Package 'VCA'

May 7, 2026

Version 1.5.2

Date 2025-09-05

Title Variance Component Analysis

Depends R (>= 3.0.0)

Imports stats, graphics, grDevices, lme4, Matrix, methods, numDeriv

Suggests VFP, STB, knitr, rmarkdown, prettydoc, RUnit

Description ANOVA and REML estimation of linear mixed models is implemented, once following Searle et al. (1991, ANOVA for unbalanced data), once making use of the 'lme4' package. The primary objective of this package is to perform a variance component analysis (VCA) according to CLSI EP05-A3 guideline ``Evaluation of Precision of Quantitative Measurement Procedures" (2014). There are plotting methods for visualization of an experimental design, plotting random effects and residuals. For ANOVA type estimation two methods for computing ANOVA mean squares are implemented (SWEEP and quadratic forms). The covariance matrix of variance components can be derived, which is used in estimating confidence intervals. Linear hypotheses of fixed effects and LS means can be computed. LS means can be computed at specific values of covariables and with custom weighting schemes for factor variables. See ?VCA for a more comprehensive description of the features.

License GPL (>= 3)

VignetteBuilder knitr, rmarkdown

RoxygenNote 7.2.3

NeedsCompilation yes

Author Andre Schuetzenmeister [aut, cre] (ORCID: <https://orcid.org/0000-0002-2964-5502>), Florian Dufey [aut] (ORCID: <https://orcid.org/0000-0001-6467-8556>)

Maintainer Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Repository CRAN

Date/Publication 2025-09-06 05:12:08 UTC

Contents

VCA-package *(V)ariance (C)omponent (A)nalysis.*

Description

This package implements ANOVA-type estimation of variance components (VC) for linear mixed models (LMM), and provides Restricted Maximum Likelihood (REML) estimation incorporating functionality of the lme4 package. For models fitted by REML the typical VCA-table is derived, also containing the variances of VC, which are approximated by the method outlined in Giesbrecht & Burns (1985). REML-estimation is available via functions remlVCA for variance component analysis (VCA) and remlMM for fitting general LMM.

ANOVA-methodology is a special method of moments approach for estimating (predicting) variance components implemented in functions anovaMM and anovaVCA. The former represents a general, unrestricted approach to fitting linear mixed models, whereas the latter is tailored for performing a VCA on random models. Experiments of this type frequently occur in performance evaluation analyses of diagnostic tests or analyzers (devices) quantifying various types of precision (see e.g. guideline EP05-A2/A3 of the Clinical and Laboratory Standards Institute - CLSI).

The general Satterthwaite approximation of denominator degrees of freedom for tests of fixed effects (test.fixef) and LS Means (test.lsmeans) is implemented as used in SAS PROC MIXED. Results differ for unbalanced designs because of the different approaches to estimating the covariance matrix of variance components. Here, two algorithms are implemented for models fitted via ANOVA, 1^{st} the "exact" method described in Searle et. al (1992), 2^{nd} an approximation described in Giesbrecht & Burns (1985). The latter is also used for models fitted by REML. See test.fixef and getGB for details on this topic.

Furthermore, the Satterthwaite approximation of degrees of freedom for individual VCs and total variance is implemented. These are employed in Chi-Squared tests of estimated variances against a claimed value (total, error), as well as in Chi-Squared based confidence intervals (CI) (see VCAinference). Whenever ANOVA-type estimated VCs become negative, the default is to set them equal to 0. ANOVA mean squares used within the Satterthwaite approximation will be adapted to this situation by re-computing ANOVA mean squares (s_{MS}) as $s_{MS} = C * \sigma^2$, where C is a coefficient matrix and a function of the design matrix and σ^2 is the column-vector of adapted variance components. Total variance corresponds to a conservative estimate of the total variability in these cases, i.e. it will be larger than e.g. the total variance of the same model fitted by REML, because the negative VC will not contribute to total variance. See the documentation anovaVCA and anovaMM for details, specifically argument NegVC.

Additionally to fitting linear mixed models and performing VCA-analyses, various plotting methods are implemented, e.g. a variability chart visualizing the variability in sub-classes emerging from an experimental design (varPlot). Random effects and residuals can be transformed and plotted using function plotRandVar. Standardization and studentization are generally available, Pearson-type transformation is only available for residuals. Plotting (studentized) random variates of a LMM should always be done to reveal potential problems of the fitted model, e.g. violation of model assumptions and/or whether there are outlying observations.

There are not any more two approaches to estimating ANOVA sums (SSQ) of squares as in previous package-versions. Now, only a fast FORTRAN-routine is used generating the column vector of

SSQ, coefficient matrix C (previously computed using the Abbreviated Doolittle and Square Root Method), and the covariance matrix of VC. Overall, this leads to a dramatic reduction of computation time for models fitted using ANOVA.

Further reduction of the computation time can be achieved using Intel's `Math Kernel Library (MKL)`. When the package is loaded it will be automatically checked whether this is the case or not.

In LS Means computation of fitted LMM it is possible to compute LS Means using specific values of covariables, which is equivalent to using option 'AT' in the 'lsmeans'-statement of SAS PROC MIXED. It is also possible to apply other than the default weighting scheme for (fixed) factor-variables. See the details section in `lsmeans` and the description of argument `at`.

Note: The 'UnitTests' directory within the package-directory contains a pre-defined test-suite which can be run by sourcing 'RunAllTests.R' for user side testing (installation verification). It requires the 'RUnit' package and checks the numerical equivalence to reference results (SAS PROC MIXED method=type1/reml, SAS PROC VARCOMP) for balanced and unbalanced data and different experimental designs.

Details

Package:	VCA
Type:	Package
Version:	1.5.2
Date:	2025-09-05
License:	GPL (>=3)
LazyLoad:	yes

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>, Florian Dufey <florian.dufey@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Goodnight, J.H. (1979), A Tutorial on the SWEEP Operator, The American Statistician, 33:3, 149-158

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

Satterthwaite, F.E. (1946), An Approximate Distribution of Estimates of Variance Components., Biometrics Bulletin 2, 110-114

Gaylor,D.W., Lucas,H.L., Anderson,R.L. (1970), Calculation of Expected Mean Squares by the Abbreviated Doolittle and Square Root Methods., Biometrics 26 (4): 641-655

SAS Help and Documentation PROC MIXED, SAS Institute Inc., Cary, NC, USA

anovaMM *ANOVA-Type Estimation of Mixed Models*

Description

Estimate/Predict random effects employing ANOVA-type estimation and obtain generalized least squares estimates of fixed effects for any linear mixed model including random models and linear models.

Usage

```
anovaMM(
  form,
  Data,
  by = NULL,
  VarVC.method = c("scm", "gb"),
  NegVC = FALSE,
  quiet = FALSE,
  order.data = TRUE
)
```

Arguments

form	(formula) specifying the linear mixed model (fixed and random part of the model), all random terms need to be enclosed by round brackets. Any variable not being bracketed will be considered as fixed. Interaction terms containing at least one random factor will automatically be random (Piepho et al. 2003). All terms appearing in the model (fixed or random) need to be compliant with the regular expression "^[^[\.]]?[[:alnum:]_\.]*$", i.e. they may not start with a dot and may then only consist of alpha-numeric characters, dot and underscore. Otherwise, an error will be issued.
Data	(data.frame) containing all variables referenced in 'form', note that variables can only be of type "numeric", "factor" or "character". The latter will be automatically converted to "factor".
by	(factor, character) variable specifying groups for which the analysis should be performed individually, i.e. by-processing
VarVC.method	(character) string specifying whether to use the algorithm given in Searle et al. (1992) which corresponds to `VarVC.method="scm"` or in Giesbrecht and Burns (1985) which can be specified via "gb". Method "scm" (Searle, Casella, McCulloch) is the exact algorithm, "gb" (Giesbrecht, Burns) is termed "rough approximation" by the authors, but sufficiently exact compared to e.g. SAS PROC MIXED (method=type1) which uses the inverse of the Fisher-Information matrix as approximation. For balanced designs all methods give identical results, only in unbalanced designs differences occur.

NegVC	(logical) FALSE = negative variance component estimates (VC) will be set to 0 and they will not contribute to the total variance (as done e.g. in SAS PROC NESTED, conservative estimate of total variance). The original ANOVA estimates can be found in element 'VCoriginal'. The degrees of freedom of the total variance are based on adapted mean squares (MS) (see details). TRUE = negative variance component estimates will not be set to 0 and they will contribute to the total variance (original definition of the total variance).
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise
order.data	(logical) TRUE = class-variables will be ordered increasingly, FALSE = ordering of class-variables will remain as is

Details

A Linear Mixed Model, noted in standard matrix notation, can be written as $y = Xb+Zg+e$, where y is the column vector of observations, X and Z are design matrices assigning fixed (b), respectively, random (g) effects to observations, and e is the column vector of residual errors. Whenever there is an intercept in the model, i.e. the substring "-1" is not part of the model formula, the same restriction as in SAS PROC MIXED is introduced setting the last fixed effect equal to zero. Note, that the results of an linear contrasts are not affected by using an intercept or not, except that constrained fixed effects cannot be part of such contrasts (one could use the intercept estimated instead).

Here, no further restrictions on the type of model are made. One can fit mixed models as well as random models, which constitute a sub-set of mixed models (intercept being the only fixed effect). Variables must be either of type "numeric" or "factor". "character" variables are automatically converted to factors and the response variable has to be numeric, of course. In case that 'class(Data[,i])' is neither one of these three options, an error is issued. Even simple linear models can be fitted, i.e. models without a random part (without Zg) besides the residual errors. In this case, an Analysis of Variance (ANOVA) table is computed in the same way as done by function 'anova.lm'.

One drawback of using ANOVA-type estimation of random effects is, that random effects are independent, i.e they have zero covariance by definition $cov(g_i, g_j) = 0$. Another one is that estimated variance components may become negative under certain conditions. The latter situation is addressed by setting negative variance estimates equal to zero and adapting ANOVA mean squares (MS) as $MS = C * VC$, where C is a coefficient matrix and a function of the design matrix $[XZ]$ and VC is the column-vector of adapted variance components. The Satterthwaite approximation of total degrees of freedom (DF for total variance) will use adapted MS-values.

Note, that setting negative VCs equal to zero results in a conservative estimate of the total variance, i.e. it will be larger than the estimate including the negative VC(s). Use parameter 'NegVC=TRUE' to explicitly allow negative variance estimates.

For further details on ANOVA Type-I estimation methods see anovaVCA.

