Variogram for a corRatio Object Variogram.corRatio

This method function calculates the semi-variogram values corresponding to the Rational Quadratic correlation model, using the estimated coefficients corresponding to `object`, at the distances defined by `distance`.

```
Variogram(object, distance, sig2, length.out)
```

ARGUMENTS

`object`: an object inheriting from class `corRatio`, representing an Rational Quadratic spatial correlation structure.

`distance`: an optional numeric vector with the distances at which the semi-variogram is to be calculated. Defaults to `NULL`, in which case a sequence of length `length.out` between the minimum and maximum values of `getCovariate(object)` is used.

`sig2`: an optional numeric value representing the process variance. Defaults to 1.

`length.out`: an optional integer specifying the length of the sequence of distances to be used for calculating the semi-variogram, when `distance = NULL`. Defaults to 50.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

SEE ALSO

`corRatio`, `plot.Variogram`

EXAMPLE

```
cs1 <- corRatio(7, form = ~ Time | Rat)
cs1 <- initialize(cs1, BodyWeight)
Variogram(cs1)[1:10,]
```

Variogram.corSpherCalculate Semi-Variogram for a corSpher ObjectVariogram.corSpher

This method function calculates the semi-variogram values corresponding to the Spherical correlation model, using the estimated coefficients corresponding to object, at the distances defined by distance.

```
Variogram(object, distance, sig2, length.out)
```

ARGUMENTS

object: an object inheriting from class `corSpher`, representing an Spherical spatial correlation structure.

distance: an optional numeric vector with the distances at which the semi-variogram is to be calculated. Defaults to `NULL`, in which case a sequence of length `length.out` between the minimum and maximum values of `getCovariate(object)` is used.

sig2: an optional numeric value representing the process variance. Defaults to 1.

length.out: an optional integer specifying the length of the sequence of distances to be used for calculating the semi-variogram, when `distance = NULL`. Defaults to 50.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

SEE ALSO

`corSpher`, `plot.Variogram`

EXAMPLE

```
cs1 <- corSpher(15, form = ~ Time | Rat)
cs1 <- initialize(cs1, BodyWeight)
Variogram(cs1)[1:10,]
```

This method function calculates the semi-variogram for an arbitrary vector object, according to the distances in `distance`. For each pair of elements x, y in object, the corresponding semi-variogram is $(x - y)^2/2$. The semi-variogram is useful for identifying and modeling spatial correlation structures in observations with constant expectation and constant variance.

```
Variogram(object, distance)
```

ARGUMENTS

object: a numeric vector with the values to be used in calculating the semi-variogram, usually a residual vector from a fitted model.

distance: a numeric vector with the pairwise distances corresponding to the elements of object. The order of the elements in `distance` must correspond to the pairs $(1,2), (1,3), \dots, (n-1,n)$, with n representing the length of object, and must have length $n(n-1)/2$.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.

Diggle, P.J., Liang, K.Y. and Zeger, S. L. (1994) "Analysis of Longitudinal Data", Oxford University Press Inc.

SEE ALSO

`Variogram.gls`, `Variogram.lme`, `plot.Variogram`

EXAMPLE

```
fml <- lm(follicles ~ sin(2 * pi * Time) + cos(2 * pi * Time),
         Ovary, subset = Mare == 1)
Variogram(resid(fml), dist(1:29))[1:10,]
```

This method function calculates the semi-variogram for the residuals from an `gls` fit. The semi-variogram values are calculated for pairs of residuals within the same group level, if a grouping factor is present. If `collapse` is different from "none", the individual semi-variogram values are collapsed using either a robust estimator (`robust = TRUE`) defined in Cressie (1993), or the average of the values within the same distance interval. The semi-variogram is useful for modeling the error term correlation structure.

```
Variogram(object, distance, form, resType, data, na.action, maxDist,
          length.out, collapse, nint, breaks, robust, metric)
```

ARGUMENTS

object: an object inheriting from class `gls`, representing a generalized least squares fitted model.