Value

(VCA) object

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Goodnight, J.H. (1979), A Tutorial on the SWEEP Operator, The American Statistician, 33:3, 149-158

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

H.P.Piepho, A.Buechse and K.Emrich (2003), A Hitchhiker's Guide to Mixed Models for Randomized Experiments, J.Agronomy & Crop Science 189, p. 310-322

Gaylor,D.W., Lucas,H.L., Anderson,R.L. (1970), Calculation of Expected Mean Squares by the Abbreviated Doolittle and Square Root Methods., Biometrics 26 (4): 641-655

SAS Help and Documentation PROC MIXED, SAS Institute Inc., Cary, NC, USA

See Also

anovaVCA

anovaVCA, VCAinference, remlVCA, remlMM ranef, fixef, vcov, vcovVC, test.fixef, test.lsmeans, plotRandVar

Examples

```
## Not run:

data(dataEP05A2_2)

# assuming 'day' as fixed, 'run' as random
anovaMM(y~day/(run), dataEP05A2_2)

# assuming both as random leads to same results as
# calling anovaVCA
anovaMM(y~(day)/(run), dataEP05A2_2)
anovaVCA(y~day/run, dataEP05A2_2)

# use different approaches to estimating the covariance of
# variance components (covariance parameters)
dat.ub <- dataEP05A2_2[-c(11,12,23,32,40,41,42),] # get unbalanced data
m1.ub <- anovaMM(y~day/(run), dat.ub, VarVC.method="scm")
m2.ub <- anovaMM(y~day/(run), dat.ub, VarVC.method="gb")
V1.ub <- round(vcovVC(m1.ub), 12)
V2.ub <- round(vcovVC(m2.ub), 12)
all(V1.ub == V2.ub)

# fit a larger random model
data(VCAdata1)
fitMM1 <- anovaMM(y~((lot)+(device))/(day)/(run), VCAdata1[VCAdata1$sample==1,])
fitMM1
# now use function tailored for random models
fitRM1 <- anovaVCA(y~(lot+device)/day/run, VCAdata1[VCAdata1$sample==1,])
fitRM1
```

```
# there are only 3 lots, take 'lot' as fixed
fitMM2 <- anovaMM(y~(lot+(device))/(day)/(run), VCAdata1[VCAdata1$sample==2,])

# the following model definition is equivalent to the one above,
# since a single random term in an interaction makes the interaction
# random (see the 3rd reference for details on this topic)
fitMM3 <- anovaMM(y~(lot+(device))/day/run, VCAdata1[VCAdata1$sample==2,])

# fit same model for each sample using by-processing
lst <- anovaMM(y~(lot+(device))/day/run, VCAdata1, by="sample")
lst

# fit mixed model originally from 'nlme' package

library(nlme)
data(Orthodont)
fit.lme <- lme(distance~Sex*I(age-11), random=~I(age-11)|Subject, Orthodont)

# re-organize data for using 'anovaMM'
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))
fit.anovaMM1 <- anovaMM(distance~Sex*age2+(Subject)*age2, Ortho)

# use simplified formula avoiding unnecessary terms
fit.anovaMM2 <- anovaMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2, Ortho)

# and exclude intercept
fit.anovaMM3 <- anovaMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)

# compare results
fit.lme
fit.anovaMM1
fit.anovaMM2
fit.anovaMM3

# are there a sex-specific differences?
cmat <- getL(fit.anovaMM3, c("SexMale-SexFemale", "SexMale:age2-SexFemale:age2"))
cmat

test.fixef(fit.anovaMM3, L=cmat)

# former versions of the package used R-function 'lm' and 'anova',
# which is significantly slower for sufficiently large/complex models
data(realData)
datP1 <- realData[realData$PID==1,]
system.time(anova.lm.Tab <- anova(lm(y~lot/calibration/day/run, datP1)))
# Using the sweeping approach for estimating ANOVA Type-1 sums of squares
# this is now the default setting.
system.time(anovaMM.Tab1  <- anovaMM(y~lot/calibration/day/run, datP1))

# compare results, note that the latter corresponds to a linear model,
# i.e. without random effects. Various matrices have already been computed,
```

```
# e.g. "R", "V" (which are identical in this case).
anova.lm.Tab
anovaMM.Tab1

## End(Not run)
```

---

anovaVCA *ANOVA-Type Estimation of Variance Components for Random Models*

Description

This function equates observed ANOVA Type-I sums of squares (SS) to their expected values and solves the resulting system of linear equations for variance components.

Usage

```
anovaVCA(
  form,
  Data,
  by = NULL,
  NegVC = FALSE,
  VarVC.method = c("scm", "gb"),
  MME = FALSE,
  quiet = FALSE,
  order.data = TRUE
)
```

Arguments

form	(formula) specifying the model to be fit, a response variable left of the '~' is mandatory
Data	(data.frame) containing all variables referenced in 'form'
by	(factor, character) variable specifying groups for which the analysis should be performed individually, i.e. by-processing
NegVC	(logical) FALSE = negative variance component estimates (VC) will be set to 0 and they will not contribute to the total variance (as done in SAS PROC NESTED, conservative estimate of total variance). The original ANOVA estimates can be found in element 'VCoriginal'. The degrees of freedom of the total variance are based on adapted mean squares (MS), i.e. adapted MS are computed as $D * VC$, where VC is the column vector with negative VCs set to 0. TRUE = negative variance component estimates will not be set to 0 and they will contribute to the total variance (original definition of the total variance).

VarVC.method	(character) string specifying whether to use the algorithm given in Searle et al. (1992) which corresponds to VarVC.method="scm" or in Giesbrecht and Burns (1985) which can be specified via "gb". Method "scm" (Searle, Casella, McCulloch) is the exact algorithm, "gb" (Giesbrecht, Burns) is termed "rough approximation" by the authors, but sufficiently exact compared to e.g. SAS PROC MIXED (method=type1) which uses the inverse of the Fisher-Information matrix as approximation. For balanced designs all methods give identical results, only in unbalanced designs differences occur.
MME	(logical) TRUE = (M)ixed (M)odel (E)quations will be solved, i.e. 'VCA' object will have additional elements "RandomEffects", "FixedEffects", "VarFixed" (variance-covariance matrix of fixed effects) and the "Matrices" element has addional elements corresponding to intermediate results of solving MMEs. FALSE = do not solve MMEs, which reduces the computation time for very complex models significantly.
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise
order.data	(logical) TRUE = class-variables will be ordered increasingly, FALSE = ordering of class-variables will remain as is

Details

For diagnostics, a key parameter is "precision", i.e. the accuracy of a quantification method influenced by varying sources of random error. This type of experiments is requested by regulatory authorities to proof the quality of diagnostic tests, e.g. quantifying intermediate precision according to CLSI guideline EP5-A2/A3. No, fixed effects are allowed besides the intercept. Whenever fixed effects are part of the model to be analyzed, use function anovaMM instead.

Function anovaVCA is tailored for performing Variance Component Analyses (VCA) for random models, assuming all VCs as factor variables, i.e. their levels correspond to distinct columns in the design matrix (dummy variables). Any predictor variables are automatically converted to factor variables, since continuous variables may not be used on the right side of the formula 'form'.

ANOVA SS are computed employing the SWEEP-operator (Goodnight 1979, default). according to Searle et al. (1992) which corresponds to VarVC.method="scm".

Function anovaVCA represents a special form of the "method of moments" approach applicable to arbitrary random models either balanced or unbalanced. The system of linear equations, which is built from the ANOVA Type-I sums of squares, is closely related to the method used by SAS PROC VARCOMP, where ANOVA mean squares (MS) are used. The former can be written as $ss = C * s$ and the latter as $ms = D * s$, where C and D denote the respective coefficient matrices, s the column-vector of variance components (VC) to be estimated/predicted, and ss and ms the column vector of ANOVA sum of squares, respectively, mean squares. Mutliplying element d_ij of matrix D by element c_in of matrix C $(i, j = 1, ..., n)$, results in matrix C. Thus, C can easily be converted to D by the inverse operation. Matrix D is used to estimate total degrees of freedom (DF) according to Satterthwaite (1946).

The method for computing ANOVA Type-I SS is much faster than fitting the linear model via lm and calling function anova on the 'lm' object for complex models, where complex refers to the number of columns of the design matrix and the degree of unbalancedness. DF are directly derived from the SWEEP-operator as the number of linearly independent columns of the partial design matrix corresponding to a specific VC.

Value

(object) of class 'VCA'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Goodnight, J.H. (1979), A Tutorial on the SWEEP Operator, The American Statistician, 33:3, 149-158

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

Satterthwaite, F.E. (1946), An Approximate Distribution of Estimates of Variance Components., Biometrics Bulletin 2, 110-114

Gaylor,D.W., Lucas,H.L., Anderson,R.L. (1970), Calculation of Expected Mean Squares by the Abbreviated Doolittle and Square Root Methods., Biometrics 26 (4): 641-655

SAS Help and Documentation PROC MIXED, SAS Institute Inc., Cary, NC, USA

SAS Help and Documentation PROC VARCOMP, SAS Institute Inc., Cary, NC, USA

See Also

anovaMM, remlVCA, remlMM, print.VCA, VCAinference, ranef, plotRandVar, stepwiseVCA

Examples

```
## Not run:

# load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_2)

# perform ANOVA-estimation of variance components
res <- anovaVCA(y~day/run, dataEP05A2_2)
res

# design with two main effects (ignoring the hierarchical structure of the design)
anovaVCA(y~day+run, dataEP05A2_2)

# compute confidence intervals, perform F- and Chi-Squared tests
INF <- VCAinference(res, total.claim=3.5, error.claim=2)
INF

### load data from package
data(VCAdata1)

data_sample1 <- VCAdata1[VCAdata1$sample==1,]

### plot data for visual inspection
```

```
varPlot(y~lot/day/run, data_sample1)

### estimate VCs for 4-level hierarchical design (error counted) for sample_1 data
anovaVCA(y~lot/day/run, data_sample1)

### using different model (ignoring the hierarchical structure of the design)
anovaVCA(y~lot+day+lot:day:run, data_sample1)

### same model with unbalanced data
anovaVCA(y~lot+day+lot:day:run, data_sample1[-c(1,11,15),])

### use the numerical example from the CLSI EP05-A2 guideline (p.25)
data(Glucose,package="VCA")
res.ex <- anovaVCA(result~day/run, Glucose)

### also perform Chi-Squared tests
### Note: in guideline claimed SD-values are used, here, claimed variances are used
VCAinference(res.ex, total.claim=3.4^2, error.claim=2.5^2)

### now use the six sample reproducibility data from CLSI EP5-A3
### and fit per sample reproducibility model
data(CA19_9)
fit.all <- anovaVCA(result~site/day, CA19_9, by="sample")

reproMat <- data.frame(
Sample=c("P1", "P2", "Q3", "Q4", "P5", "Q6"),
Mean= c(fit.all[[1]]$Mean, fit.all[[2]]$Mean, fit.all[[3]]$Mean,
fit.all[[4]]$Mean, fit.all[[5]]$Mean, fit.all[[6]]$Mean),
Rep_SD=c(fit.all[[1]]$aov.tab["error","SD"], fit.all[[2]]$aov.tab["error","SD"],
fit.all[[3]]$aov.tab["error","SD"], fit.all[[4]]$aov.tab["error","SD"],
fit.all[[5]]$aov.tab["error","SD"], fit.all[[6]]$aov.tab["error","SD"]),
Rep_CV=c(fit.all[[1]]$aov.tab["error","CV[%]"],fit.all[[2]]$aov.tab["error","CV[%]"],
fit.all[[3]]$aov.tab["error","CV[%]"],fit.all[[4]]$aov.tab["error","CV[%]"],
fit.all[[5]]$aov.tab["error","CV[%]"],fit.all[[6]]$aov.tab["error","CV[%]"]),
WLP_SD=c(sqrt(sum(fit.all[[1]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[2]]$aov.tab[3:4, "VC"])),
sqrt(sum(fit.all[[3]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[4]]$aov.tab[3:4, "VC"])),
sqrt(sum(fit.all[[5]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[6]]$aov.tab[3:4, "VC"]))),
WLP_CV=c(sqrt(sum(fit.all[[1]]$aov.tab[3:4,"VC"]))/fit.all[[1]]$Mean*100,
sqrt(sum(fit.all[[2]]$aov.tab[3:4,"VC"]))/fit.all[[2]]$Mean*100,
sqrt(sum(fit.all[[3]]$aov.tab[3:4,"VC"]))/fit.all[[3]]$Mean*100,
sqrt(sum(fit.all[[4]]$aov.tab[3:4,"VC"]))/fit.all[[4]]$Mean*100,
sqrt(sum(fit.all[[5]]$aov.tab[3:4,"VC"]))/fit.all[[5]]$Mean*100,
sqrt(sum(fit.all[[6]]$aov.tab[3:4,"VC"]))/fit.all[[6]]$Mean*100),
Repro_SD=c(fit.all[[1]]$aov.tab["total","SD"],fit.all[[2]]$aov.tab["total","SD"],
fit.all[[3]]$aov.tab["total","SD"],fit.all[[4]]$aov.tab["total","SD"],
fit.all[[5]]$aov.tab["total","SD"],fit.all[[6]]$aov.tab["total","SD"]),
Repro_CV=c(fit.all[[1]]$aov.tab["total","CV[%]"],fit.all[[2]]$aov.tab["total","CV[%]"],
fit.all[[3]]$aov.tab["total","CV[%]"],fit.all[[4]]$aov.tab["total","CV[%]"],
fit.all[[5]]$aov.tab["total","CV[%]"],fit.all[[6]]$aov.tab["total","CV[%]"]))

for(i in 3:8) reproMat[,i] <- round(reproMat[,i],digits=ifelse(i%%2==0,1,3))
reproMat
```

```
# now plot the precision profile over all samples
plot(reproMat[,"Mean"], reproMat[,"Rep_CV"], type="l", main="Precision Profile CA19-9",
xlab="Mean CA19-9 Value", ylab="CV[%]")
grid()
points(reproMat[,"Mean"], reproMat[,"Rep_CV"], pch=16)

# load another example dataset and extract the "sample==1" subset
data(VCAdata1)
sample1 <- VCAdata1[which(VCAdata1$sample==1),]

# generate an additional factor variable and random errors according to its levels
sample1$device <- gl(3,28,252)
set.seed(505)
sample1$y <- sample1$y + rep(rep(rnorm(3,,.25), c(28,28,28)),3)

# fit a crossed-nested design with main factors 'lot' and 'device'
# and nested factors 'day' and 'run' nested below
res1 <- anovaVCA(y~(lot+device)/day/run, sample1)
res1

# fit same model for each sample using by-processing
lst <- anovaVCA(y~(lot+device)/day/run, VCAdata1, by="sample")
lst

# now fitting a nonsense model on the complete dataset "VCAdata1"
# where the SWEEP-operator is the new default since package version 1.2
# takes ~5s
system.time(res.sw <- anovaVCA(y~(sample+lot+device)/day/run, VCAdata1))
# applying functions 'anova' and 'lm' in the same manner takes ~ 265s
system.time(res.lm <- anova(lm(y~(sample+lot+device)/day/run, VCAdata1)))
res.sw
res.lm

## End(Not run)
```

`as.matrix.VCA` *Standard 'as.matrix' Method for 'VCA' S3-Objects*

Description

Standard 'as.matrix' Method for 'VCA' S3-Objects

Usage

```
## S3 method for class 'VCA'
as.matrix(x, ...)
```

Arguments

`x`	(VCA) object
`...`	additional arguments to be passed to or from methods.

Value

(matrix) equal to x$aov.tab with additional attributes "Mean" and "Nobs"

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

as.matrix.VCAinference

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
as.matrix(fit)

## End(Not run)
```

---

as.matrix.VCAinference

Standard 'as.matrix' Method for 'VCAinference' S3-Objects

Description

This function makes use of the hidden feature of function print.VCAinference which invisibly returns character matrices of estimated variance components expressed as "VC" (variance component), "SD" (standard deviation) or "CV" (coefficient of variation). If argument "what" is not specified, a named list will be returned with all three matrices.

Usage

```
## S3 method for class 'VCAinference'
as.matrix(x, what = c("VC", "SD", "CV"), digits = 6, ...)
```

Arguments

x	(VCAinference) object
what	(character) one or multiple choices from "VC" (variance component), "SD" (standard deviation) or "CV" (coefficient of variation)
digits	(integer) number of decimal digits to be used
...	additional arguments to be passed to or from methods.

Value

(matrix) with point estimates, one- and two-sided confidence intervals and variances of the estimated variance components

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

print.VCAinference, as.matrix.VCA

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
inf <- VCAinference(fit, VarVC=TRUE)
as.matrix(inf, what="VC", digits=6)
as.matrix(inf, what="SD", digits=6)
as.matrix(inf, what="CV", digits=2)

# request list of matrices
as.matrix(inf)

## End(Not run)
```

---

buildList *Build a Nested List.*

Description

Function buildList creates a nested-list reflecting the hierarchical structure of a fully-nested model, respectively, the imputed hierarchical structure of the data (see details).

Usage

```
buildList(
  Data,
  Nesting,
  Current,
  resp,
  keep.order = TRUE,
  useVarNam = TRUE,
  sep = "",
  na.rm = TRUE,
  Points = list(pch = 16, cex = 0.5, col = "black")
)
```

Arguments

Data	(data.frame) with the data
Nesting	(character) vector specifying the nesting structure with the top-level variable name as 1st element and the variance component one above the residual error as last element
Current	(character) string specifying the current level which has to be processed
resp	(character) string specifying the name of the response variable (column in 'Data')
keep.order	(logical) TRUE = the ordering of factor-levels is kept as provided by 'Data', FALSE = factor-levels are sorted on and within each level of nesting
useVarNam	(logical) TRUE = each factor-level specifier is pasted to the variable name of the current variable and used as list-element name, FALSE = factor-level specifiers are used as names of list-elements; the former is useful when factor levels are indicated as integers, e.g. days as 1,2,..., the latter is useful when factor levels are already unique, e.g. day1, day2, ...
sep	(character) string specifying the separator-string in case useVarNam=TRUE
na.rm	(logical) TRUE = NAs will be removed before computing the descriptive statistics AND NAs will be omitted when counting number of elements, FALSE = if there are NAs, this will result in NAs for the descriptive statistics
Points	(list) specifying all parameters applicable to function 'points', used to specify scatterplots per lower-end factor-level (e.g. run/part in EP05-A2 experiments). If list-element "col" is itself a list with elements "var" and "col", where the former specifies a variable used for assigning colors "col" according to the class-level of "var", point-colors can be used for indicating specific sub-classes not addressed by the model/design (see examples).

Details

This function is not intended to be used directly and serves as helper function for varPlot. Each factor-level, on each level of nesting is accompanied with a set of descriptive statistics, such as mean, median, var, sd, ... which can be evaluated later on. These information are used in function varPlot, which implements a variability chart. Note, that this function is also used if data does not correspond to a fully-nested design, i.e. the hierarchical structure is inferred from the model formula. The order of main factors (not nested within other factors) appearing in the model formula determines the nesting structure imputed in order to plot the data as variability chart.

Value

(list) which was recursively built, representing the data of the fully-nested as hierarchy

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
# load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_3)

# build a list representing the hierarichal structure of a fully-nested model
# there needs to be a distinct hierarchy for being able to plot the data
# as variability chart (this function is not exported)
lst <- VCA:::buildList(Data=dataEP05A2_3, Nesting=c("day", "run"), Current="day", resp="y")

## End(Not run)
```

CA19_9 *Reproducibility Example Dataset from CLSI EP05-A3*

Description

This data set consists of the example data of a complete three-site, multisample reproducibility study as presented in the CLSI EP5-A3 guideline. It shows quantitative results of an automated immunometric assay measuring parameter CA19-9. This dataset is described in Appendix B of this guideline consisting of 6 samples, each measured on one of three sites, at five days with five replicates per day.

Usage

```
data(CA19_9)
```

Format

data.frame with 450 rows and 4 variables.

References

CLSI EP05-A3 - Evaluation of Precision of Quantitative Measurement Procedures; Approved Guideline - Third Edition. CLSI

---

check4MKL *Check for Availability of Intel's Math Kernel Library*

Description

Majority of the code is borrowed from the Microsoft R Open Rprofile.site file. In case MKL can be detected this information will be stored in a separate envrionment, which other function know about. If so, an optimized version of function `getGB` will be used which used ordinary matrix-objects instead of matrices defined by the `Matrix`-package. This seems to accelerate computation time for large datasets by up to factor 30.

Usage

```
check4MKL()
```

Details

This function is for internal use only and therefore not exported.

Value

variable 'MKL' in envir "msgEnv" will be set to TRUE/FALSE

Author(s)

Authors of the Rprofile.site file in Microsoft R Open, Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

---

checkData *Check Random Model for Given Dataset.*

Description

This function is intended to check a variance component analysis either before or after performing it. This is particularily important for less experienced users who my not exactly know where error messages come from. External software using functions anovaVCA or remlVCA also via function fitVCA may also benefit from more user-friendly error messages.

Usage

```
checkData(form, Data)
```

Arguments

form	(formula) describing the model to be analyzed
Data	(data.frame) with all variables used in 'form'

Value

(list) of length equal to the number of terms in 'form' each element being a list of messages with identified, potential problems.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(dataEP05A2_1)
dat0 <- dataEP05A2_1[1:16,]
# everything should be ok
checkData(y~day/run, dat0)
# data identical response for all obs
dat1 <- dat0
dat1$y <- dat1[1,"y"]
remlVCA(y~day/run, dat1)
checkData(y~day/run, dat1)
# now factor-levels have identical values
dat2 <- dat0
dat2$y <- dat2$y[rep(seq(1,7,2), rep(2,4))]
checkData(y~day/run, dat2)
remlVCA(y~day/run, dat2, quiet=TRUE)
# indistinguishable terms are also problematic
dat3 <- data.frame( y=rnorm(8,10),
day=paste("day",rep(c(1,2),c(4,4))),
run=rep(c(2,1), c(4,4)))
checkData(y~day/run, dat3)
anovaVCA(y~day/run, dat3)
# no replicates, thus, no error variability
dat4 <- dat0[seq(1,15,2),]
dat4$day <- factor(dat4$day)
dat4$run <- factor(dat4$run)
checkData(y~day/run, dat4)
remlVCA(y~day/run, dat4)

## End(Not run)
```

---

checkVars *Check Tow Formula Terms for Potential Problems.*

Description

Is is checked whether 'term2' is different from 'term1' in adding information to the model. If both are main factors, i.e. no interactions terms, it is checked whether levels of 'term2' differ from those of 'term1'. Otherwise, 'term2' is an interaction with a part being different from 'term1'.