distance: an optional numeric vector with the distances between residual pairs. If a grouping variable is present, only the distances between residual pairs within the same group should be given. If missing, the distances are calculated based on the values of the arguments `form`, `data`, and `metric`, unless `object` includes a `corSpatial` element, in which case the associated covariate (obtained with the `getCovariate` method) is used.

form: an optional one-sided formula specifying the covariate(s) to be used for calculating the distances between residual pairs and, optionally, a grouping factor for partitioning the residuals (which must appear to the right of a `|` operator in `form`). Default is `~1`, implying that the observation order within the groups is used to obtain the distances.

resType: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "pearson".

data: an optional data frame in which to interpret the variables in `form`. By default, the same data used to fit `object` is used.

na.action: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes an error message to be printed and the function to terminate, if there are any incomplete observations.

maxDist: an optional numeric value for the maximum distance used for calculating the semi-variogram between two residuals. By default all residual pairs are included.

`length.out`: an optional integer value. When object includes a `corSpatial` element, its semi-variogram values are calculated and this argument is used as the `length.out` argument to the corresponding Variogram method. Defaults to 50.

`collapse`: an optional character string specifying the type of collapsing to be applied to the individual semi-variogram values. If equal to "quantiles", the semi-variogram values are split according to quantiles of the distance distribution, with equal number of observations per group, with possibly varying distance interval lengths. Else, if "fixed", the semi-variogram values are divided according to distance intervals of equal lengths, with possibly different number of observations per interval. Else, if "none", no collapsing is used and the individual semi-variogram values are returned. Defaults to "quantiles".

`nint`: an optional integer with the number of intervals to be used when collapsing the semi-variogram values. Defaults to 20.

`robust`: an optional logical value specifying if a robust semi-variogram estimator should be used when collapsing the individual values. If TRUE the robust estimator is used. Defaults to FALSE.

`breaks`: an optional numeric vector with the breakpoints for the distance intervals to be used in collapsing the semi-variogram values. If not missing, the option specified in `collapse` is ignored.

`metric`: an optional character string specifying the distance metric to be used. The currently available options are "euclidean" for the root sum-of-squares of distances; "maximum" for the maximum difference; and "manhattan" for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to "euclidean".

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. If the semi-variogram values are collapsed, an extra column, `n.pairs`, with the number of residual pairs used in each semi-variogram calculation, is included in the returned data frame. If object includes a `corSpatial` element, a data frame with its corresponding semi-variogram is included in the returned value, as an attribute "modelVariog". The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons.
Diggle, P.J., Liang, K.Y. and Zeger, S. L. (1994) "Analysis of Longitudinal Data", Oxford University Press Inc.

SEE ALSO

`gls`, `Variogram.default`, `Variogram.gls`, `plot.Variogram`

EXAMPLE

```
fml <- gls(weight ~ Time * Diet, BodyWeight)
Variogram(fml, form = ~ Time | Rat, nint = 10, robust = TRUE)
```

Variogram.lme Calculate Semi-Variogram for Residuals from an lme Object **Variogram.lme**

This method function calculates the semi-variogram for the within-group residuals from an `lme` fit. The semi-variogram values are calculated for pairs of residuals within the same group. If `collapse` is different from "none", the individual semi-variogram values are collapsed using either a robust estimator (`robust = TRUE`) defined in Cressie (1993), or the average of the values within the same distance interval. The semi-variogram is useful for modeling the error term correlation structure.

```
Variogram(object, distance, form, resType, data, na.action, maxDist,
          length.out, collapse, nint, breaks, robust, metric)
```