Usage

```
checkVars(Data, term1, term2)
```

Arguments

Data	(data.frame) containing all variables of 'term1' and 'term2'
term1	(character) term of a model formula as returned by 'attr(terms(form), \"term.labels\")'
term2	(character) 2nd term of a model formula as returned by 'attr(terms(form), \"term.labels\")' to check against 'term1'

Value

(list) with components \"Diff\"=part of 'term2' distinguishing it from 'term1', \"AddInfo\"=message informing about potential problems with 'term2'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

---

chol2invData *Dataset Generating Error in Function 'chol2inv'*

Description

This dataset was added because it generates an error in function 'chol2inv' when trying to invert the variance-covariance matrix 'V' of the mixed model 'value~ID+(Site)'. This dataset and the associated mixed model are part of the unit-test collection of the package.

Usage

```
data(chol2invData)
```

Format

A data frame with 158 observations on the following 3 variables.

- value

  The response variable.

- ID

  Variable with 6 levels corresponding to samples.

- Site

  Variable with 3 levels corresponding to sites/devices.

coef.VCA *Extract Fixed Effects from 'VCA' Object*

Description

For conveniently using objects of class 'VCA' with other packages expecting this function, e.g. the 'multcomp' package for general linear hypotheses for parametric models (currently not fully implemented).

Usage

```
## S3 method for class 'VCA'
coef(object, quiet = FALSE, ...)
```

Arguments

object	(VCA) object where fixed effects shall be extracted
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise
...	additional parameters

Examples

```
## Not run:
data(dataEP05A2_1)
fit1 <- anovaMM(y~day/(run), dataEP05A2_1)
coef(fit1)
fit2 <- anovaVCA(y~day/run, dataEP05A2_1)
coef(fit2)

## End(Not run)
```

combineVC *Combine Variance Components and Adjust Degrees of Freedom.*

Description

Combining variance estimates of multiple terms, mostly, effects of a full model containing a single source of variability. One example would be a precision experiment where three reagentlots are measured on one instrument in parallel, i.e. on the same days. As days are not independent of each other one needs the mean effect 'day' but as there is also interaction between reagentlot and day an interaction term 'lot:day' is needed. As the interaction term cannot be interpreted in a straight forward manner, effects 'day' and 'lot:day' need to be combined accounting for day-to-day variability.

Usage

```
combineVC(obj, comb = NULL)
```

Arguments

obj	(object) of class 'VCA', either fitted by ANOVA or REML
comb	(character) strings refering to variance components the shall be combined

Value

(object) of class 'VCA' with an updated 'aov.tab' element

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
# assume interactions lot:day
set.seed(323)
dat <- data.frame(
 lot = rep(1:3, c(25, 25, 25)),
 day = rep(rep(1:5, rep(5,5)), 3),
  res = 25)
# mean value of measurements is 25
# add variance component 'lot' (5% CV)
dat$res <- dat$res + rep(rnorm(3,,1.25), c(25, 25, 25))
# add variance component 'day' (4% CV)
dat$res <- dat$res + rep(rep(rnorm(5,,1), rep(5,5)), 3)
# add interaction variance 'lot:day' (4% CV)
dat$res <- dat$res + rep(rnorm(15,,1), rep(5, 15))
# add repeatability variance(2% CV)
dat$res <- dat$res + rnorm(75,,0.5)

# data analysis
# 3 lots were measured in parallel on the same 5 days
# using main effects 'lot' and 'day' shifts interaction
# variability 'lot:day' into 'error'
fitW <- anovaVCA(res~lot+day, dat)
fitW
# use saturated model and combine variance components
# 'day' and 'lot:day'
fit0 <- anovaVCA(res~lot*day, dat)
fit1 <- VCA:::combineVC(fit0, c("day", "lot:day"))
fit0
fit1
fitW
```

dataEP05A2_1 *Simulated Data of a CLSI EP05-A2 20/2/2 Experiment*

Description

This data set consists of simulated measurements for an experiment conducted to evaluate the precision performance of measurement methods. On 20 days two separate runs with two replicates of the same sample are measured. Thus, factor 'day' is the top-level random factor (variance component), factor 'run' is nested within 'day'.

Usage

```
data(dataEP05A2_1)
```

Format

data.frame with 80 rows and 3 variables.

References

CLSI EP05-A2 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataEP05A2_2 *Simulated Data of a CLSI EP05-A2 20/2/2 Experiment*

Description

This data set consists of simulated measurements for an experiment conducted to evaluate the precision performance of measurement methods. On 20 days two separate runs with two replicates of the same sample are measured. Thus, factor 'day' is the top-level random factor (variance component), factor 'run' is nested within 'day'.

Usage

```
data(dataEP05A2_2)
```

Format

data.frame with 80 rows and 3 variables.

References

CLSI EP05-A2 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataEP05A2_3 *Simulated Data of a CLSI EP05-A2 20/2/2 Experiment*

Description

This data set consists of simulated measurements for an experiment conducted to evaluate the precision performance of measurement methods. On 20 days two separate runs with two replicates of the same sample are measured. Thus, factor 'day' is the top-level random factor (variance component), factor 'run' is nested within 'day'.

Usage

```
data(dataEP05A2_3)
```

Format

data.frame with 80 rows and 3 variables.

References

CLSI EP05-A2 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataEP05A3_MS_1 *Simulated Data of a CLSI EP05-A3 3/5/5 Multi-Site Experiment*

Description

This data set consists of simulated measurements for an experiment to be conducted for evaluation of the precision performance of measurement methods. On 3 sites, on 5 days 5 replicates of the same sample are measured. Thus, factor 'site' is the top-level random factor (variance component), factor 'day' is nested within 'site'.

Usage

```
data(dataEP05A3_MS_1)
```

Format

data.frame with 75 rows and 3 variables.

References

Draft of CLSI EP05-A3 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataEP05A3_MS_2 *Simulated Data of a CLSI EP05-A3 3/5/5 Multi-Site Experiment*

Description

This data set consists of simulated measurements for an experiment to be conducted for evaluation of the precision performance of measurement methods. On 3 sites, on 5 days 5 replicates of the same sample are measured. Thus, factor 'site' is the top-level random factor (variance component), factor 'day' is nested within 'site'.

Usage

```
data(dataEP05A3_MS_2)
```

Format

data.frame with 75 rows and 3 variables.

References

Draft of CLSI EP05-A3 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataEP05A3_MS_3 *Simulated Data of a CLSI EP05-A3 3/5/5 Multi-Site Experiment*

Description

This data set consists of simulated measurements for an experiment to be conducted for evaluation of the precision performance of measurement methods. On 3 sites, on 5 days 5 replicates of the same sample are measured. Thus, factor 'site' is the top-level random factor (variance component), factor 'day' is nested within 'site'.

Usage

```
data(dataEP05A3_MS_3)
```

Format

data.frame with 75 rows and 3 variables.

References

Draft of CLSI EP05-A3 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

dataRS0003_1 *Simulated Repeated Measurements Data.*

Description

This data set consists of 21 measurements of the same sample, suitable to quantify the mesurement error on the same device.

Usage

```
data(dataRS0003_1)
```

Format

data.frame with 21 rows and 1 variable.

dataRS0003_2 *Simulated Repeated Measurements Data.*

Description

This data set consists of 21 measurements of the same sample, suitable to quantify the mesurement error on the same device.

Usage

```
data(dataRS0003_2)
```

Format

data.frame with 21 rows and 1 variable.

dataRS0003_3 *Simulated Repeated Measurements Data.*

Description

This data set consists of 21 measurements of the same sample, suitable to quantify the mesurement error on the same device.

Usage

```
data(dataRS0003_3)
```

Format

data.frame with 21 rows and 1 variable.

dataRS0005_1 *Simulated Data of 5/3 Experiment.*

Description

This data set consists of 15 measurements of the same sample, measured on 5 days with 3 measurements on each day. This small experiment is suitable to quantify between-day variability on the same device.

Usage

```
data(dataRS0005_1)
```

Format

data.frame with 15 rows and 2 variables.

dataRS0005_2 *Simulated Data of 5/3 Experiment.*

Description

This data set consists of 15 measurements of the same sample, measured on 5 days with 3 measurements on each day. This small experiment is suitable to quantify between-day variability on the same device.

Usage

```
data(dataRS0005_2)
```

Format

data.frame with 15 rows and 2 variables.

dataRS0005_3 *Simulated Data of 5/3 Experiment.*

Description

This data set consists of 15 measurements of the same sample, measured on 5 days with 3 measurements on each day. This small experiment is suitable to quantify between-day variability on the same device.

Usage

```
data(dataRS0005_3)
```

Format

data.frame with 15 rows and 2 variables.

DfSattHelper *Variance-Covariance Matrix of Fixed Effects as Function of Covariance Parameter Estimates*

Description

This is a helper function for function `test.fixef` approximating degrees of freedom for linear contrasts of fixed effect parameter estimates.

Usage

```
DfSattHelper(obj, x)
```

Arguments

obj	(VCA) object
x	(numeric) vector of covariance parameter estimates

Value

(matrix) corresponding to the variance-covariance matrix of fixed effects

errorMessage *Convert Objects to Detailed Error Message.*

Description

Function takes one or multiple objects and converts them to a single error-message. Objects can be output of functions try or checkData.

Usage

```
errorMessage(...)
```

Arguments

...	one or multiple objects separated by comma

Value

(characer) string combining information from input-objects

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(dataEP05A2_1)
dat2 <- dataEP05A2_1[1:16,]
dat2$y <- dat2$y[rep(seq(1,7,2), rep(2,4))]
errorMessage(try(1/"a"), checkData(y~day/run, dat2))

## End(Not run)
```

fitLMM *Fit Linear Mixed Model by ANOVA or REML*

Description

Function serves as interface to functions anovaMM and remlMM for fitting a linear mixed model (LMM) either by ANOVA or REML. All arguments applicable to either one of these functions can be specified (see anovaMM or remlMM for details).