ARGUMENTS

- object**: an object inheriting from class `lme`, representing a fitted linear mixed-effects model.
- distance**: an optional numeric vector with the distances between residual pairs. If a grouping variable is present, only the distances between residual pairs within the same group should be given. If missing, the distances are calculated based on the values of the arguments `form`, `data`, and `metric`, unless `object` includes a `corSpatial` element, in which case the associated covariate (obtained with the `getCovariate` method) is used.
- form**: an optional one-sided formula specifying the covariate(s) to be used for calculating the distances between residual pairs and, optionally, a grouping factor for partitioning the residuals (which must appear to the right of a `|` operator in `form`). Default is `~1`, implying that the observation order within the groups is used to obtain the distances.
- resType**: an optional character string specifying the type of residuals to be used. If "response", the "raw" residuals (observed - fitted) are used; else, if "pearson", the standardized residuals (raw residuals divided by the corresponding standard errors) are used; else, if "normalized", the normalized residuals (standardized residuals pre-multiplied by the inverse square-root factor of the estimated error correlation matrix) are used. Partial matching of arguments is used, so only the first character needs to be provided. Defaults to "pearson".
- data**: an optional data frame in which to interpret the variables in `form`. By default, the same data used to fit `object` is used.

`na.action`: a function that indicates what should happen when the data contain NAs. The default action (`na.fail`) causes an error message to be printed and the function to terminate, if there are any incomplete observations.

`maxDist`: an optional numeric value for the maximum distance used for calculating the semi-variogram between two residuals. By default all residual pairs are included.

`length.out`: an optional integer value. When object includes a `corSpatial` element, its semi-variogram values are calculated and this argument is used as the `length.out` argument to the corresponding `Variogram` method. Defaults to 50.

`collapse`: an optional character string specifying the type of collapsing to be applied to the individual semi-variogram values. If equal to `"quantiles"`, the semi-variogram values are split according to quantiles of the distance distribution, with equal number of observations per group, with possibly varying distance interval lengths. Else, if `"fixed"`, the semi-variogram values are divided according to distance intervals of equal lengths, with possibly different number of observations per interval. Else, if `"none"`, no collapsing is used and the individual semi-variogram values are returned. Defaults to `"quantiles"`.

`nint`: an optional integer with the number of intervals to be used when collapsing the semi-variogram values. Defaults to 20.

`robust`: an optional logical value specifying if a robust semi-variogram estimator should be used when collapsing the individual values. If `TRUE` the robust estimator is used. Defaults to `FALSE`.

`breaks`: an optional numeric vector with the breakpoints for the distance intervals to be used in collapsing the semi-variogram values. If not missing, the option specified in `collapse` is ignored.

`metric`: an optional character string specifying the distance metric to be used. The currently available options are `"euclidean"` for the root sum-of-squares of distances; `"maximum"` for the maximum difference; and `"manhattan"` for the sum of the absolute differences. Partial matching of arguments is used, so only the first three characters need to be provided. Defaults to `"euclidean"`.

VALUE

a data frame with columns `variog` and `dist` representing, respectively, the semi-variogram values and the corresponding distances. If the semi-variogram values are collapsed, an extra column, `n.pairs`, with the number of residual pairs used in each semi-variogram calculation, is included in the returned data frame. If object includes a `corSpatial` element, a data frame with its corresponding semi-variogram is included in the returned value, as an attribute `"modelVariog"`. The returned value inherits from class `Variogram`.

REFERENCES

Cressie, N.A.C. (1993), "Statistics for Spatial Data", J. Wiley & Sons. Diggle, P.J., Liang, K.Y. and Zeger, S. L. (1994) "Analysis of Longitudinal Data", Oxford University Press Inc.

SEE ALSO

`lme`, `Variogram.default`, `Variogram.gls`, `plot.Variogram`

EXAMPLE

```
fml <- lme(weight ~ Time * Diet, BodyWeight, ~ Time | Rat)
Variogram(fml, form = ~ Time | Rat, nint = 10, robust = TRUE)
```

varPower	Power Variance Function	varPower
-----------------	-------------------------	-----------------

This function is a constructor for the `varPower` class, representing a power variance function structure. Letting v denote the variance covariate and $\sigma^2(v)$ denote the variance function evaluated at v , the power variance function is defined as $\sigma^2(v) = |v|^{2\theta}$, where θ is the variance function coefficient. When a grouping factor is present, a different θ is used for each factor level.

```
varPower(value, form, fixed)
```