Usage

```
fitLMM(
  form,
  Data,
  method = c("anova", "reml"),
  scale = TRUE,
  VarVC = TRUE,
  ...
)
```

Arguments

form	(formula) specifiying the linear mixed model, random effects need to be identified by enclosing them in round brackets, i.e. ~a/(b) will model factor 'a' as fixed and 'b' as random
Data	(data.frame) containing all variables referenced in 'form', note that variables can only be of type "numeric", "factor" or "character". The latter will be automatically converted to "factor"
method	(character) either "anova" to use ANOVA Type-I estimation of variance components or "reml" to use restricted maximum likelihood (REML) estimation of variance component
scale	(logical) TRUE = scale values of the response aiming to avoid numerical problems when numbers are either very small or very large, FALSE = use original scale
VarVC	(logical) TRUE = variance-covariance matrix of variance components will be computed, FALSE = it will not be computed
...	additional arguments to be passed to function anovaMM or function remlMM.

Details

Besides offering a convenient interface to both functions for fitting a LMM, this function also provides all elements required for standard task of fitted models, e.g. prediction, testing general linear hypotheses via R-package multcomp, etc. (see examples).

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

fitVCA, anovaMM, remlMM

Examples

```
## Not run:
data(dataEP05A2_2)

# assuming 'day' as fixed, 'run' as random
```

```
# Note: default method is "anova"
fitLMM(y~day/(run), dataEP05A2_2)

# explicitly request "reml"
fitLMM(y~day/(run), dataEP05A2_2, method="reml")

# assuming both as random leads to same results as
# calling anovaVCA (ANOVA is the default)
fitLMM(y~(day)/(run), dataEP05A2_2)
anovaVCA(y~day/run, dataEP05A2_2)

# now using REML-estimation
fitLMM(y~(day)/(run), dataEP05A2_2, "reml")
remlVCA(y~day/run, dataEP05A2_2)

# use different approaches to estimating the covariance of
# variance components (covariance parameters)
# create unbalanced data
dat.ub <- dataEP05A2_2[-c(11,12,23,32,40,41,42),]
m1.ub <- fitLMM(y~day/(run), dat.ub, VarVC.method="scm")
# VarVC.method="gb" is an approximation not relying on quadratic forms
m2.ub <- fitLMM(y~day/(run), dat.ub, VarVC.method="gb")
# REML-estimated variance components usually differ from ANOVA-estimates
# and so do the variance-covariance matrices
m3.ub <- fitLMM(y~day/(run), dat.ub, "reml", VarVC=TRUE)
V1.ub <- round(vcovVC(m1.ub), 12)
V2.ub <- round(vcovVC(m2.ub), 12)
V3.ub <- round(vcovVC(m3.ub), 12)

# fit a larger random model
data(VCAdata1)
fitMM1 <- fitLMM(y~((lot)+(device))/(day)/(run), VCAdata1[VCAdata1$sample==1,])
fitMM1
# now use function tailored for random models
fitRM1 <- anovaVCA(y~(lot+device)/day/run, VCAdata1[VCAdata1$sample==1,])
fitRM1

# there are only 3 lots, take 'lot' as fixed
fitMM2 <- fitLMM(y~(lot+(device))/(day)/(run), VCAdata1[VCAdata1$sample==2,])
# use REML on this (balanced) data
fitMM2.2 <- fitLMM(y~(lot+(device))/(day)/(run), VCAdata1[VCAdata1$sample==2,], "reml")

# the following model definition is equivalent to the one above,
# since a single random term in an interaction makes the interaction
# random (see the 3rd reference for details on this topic)
fitMM3 <- fitLMM(y~(lot+(device))/day/run, VCAdata1[VCAdata1$sample==2,])

# fit same model for each sample using by-processing
lst <- fitLMM(y~(lot+(device))/day/run, VCAdata1, by="sample")
lst

# fit mixed model originally from 'nlme' package
```

```
library(nlme)
data(Orthodont)
fit.lme <- lme(distance~Sex*I(age-11), random=~I(age-11)|Subject, Orthodont)

# re-organize data
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))
fit.anovaMM1 <- fitLMM(distance~Sex*age2+(Subject)*age2, Ortho)

# use simplified formula avoiding unnecessary terms
fit.anovaMM2 <- fitLMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2, Ortho)

# and exclude intercept
fit.anovaMM3 <- fitLMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)

# compare results
fit.lme
fit.anovaMM1
fit.anovaMM2
fit.anovaMM3

# are there a sex-specific differences?
cmat <- getL(fit.anovaMM3, c("SexMale-SexFemale", "SexMale:age2-SexFemale:age2"))
cmat

test.fixef(fit.anovaMM3, L=cmat)

# fit LMM with fixed lot and device effects and test for lot-differences
data(VCAdata1)
fitS5 <- fitLMM(y~(lot+device)/(day)/(run), subset(VCAdata1, sample==5), "reml")
fitS5

# apply Tukey-HSD test to screen for lot differences
library(multcomp)
res.tuk <- glht(fitS5, linfct=mcp(lot="Tukey"))
summary(res.tuk)

# compact letter display
res.tuk.cld <- cld(res.tuk, col=paste0("gray", c(90,60,75)))
plot(res.tuk.cld)

## End(Not run)
```

`fitVCA` *Fit Variance Component Model by ANOVA or REML*

Description

Function serves as interface to functions `anovaVCA` and `remlVCA` for fitting a variance component models (random models) either by ANOVA or REML. All arguments applicable to either one of

these functions can be specified (see anovaVCA or remlVCA for details).

Usage

```
fitVCA(
  form,
  Data,
  method = c("anova", "reml"),
  scale = TRUE,
  VarVC = TRUE,
  ...
)
```

Arguments

form	(formula) specifiying the variance component model (see anovaVCA and/or remlVCA)
Data	(data.frame) containing all variables referenced in 'form'
method	(character) either "anova" to use ANOVA Type-I estimation of variance components or "reml" to use restricted maximum likelihood (REML) estimation of variance component
scale	(logical) TRUE = scale values of the response aiming to avoid numerical problems when numbers are either very small or very large, FALSE = use original scale
VarVC	(logical) TRUE = variance-covariance matrix of variance components will be computed, FALSE = it will not be computed
...	additional arguments to be passed to function anovaVCA or function remlVCA.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

fitLMM, anovaVCA, remlVCA

Examples

```
## Not run:
#load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_2)

# perform ANOVA-estimation of variance components
res.anova <- fitVCA(y~day/run, dataEP05A2_2, "anova")
# perform REML-estimation of variance components
res.reml <- fitVCA(y~day/run, dataEP05A2_2, "reml")

# compare scaling vs. not scaling the response
fit0 <- fitVCA(y~day/run, dataEP05A2_2, "anova", scale=TRUE)
fit1 <- fitVCA(y~day/run, dataEP05A2_2, "anova", scale=FALSE)
```

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

`fixef` *Generic Method for Extracting Fixed Effects from a Fitted Model*

Description

Generic Method for Extracting Fixed Effects from a Fitted Model

Usage

```
fixef(object, ...)
```

Arguments

`object`	(object)
`...`	additional parameters

See Also

`fixef.VCA`

`fixef.VCA` *Extract Fixed Effects from 'VCA' Object*

Description

Conveniently extracting the 'FixedEffects' element of an 'VCA' object.

Usage

```
## S3 method for class 'VCA'
fixef(
  object,
  type = c("simple", "complex"),
  ddfm = c("contain", "residual", "satterthwaite"),
  tol = 1e-12,
  quiet = FALSE,
  ...
)
```

Arguments

object	(VCA) object where fixed effects shall be extracted
type	(character) string or partial string, specifying whether to return "simple" (reduced) or a rather "complex" (more detailed) information about fixed effects
ddfm	(character) string specifying the method used for computing the degrees of freedom of the t-statistic. Only used when type="complex". Available methods are "contain", "residual", and "satterthwaite".
tol	(numeric) value representing the numeric tolerance use in comparisons, values smaller than 'tol' will be considered equal to 0
quiet	(logical) TRUE = suppress warning messages, e.g. for non-estimable contrasts
...	additional parameters

Details

The default is to return the fixed effects estimates together with their standard errors. If setting 'type="complex"' or to an abbreviation (e.g. "c") additional inferential statistics on these estimates will be returned, i.e. "t Value", "DF" and respective p-value "Pr > |t|". One can choose one of three denominator degrees of freedom ('ddfm')-methods. The implementation of these methods are an attempt to align with the results of SAS PROC MIXED. See the respective SAS-documentation for details.

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/(run), dataEP05A2_1)
fixef(fit)

# for complex models it might take some time computing complex output
data(VCAdata1)
fit <- anovaMM(y~(lot+device)/(day)/(run), VCAdata1[VCAdata1$sample==2,])
fixef(fit, "c")

## End(Not run)
```

Fsweep *Calling F90-implementation of the SWEEP-Operator*

Description

Function calls a fast Fortran90-implementation of the SWEEP operator using the transpose of the original augmented matrix $X'X$ (see getSSQsweep). In the sweeping step, also the C matrix, needed to obtain the variance estimates from the sum of squares and the Covariance matrix of the estimates are calculated.

Usage

```
Fsweep(M, asgn, thresh = 1e-10, tol = 1e-10, Ncpu = 1)
```

Arguments

`M`	(matrix) matrix, representing the augmented matrix $X'X$
`asgn`	(integer) vector, identifying columns in M corresponding to variables, respectively, to their coefficients
`thresh`	(numeric) value used to check whether the influence of the a coefficient to reducing the error sum of squares is small enough to conclude that the corresponding column in $X'X$ is a linear combination of preceding columns
`tol`	(numeric) value used to check numerical equivalence to zero
`Ncpu`	(integer) number of cores to be used for parallel processing (not yet used)

Details

This is an utility-function not intended to be called directly.

Value

(list) with eight elements:

`SSQ`	(numeric) vector of ANOVA sum of squares
`LC`	(integer) vector indicating linear dependence of each column
`DF`	(integer) degrees of freedom
`C`	(double precision) Matrix relating the sums of squares to the variances
`Ci`	(double precision) inverse of matrix relating the sums of squares to the variances
`VC`	(double precision) variance
`SD`	(double precision) standard deviations
`Var`	(double precision) covariance matrix of the estimated variances

Author(s)

Florian Dufey <florian.dufey@roche.com>

References

Goodnight, J.H. (1979), A Tutorial on the SWEEP Operator, The American Statistician, 33:3, 149-158

getCI *Extract Confidence Intervals from VCA-Objects.*

Description

This utility function acutally calls function 'VCAinference' first and then extracts the requested confidence interval (CI) information from the resulting object. You can specify single variance components (VC) or multiple. Not specifying any specific VC will return all.

Usage

```
getCI(
  obj,
  vc = NULL,
  type = c("vc", "sd", "cv"),
  tail = c("one-sided", "two-sided"),
  conf.level = 0.95,
  quiet = FALSE
)
```

Arguments

obj	(object) of class "VCA"
vc	(integer, character) specifying which variance component to extract CI for
type	(character) on which scale should results be returned
tail	(character) should one- or two-sided CI be returned
conf.level	(numeric) confidence-level to be used
quiet	(logical) TRUE = suppress additional information to be printed

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
data(dataEP05A2_3)
fit <- remlVCA(y~day/run, dataEP05A2_3)
getCI(fit) # will return one-sided CI for all VC
getCI(fit, type="cv") # now on CV-scale
getCI(fit, type="cv", conf.level=.9)
# multiple row at once
getCI(fit, vc=1:3, type="cv")
getCI(fit, vc=c("total", "error"), type="cv")
```

getDDFM *Degrees of Freedom for Testing Linear Contrasts of Fixed Effects and Least Square Means*

Description

There are three methods implemented, which are all available in SAS PROC MIXED, namely "contain", "residual", and "satterthwaite" approximations. See the documentation of SAS PROC MIXED for details on this topic.