ARGUMENTS

value: an optional numeric vector, or list of numeric values, with the variance function coefficients. **value** must have length one, unless a grouping factor is specified in **form**. If **value** has length greater than one, it must have names which identify its elements to the levels of the grouping factor defined in **form**. If a grouping factor is present in **form** and **value** has length one, its value will be assigned to all grouping levels. Default is `numeric(0)`, which results in a vector of zeros of appropriate length being assigned to the coefficients when object is initialized (corresponding to constant variance equal to one).

form: an optional one-sided formula of the form $\sim v$, or $\sim v \mid g$, specifying a variance covariate v and, optionally, a grouping factor g for the coefficients. The variance covariate must evaluate to a numeric vector and may involve expressions using `"."`, representing a fitted model object from which fitted values (`fitted(.)`) and residuals (`resid(.)`) can be extracted (this allows the variance covariate to be updated during the optimization of an objective function). When a grouping factor is present in **form**, a different coefficient value is used for each of its levels. Several grouping variables may be simultaneously specified, separated by the `*` operator, like in $\sim v \mid g1 * g2 * g3$. In this case, the levels of each grouping variable are pasted together and the resulting factor is used to group the observations. Defaults to $\sim \text{fitted}(.)$ representing a variance covariate given by the fitted values of a fitted model object and no grouping factor.

fixed: an optional numeric vector, or list of numeric values, specifying the values at which some or all of the coefficients in the variance function should be fixed. If a grouping factor is specified in `form`, `fixed` must have names identifying which coefficients are to be fixed. Coefficients included in `fixed` are not allowed to vary during the optimization of an objective function. Defaults to `NULL`, corresponding to no fixed coefficients.

VALUE

a `varPower` object representing a power variance function structure, also inheriting from class `varFunc`.

SEE ALSO

`varWeights.varFunc`, `coef.varPower`

EXAMPLE

```
vf1 <- varPower(0.2, form = ~ age|Sex)
```

varWeights	Extract Variance Function Weights	varWeights
-------------------	-----------------------------------	-------------------

The inverse of the standard deviations corresponding to the variance function structure represented by `object` are returned.

```
varWeights(object)
```

ARGUMENTS

object: an object inheriting from class `varFunc`, representing a variance function structure.

VALUE

if `object` has a `weights` attribute, its value is returned; else `NULL` is returned.

SEE ALSO

`logLik.varFunc`

EXAMPLE

```
vf1 <- varPower(form=~ age)
vf1 <- initialize(vf1, Orthodont)
coef(vf1) <- 0.3
varWeights(vf1)[1:10]
```

varWeights.glsStruct

glsStruct Variance Weights

varWeights.glsStruct

If `object` includes a `varStruct` component, the inverse of the standard deviations of the variance function structure represented by the corresponding `varFunc` object are returned; else, a vector of ones of length equal to the number of observations in the data frame used to fit the associated linear model is returned.

```
varWeights(object)
```

ARGUMENTS

object: an object inheriting from class `glsStruct`, representing a list of linear model components, such as `corStruct` and `varFunc` objects.

VALUE

if `object` includes a `varStruct` component, a vector with the corresponding variance weights; else, or a vector of ones.

SEE ALSO

```
varWeights
```

varWeights.lmeStruct

lmeStruct Variance Weights

varWeights.lmeStruct

If `object` includes a `varStruct` component, the inverse of the standard deviations of the variance function structure represented by the corresponding `varFunc` object are returned; else, a vector of ones of length equal to the number of observations in the data frame used to fit the associated linear mixed-effects model is returned.

```
varWeights(object)
```

ARGUMENTS

object: an object inheriting from class `lmeStruct`, representing a list of linear mixed-effects model components, such as `reStruct`, `corStruct`, and `varFunc` objects.

VALUE

if `object` includes a `varStruct` component, a vector with the corresponding variance weights; else, or a vector of ones.

SEE ALSO

```
varWeights
```