Usage

```
getDDFM(
  obj,
  L,
  ddfm = c("contain", "residual", "satterthwaite"),
  tol = 1e-12,
  method.grad = "simple",
  opt = TRUE,
  items = NULL
)
```

Arguments

obj	(VCA) object
L	(numeric) vector specifying the linear combination of the fixed effect or LS Means
ddfm	(character) string specifying the method used for computing the denominator degrees of freedom for tests of fixed effects or LS Means. Available methods are "contain", "residual", and "satterthwaite".
tol	(numeric) value specifying the numeric tolerance for testing equality to zero
method.grad	(character) string specifying the method to be used for approximating the gradient of the variance-covariance matrix of fixed effects at the estimated covariance parameter estimates (see function 'grad' (numDeriv) for details)
opt	(logical) TRUE = tries to optimize computation time by avoiding unnecessary computations for balanced datasets (see test.fixef).
items	(list) of pre-computed values

Details

The implementation of the Satterthwaite approximation was inspired by the code of function 'calcSatterth' of R-package 'lmerTest'.

Value

(numeric) vector with the specified type of degrees of freedom

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

test.fixef

---

getDF *Extract Degrees of Freedom from Linear Hypotheses of Fixed Effects or LS Means*

Description

Determine degrees of freedom for custom linear hypotheses of fixed effects or LS Means using one of three possible approximation methods.

Usage

```
getDF(obj, L, method = c("contain", "residual", "satterthwaite"), ...)
```

Arguments

obj	(VCA) object
L	(matrix) specifying one or multiple linear hypothese, as returned by function getL
method	(character) the method to be used to determine the degrees of freedom for a linear hypothesis
...	additional parameters

Details

This is a convenience function to determine the DFs for linear hypotheses in the same way as function test.fixef. Only the "DF" part is returned here which can be passed to other functions expecting DFs as input.

Value

(numeric) vector with the DFs for each row of 'L'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(VCAdata1)
tmpDat <- VCAdata1[VCAdata1$sample==1,]
tmpDat <- tmpDat[-c(11,51,73:76),]
fitMM <- anovaMM(y~(lot+device)/(day)/(run), tmpDat)
fitMM
L <- getL(fitMM, c("lot1-lot2", "device1-device2"))
getDF(fitMM, L) # method="contain" is Default
getDF(fitMM, L, method="res")

getDF(fitMM, L, method="satt") # takes quite long for this model

## End(Not run)
```

---

getGB *Giesbrecht & Burns Approximation of the Variance-Covariance Matrix of Variance Components*

Description

Compute variance covariance matrix of variance components of a linear mixed model via the method stated in Giesbrecht and Burns (1985).

Usage

```
getGB(obj, tol = 1e-12)
```

Arguments

obj	(object) with list-type structure, e.g. VCA object fitted by ANOVA or a premature VCA object fitted by REML
tol	(numeric) values < 'tol' will be considered being equal to zero

Details

This function is not intended to be called by users and therefore not exported.

Value

(matrix) corresponding to the Giesbrecht & Burns approximation of the variance-covariance matrix of variance components

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>, Florian Dufey <florian.dufey@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

See Also

vcovVC, remlVCA, remlMM

Examples

```
## Not run:
data(dataEP05A2_3)
fit <- anovaVCA(y~day/run, dataEP05A2_3)
fit <- solveMME(fit) # some additional matrices required
getGB(fit)

## End(Not run)
```

`getIP.remlVCA` *Intermediate Precision for remlVCA-fitted objects of class 'VCA'*

Description

Intermediate precision in this context here means any sum of variances below the full model originally fitted. A typical use case could be reproducibility-experiments with a single lot or multiple lots, where a pooled version of within-lab precision shall be determined.

Usage

```
getIP.remlVCA(obj, vc)
```

Arguments

obj	(object) of class 'VCA' fitted by 'remlVCA'
vc	(character) string specifying the variance component up to which an intermediate precision shall be derived

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
data(dataEP05A2_3)
res <- remlVCA(y~day/run, dataEP05A2_3)
IPday <- getIP.remlVCA(res, "day:run")
VCAinference(IPday)
```

getL *Construct Linear Contrast Matrix for Hypothesis Tests*

Description

Function constructs coefficient/contrast matrices from a string-representation of linear hypotheses.

Usage

```
getL(obj, s, what = c("fixef", "lsmeans"))
```

Arguments

obj	(VCA) object
s	(character) string or vector of strings, denoting one or multiple linear contrasts
what	(character) string specifying whether to construct contrast matrices of fixed effects ("fixed") or LS Means ("lsmeans"), abbreviations are allowed.

Details

Function constructs matrices expressing custom linear hypotheses of fixed effects or LS Means. The user has to specify a string denoting this contrast which is then transformed into a coefficient/contrast matrix. This string may contain names of fixed effects belonging to same same fixed term, numeric coefficients and mathematical operators "+" and "-" (see examples).

Value

(matrix) representing one linear hypothesis of fixed effects or LS Means per row

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(dataEP05A2_2)
fit <- anovaMM(y~day/(run), dataEP05A2_2)
L <- getL(fit, c("day1-day2", "day5-day10"), what="fixef")
L
test.fixef(fit, L=L)

# another custom hypothesis
L2 <- getL(fit, "0.25*day1+0.25*day2+0.5*day3-0.5*day4-0.5*day5")
L2

# more complex model
data(VCAdata1)
```

```
dataS2 <- VCAdata1[VCAdata1$sample==2,]
fit.S2 <- anovaMM(y~(lot+device)/day/(run), dataS2)
L3 <- getL(fit.S2, c("lot1-lot2", "lot1:device3:day19-lot1:device3:day20",
"lot1:device1:day1-lot1:device1:day2"))
L3
test.fixef(fit.S2, L3)

## End(Not run)
```

---

getMat *Extract a Specific Matrix from a 'VCA' Object*

Description

For convenience only, extracting a specific matrix from the "Matrices" element of a 'VCA' object if this matrix exists.

Usage

```
getMat(obj, mat)
```

Arguments

obj	... (VCA) object
mat	... (character) string specifying the matrix to be extracted

Details

When 'mat="Z"' the design matrix of random effects will be returned. If one is interested in the design matrix of random effects for a specific variance component use a name like "Z" + NAME, where NAME has to be equal to the name of the VC in the 'VCA' object (see examples). The same applies to the A-matrices in the quadratic forms, use "A" + NAME for extracting a specific A-matrix.

Value

(matrix) as requested by the user

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
getMat(fit, "Z")
getMat(fit, "Zday")
getMat(fit, "Zday:run")
getMat(fit, "Zerror")
fit2 <- anovaMM(y~day/(run), dataEP05A2_1)
getMat(fit2, "V")   # Var(y)
```

```
getMat(fit2, "G") # Var(re)

## End(Not run)
```

---

getMM *Overparameterized Design Matrices*

Description

Function getMM constructs overparameterized design matrices from a model formula and a data.frame.

Usage

```
getMM(form, Data, keep.order = TRUE)
```

Arguments

form	(formula) with or without response specifying the model to be fit
Data	(data.frame) with the data
keep.order	(logical) TRUE = terms in 'form' should keep their positions, otherwise main effects come first and all interactions will be put into increasing order

Details

This function constructs the overparameterized design matrix for a given dataset 'Data' according to the model formula 'form'. Each combination of factor-levels and or numeric variables is identified and accounted for by a separate column. See examples for differences compared to function 'model.matrix' (stats). This type of design matrix is used e.g. in constructing A-matrices of quadratic forms in y expressing ANOVA sums of squares as such. This is key functionality of functions anovaVCA and anovaMM used e.g. in constructing the coefficient matrix C whose inverse is used in solving for ANOVA Type-1 based variance components..

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
# load example data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_3)
tmpData <- dataEP05A2_3[1:10,]

# check out the differences
getMM(~day+day:run, tmpData)
model.matrix(~day+day:run, tmpData)

# adapt factor variables in 'tmpData'
```

```
tmpData$day <- factor(tmpData$day)

# check out the differences now
getMM(~day+day:run, tmpData)
model.matrix(~day+day:run, tmpData)

# numeric covariate 'cov'
tmpData2 <- dataEP05A2_3[1:10,]
tmpData2$cov <- 10+rnorm(10,,3)
model.matrix(~day*cov, tmpData2)

## End(Not run)
```

`getSSQsweep` *ANOVA Sum of Squares via Sweeping*

Description

Compute ANOVA Type-1 sum of squares for linear models.

Usage

```
getSSQsweep(Data, tobj, random = NULL)
```

Arguments

`Data`	(data.frame) with the data
`tobj`	(terms) object derived from original formula object
`random`	(character) vector, optionally containing information about each model term, whether it is random or fixed (only used in mixed models)

Details

This function performs estimation of ANOVA Type-1 sum of squares using the SWEEP-operator (see reference), operating on the augmented matrix $X'X$, where X represents the design matrix not differentiating between fixed and random factors. See the numerical example in `Fsweep` exemplifying the type of augmentation of $X'X$ on which sweeping is carried out.

This is an utility function not intended to be called directly. For each term in the formula the design-matrix Z is constructed. Matrix X corresponds to binding all these Z-matrices together column-wise.

Degrees of freedom for each term are determined by subtracting the number of linearly dependent columns from the total number of column in X asigned to a specific term.

Value

(list) representing the with variables:

aov.tab	basic ANOVA-table with degrees of freedom (DF), SS and MS
Lmat	(list) with components 'Z' and 'A'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>, Florian Dufey <florian.dufey@roche.com>

References

Goodnight, J.H., (1979), A Tutorial on the SWEEP Operator, The American Statistician, 33:3, p.149-158

See Also

Fsweep

Examples

```
## Not run:
data(dataEP05A2_1)
res <- VCA:::getSSQsweep(dataEP05A2_1, terms(y~day/run))
str(res)

## End(Not run)
```

---

getV *Determine V-Matrix for a 'VCA' Object*

Description

Determine the estimated variance-covariance matrix of observations y.

Usage

```
getV(obj)
```

Arguments

obj	(VCA) object

Details

A linear mixed model can be written as $y = Xb + Zg + e$, where y is the column vector of observations, X and Z are design matrices assigning fixed (b), respectively, random (g) effects to observations, and e is the column vector of residual errors. The variance-covariance matrix of y is equal to $Var(y) = ZGZ^{-T} + R$, where R is the variance-covariance matrix of e and G is the variance-covariance matrix of g. Here, G is assumed to be a diagonal matrix, i.e. all random effects g are mutually independent (uncorrelated).

Value

(VCA) object with additional elements in the 'Matrices' element, including matrix V.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Glucose *Inermediate Precision Data from CLSI EP05-A3*

Description

This data set consists of the Glucose intermediate precision data in the CLSI EP05-A3 guideline, i.e. total variance for a fully-nested design with 3 variance components (day, run, error).
Note, that the results in the original EP05-A3 guideline were obtained using rounded intermediate results, whereas, package `VCA` uses full precision. Any differences between results listed in the CLSI EP05-A3 guideline and those generated by the package are due to error propagation in the working example presented in the CLSI guideline. Here, full precision is used for all intermediate results.

Usage

```
data(Glucose)
```

Format

data.frame with 80 rows and 3 variables.

References

CLSI EP05-A3 - Evaluation of Precision of Quantitative Measurement Procedures; Approved Guideline - Third Edition. CLSI

HugeData *Huge Dataset with Three Variables*

Description

This dataset was added to have a very large dataset available for the unit-test suite. It is an unbalanced dataset with three variables and 8070 observations.

Usage

```
data(HugeData)
```

Format

A data frame with 8070 observations on the following 3 variables.

- y

  The response variable.

- VC1

  Variable with 8 levels corresponding to top-level variance component.

- VC2

  Variable with 3920 levels corresponding to 2nd-level variance component.

isBalanced *Check Whether Design Is Balanced Or Not*

Description

Assess whether an experimental design is balanced or not.

Usage

```
isBalanced(form, Data, na.rm = TRUE)
```

Arguments

form	(formula) object defining the experimental design.
Data	(data.frame) containing all variables appearing in 'form'.
na.rm	(logical) TRUE = delete rows where any is NA, FALSE = NAs are not removed, if there are NAs in the response variable and all information in independent variables is available, then only the design is checked.

Details

This function is for internal use only. Thus, it is not exported.

The approach taken here is to check whether each cell defined by one level of a factor are all equal or not. Here, data is either balanced or unbalanced, there is no concept of "planned unbalancedness" as discussed e.g. in Searle et al. (1992) p.4. The expanded (simplified) formula is divided into main factors and nested factors, where the latter are interaction terms. The N-dimensional contingency table, N being the number of main factors, is checked for all cells containing the same number. If there are differences, the dataset is classified as "unbalanced". All interaction terms are tested individually. Firstly, a single factor is generated from combining factor levels of the first $(n-1)$ variables in the interaction term. The last variable occuring in the interaction term is then recoded as factor-object with M levels. M is the number of factor levels within each factor level defined by the first $(n-1)$ variables in the interaction term. This is done to account for the independence within sub-classes emerging from the combination of the first $(n-1)$ variables.

Value

(logical) TRUE if data is balanced, FALSE if data is unbalanced (according to the definition of balance used)

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data1 <- data.frame(site=gl(3,8), lot=factor(rep(c(2,3,1,2,3,1),
rep(4,6))), day=rep(1:12, rep(2,12)), y=rnorm(24,25,1))

# not all combinations of 'site' and 'lot' in 'data1'

VCA:::isBalanced(y~site+lot+site:lot:day, data1)

# balanced design for this model

VCA:::isBalanced(y~lot+lot:day, data1)

# gets unbalanced if observation is NA

data1[1,"y"] <- NA
VCA:::isBalanced(y~lot+lot:day, data1)
```

```
VCA:::isBalanced(y~lot+lot:day, data1, FALSE)

## End(Not run)
```

`legend.m` *Add Legend to Margin.*

Description

This function accepts all parameters applicable in and forwards them to function `legend`. There will be only made some modifications to the X-coordinate ensuring that the legend is plotted in the right margin of the graphic device. Make sure that you have reserved sufficient space in the right margin, e.g. 'plot.VFP(....., mar=c(4,5,4,10))'.

Usage

```
legend.m(
  x = c("center", "bottomright", "bottom", "bottomleft", "left", "topleft", "top",
     "topright", "right"),
  y = NULL,
  margin = c("right", "bottomright", "bottom", "bottomleft", "left", "topleft", "top",
     "topright"),
  offset = 0.05,
  ...
)
```

Arguments

`x`	(character, numeric) either one of the character strings "center","bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright", "right" or a numeric values specifying the X-coordinate in user coordinates
`y`	(numeric) value specifying the Y-coordiante in user coordinates, only used in case 'x' is numeric
`margin`	(character) string specifying in which part of the margin the legend shall be added, choices are "right", "bottomright", "bottom", "bottomleft", "left", "topleft", "top", "topright" with "right" being the default
`offset`	(numeric) value in [0, 0.5] specifying the offset as fraction in regard to width of the right margin
`...`	all parameters applicable in function `legend`

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:

par( mar=c(10,10,10,10) )
plot(1, type="n", axes=FALSE, xlab="", ylab="")
box()
# add legend to different regions within the 'margin'
legend.m(margin="topleft",  fill="black",legend=c("topleft"))
legend.m(margin="top",  fill="red",  legend=c("top"))
legend.m(margin="topright",  fill="blue",legend=c("topright"))
legend.m(margin="right",  fill="green",legend=c("right"))
legend.m(margin="bottomright",  fill="yellow",legend=c("bottomright"))
legend.m(margin="bottom",  fill="orange",legend=c("bottom"))
legend.m(margin="bottomleft",  fill="cyan",legend=c("bottomleft"))
legend.m(margin="left",  fill="magenta", legend=c("left"))

data(dataEP05A2_3)
dataEP05A2_3$user <- sample(rep(c(1,2), 40))

varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,7), VCnam=list(side=4),
        Points=list(pch=list(var="user", pch=c(2, 8))) )
# always check order of factor levels before annotating
order(unique(dataEP05A2_3$user))
legend.m(pch=c(8,2), legend=c("User 1", "User 2"))

# using different colors
varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,7), VCnam=list(side=4),
         Points=list(col=list(var="user", col=c("red", "green"))) )
legend.m(fill=c("green", "red"), legend=c("User 1", "User 2"))

# two additional classification variables
dataEP05A2_3$user <- sample(rep(c(1,2), 40))
dataEP05A2_3$cls2 <- sample(rep(c(1,2), 40))

# now combine point-coloring and plotting symbols
# to indicate two additional classification variables
varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,10),
         VCnam=list(side=4, cex=1.5),
         Points=list(col=list(var="user", col=c("red", "darkgreen")),
                     pch=list(var="cls2", pch=c(21, 22)),
                     bg =list(var="user", bg =c("orange", "green"))) )

# add legend to (right) margin
legend.m(margin="right", pch=c(21, 22, 22, 22),
         pt.bg=c("white", "white", "orange", "green"),
         col=c("black", "black", "white", "white"),
         pt.cex=c(1.75, 1.75, 2, 2),
         legend=c("Cls2=1", "Cls2=2", "User=2", "User=1"),
         cex=1.5)

## End(Not run)
```

`lmerG` *Construct Variance-Covariance Matrix of Random Effects for Models Fitted by Function 'lmer'*

Description

This function restricts the variance-covariance matrix of random effects G to be either diagonal ('cov=FALSE') or to take any non-zero covariances into account (default, 'cov=TRUE').

Usage

```
lmerG(obj, cov = FALSE)
```

Arguments

`obj`	(object) inheriting from class 'lmerMod'
`cov`	(logical) TRUE = in case of non-zero covariances a block diagonal matrix will be constructed, FALSE = a diagonal matrix with all off-diagonal element being equal to zero will be contructed

Details

This function is not intended to be called directly by users and therefore not exported!

Value

(Matrix) representing the variance-covariance structure of random effects G

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
library(lme4)
data(Orthodont)
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))
fit <-lmer(distance~Sex+Sex:age2+(age2|Subject), Ortho)
G1 <- VCA:::lmerG(fit, cov=FALSE)
G2 <- VCA:::lmerG(fit, cov=TRUE)
G1[1:10,1:10]
G2[1:10,1:10]

## End(Not run)
```

`lmerMatrices` *Derive and Compute Matrices for Objects Fitted by Function 'lmer'*

Description

Function derives and computes all matrices required for down-stream analyses of VCA-objects fitted with REML via function `lmer`.

Usage

```
lmerMatrices(obj, tab = NULL, terms = NULL, cov = FALSE, X = NULL)
```

Arguments

`obj`	(object) inheriting from 'lmerMod'
`tab`	(data.frame) representing the basic VCA-table
`terms`	(character) vector used for ordering variance components
`cov`	(logical) take non-zero covariances among random effects into account (TRUE) or not (FALSE), the latter is the default in this package and also implemented in `remlVCA`, `anovaVCA`, and `anovaMM`.
`X`	(matrix) design matrix of fixed effects as constructed to meet VCA-package requirements

Details

Mixed Model Equations (MME) are solved for fixed and random effects applying the same constraints as in `anovaMM`. The most elaborate and therefore time consuming part is to prepare all matrices required for approximating the variance-covariance matrix of variance components (see `getGB`). To reduce the computational time, this function tries to optimize object-classes depending on whether Intel's (M)ath (K)ernel (L)ibrary could be loaded or not. MKL appears to be more performant with ordinary matrix-objects, whereas all other computations are perfomred using matrix-representations of the `Matrix`-package.

This function is not intended to be called directly by users and therefore not exported.

Value

(list), a premature 'VCA' object

Author(s)

Andre Schuetzenmeister `<andre.schuetzenmeister@roche.com>`

See Also

`remlVCA`, `remlMM`

`lmerSummary` *Derive VCA-Summary Table from an Object Fitted via Function* `lmer`

Description

This function builds a variance components analysis (VCA) table from an object representing a model fitted by function `lmer` of the `lme4` R-package.

Usage

```
lmerSummary(
  obj,
  VarVC = TRUE,
  terms = NULL,
  Mean = NULL,
  cov = FALSE,
  X = NULL,
  tab.only = FALSE
)
```

Arguments

`obj`	(lmerMod) object as returned by function `lmer`
`VarVC`	(logical) TRUE = the variance-covariance matrix of variance components will be approximated following the Giesbrecht & Burns approach, FALSE = it will not be approximated
`terms`	(character) vector, optionally defining the order of variance terms to be used
`Mean`	(numeric) mean value used for CV-calculation
`cov`	(logical) TRUE = in case of non-zero covariances a block diagonal matrix will be constructed, FALSE = a diagonal matrix with all off-diagonal elements being equal to zero will be contructed
`X`	(matrix) design matrix of fixed effects as constructed to meet VCA-package requirements
`tab.only`	(logical) TRUE = will return only the VCA-results table as 'data.frame', argument 'VarVC' will be automatically set to 'FALSE' (see details)

Details

It applies the approximation of the variance-covariance matrix of variance components according to Giesbrecht & Burns (1985) and uses this information to approximate the degrees of freedom according to Satterthwaite (see SAS PROC MIXED documentation option 'CL').

This function can be used to create a VCA-results table from almost any fitted 'lmerMod'-object, i.e. one can apply it to a model fitted via function `lmer` of the `lme4`-package. The only additional argument that needs to be used is 'tab.only' (see examples).

Value

(list) still a premature 'VCA'-object but close to a "complete" 'VCA'-object

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

See Also

remlVCA, remlMM

Examples

```
## Not run:
# fit a model with a VCA-function first
data(VCAdata1)
fit0 <- remlVCA(y~(device+lot)/day/run, subset(VCAdata1, sample==5))
fit0

# fit the same model with function 'lmer' of the 'lme4'-package
library(lme4)
fit1 <- lmer(y~(1|device)+(1|lot)+(1|device:lot:day)+(1|device:lot:day:run),
subset(VCAdata1, sample==5))
lmerSummary(fit1, tab.only=TRUE)

## End(Not run)
```

---

load_if_installed *Load 'RevoUtilsMath'-package if available*

Description

This function is taken from the Rprofile.site file of Microsoft R Open. It was added to the package namespace to avoid a NOTE during the R CMD check process stating that this function is not gobally defined.

Usage

```
load_if_installed(package)
```

Arguments

package	(character) package name to load, usually this will be package 'RevoUtilsMath' if available

Details

Only change to the original version is a different bracketing scheme to match the one used in the remaining source-code of the package.

Author(s)

Authors of the Rprofile.site file in Microsoft R Open.

---

lsmeans *Least Squares Means of Fixed Effects*

Description

Computes Least Squares Means (LS Means) of fixed effects for fitted mixed models of class 'VCA'.

Usage

```
lsmeans(
  obj,
  var = NULL,
  type = c("simple", "complex"),
  ddfm = c("contain", "residual", "satterthwaite"),
  at = NULL,
  contr.mat = FALSE,
  quiet = FALSE
)
```

Arguments

obj	(VCA) object having at least one fixed effect
var	(character) string specifying a fixed effects variable for which LS Means should be computed, defaults to all fixed effects, i.e. for each level of a fixed effects variable ls means will be computed
type	(character) "simple" = fast version of computing LS means
ddfm	(character) string specifying the method used for computing the degrees of freedom of the t-statistic. Only used when type="complex". Available methods are "contain", "residual", and "satterthwaite".
at	(list) where each element corresponds either to a (numeric) covariable or to a factor-variable for which the weighting scheme should be adjusted. See details section for a thorough description of how argument 'at' works and also see the examples.

contr.mat	(logical) TRUE = the LS Means generating contrast-matrix will be added to the result as attribute contrasts
quiet	(logical) TRUE = suppress warning messages, e.g. for non-estimable contrasts

Details

Function computes LS Means of fixed effects and their corresponding standard errors. In case of setting argument 'type' equal to "complex" (or any abbreviation) a t-test is performed on each LS Mean, returning degrees of freedom, t-statistic and corresponding p-values. One can choose from one of three denominator degrees of freedom ('ddfm')-methods.

Actually, function test.fixef is called with the "no intercept" version of the fitted model. The "complex" option is significantly slower for unbalanced designs (see test.fixef for details). In case that the 'VarCov' element of the 'VCA' object already exists (calling vcovVC), which is the most time consuming part, results can be obtained in less amount of time.

Standard Errors of LS Means are computed as TPT^T, where T is the LS Means generating contrast matrix and P is the variance-covariance matrix of fixed effects.

Argument at can be used to modify the values of covariables when computing LS Means and/or to apply different weighting schemes for (fixed) factor variables in the model, e.g. when the prevelance of factor-levels differs from a uniform distribution. Usually, if the weighting scheme is not modified, each factor-level will contribute $1/N$ to the LS Mean, where N corresponds to the number of factor-levels.

Covariables have to be specified as 'name=value', where value can be a vector of length > 1. Each value will be evaluated for each row of the original LS Means contrast matrix. If multiple covariables are specified, the i-th element of covariable 1 will be matched with the i-th element of covariable(s) 2...M, where M is the number of covariables in the model.

To apply a different weighting scheme for factor-variables one has to specify 'factor-name=c(level-name_1=value_1, level-name_2=value_2, ..., level-name_N=value_N)'. The sum of all 'value_i' elements must be equal to 1, otherwise, this factor-variable will be skipped issuing a warning. If any levels 'level-name_i' cannot be found for factor-variable 'factor-name', this variable will also be skipped and a warning will be issued. See the examples section to get an impression of how this works.

Value

(matrix) with LS Means of fixed effects and respective standard errors, in case of 'type="complex"'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
#
## Not run:
data(dataEP05A2_2)
fit1 <- anovaMM(y~day/(run), dataEP05A2_2)
lsmeans(fit1)
lsmeans(fit1,, "complex")
```

```
# a more complex model
data(VCAdata1)
fit2 <- anovaMM(y~(lot+device)/(day)/(run), VCAdata1[VCAdata1$sample==2,])
lsmeans(fit2, "lot")
lsmeans(fit2, "device", "complex")

# pre-computed 'VarCov' element saves time
system.time(lsm1 <- lsmeans(fit2, "device", "complex"))
fit2$VarCov <- vcovVC(fit2)
system.time(lsm2 <- lsmeans(fit2, "device", "complex"))
lsm1
lsm2

# simulate some random data
set.seed(212)
id <- rep(1:10,10)
x <- rnorm(200)
time <- sample(1:5,200,replace=T)
y <- rnorm(200)+time
snp <- sample(0:1,200,replace=T)
dat <- data.frame(id=id,x=x,y=y,time=time,snp=snp)
dat$snp <- as.factor(dat$snp)
dat$id <- as.factor(dat$id)
dat$time <- as.numeric(dat$time)
dat$sex <- gl(2, 100, labels=c("Male", "Female"))
dat$y <- dat$y + rep(rnorm(2, 5, 1), c(100, 100))

fit3 <- remlMM(y~snp+time+snp:time+sex+(id)+(id):time, dat)

# compute standard LS Means for variable "snp"
lsmeans(fit3, var="snp")
lsmeans(fit3, var="snp", type="c")    # comprehensive output

# compute LS Means at timepoints 1, 2, 3, 4
# Note: original LS Means are always part of the output
lsmeans(fit3, var="snp", at=list(time=1:4))

# compute LS Means with different weighting scheme
# for factor-variable 'sex'
lsmeans(fit3, var="snp", at=list(sex=c(Male=.3, Female=.7)))

# combine covariables at some value and altering the
# weighting scheme
lsmeans(fit3, var="snp", at=list(time=1:4, sex=c(Male=.3, Female=.7)))

# now with comprehensive output and requesting the
# LS Means generating contrast matrix
lsmeans(fit3, var="snp", type="complex", contr.mat=TRUE,
at=list(time=1:4, sex=c(Male=.3, Female=.7)))

## End(Not run)
```

LSMeans_Data *Dataset for Unit-Testing of LS Means*

Description

This data set is used for unit-testing LS Means functionality. Reference results were generated in SAS PROC MIXED and rounded to two decimals as covariance parameter estimates slightly differ.

Usage

```
data(LSMeans_Data)
```

Format

data.frame with 200 rows and 6 variables.

lsmMat *Contrast Matrix for LS Means*

Description

Function determines appropriate contrast matrix for computing the LS Means of each factor level of one or multiple fixed effects variables.

Usage

```
lsmMat(obj, var = NULL, quiet = FALSE)
```

Arguments

obj	(VCA) object
var	(character) string specifyig the fixed effects variable for which the LS Means generating matrices should be computed
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

This functions implements the 5 rules given in the documentation of SAS PROC GLM for computing the LS Means.#' The LS Means correspond to marginal means adjusted for bias introduced by unbalancedness.

Value

(matrix) where each row corresponds to a LS Means generating contrast for each factor level of one or multiple fixed effects variable(s)

Author(s)

Andre Schutzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(dataEP05A2_1)
fit1 <- anovaMM(y~day/run, dataEP05A2_1)

VCA:::lsmMat(fit1, "day") # function not exported
VCA:::lsmMat(fit1, "run")
VCA:::lsmMat(fit1) # is equal to listing all fixed terms

# a more complex and unbalanced model
data(VCAdata1)
datS1 <- VCAdata1[VCAdata1$sample == 1, ]
set.seed(42)
datS1ub <- datS1[-sample(1:nrow(datS1))[1:25],]
fit2 <- anovaMM(y~(lot+device)/day/(run), datS1ub)
VCA:::lsmMat(fit2, c("lot", "device"))

## End(Not run)
```

MLrepro *Multi-Lot Reproducibility Data.*

Description

This data set consists of 754 observations. There are 3 laboratories (Lab), 3 lots (Lot), 21 days (Days) per lab-lot combination, and 2 runs per day. The response variable is Result. This dataset is used in examples and unit-tests (see subdir 'UnitTests' of the package-dir).

Usage

```
data(MLrepro)
```

Format

data.frame with 754 rows and 5 variables.

model.frame.VCA *Extract the Model Frame from a 'VCA' Object*

Description

Function returns the data-element of 'object' and adds the terms-element as attribute.

Usage

```
## S3 method for class 'VCA'
model.frame(formula, ...)
```

Arguments

formula	(VCA) object
...	additional arguments

Details

It enables application of functions relying on the existence of this method, e.g. the functin 'glht' of the 'multcomp' R-package.

Value

(data.frame) with attribute 'terms'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

model.matrix.VCA *Model Matrix of a Fitted VCA-Object*

Description

Function returns matrix X corresponding to the design matrix of fixed effects of the fitted model.

Usage

```
## S3 method for class 'VCA'
model.matrix(object, ...)
```

Arguments

object	(VCA) object
...	further arguments

MPinv *Moore-Penrose Generalized Inverse of a Matrix*

Description

This function is originally impelemented in package 'MASS' as function `ginv`. It was adapted to be able to deal with matrices from the 'Matrix' package, e.g. sparse matrices.

Usage

```
MPinv(X, tol = sqrt(.Machine$double.eps))
```

Arguments

`X`	(object) two-dimensional, for which a Moore-Penrose inverse has to be computed
`tol`	(numeric) tolerance value to be used in comparisons

Value

(object) A Moore-Penrose inverse of X.

Author(s)

Authors of the 'MASS' package.

orderData *Re-Order Data.Frame*

Description

Functions attempts to standardize input data for linear mixed model analyses to overcome the problem that analysis results sometimes depend on ordering of the data and definition of factor-levels.

Usage

```
orderData(Data, trms, order.data = TRUE, exclude.numeric = TRUE, quiet = FALSE)
```

Arguments

Data	(data.frame) with input data intented to put into standard-order
trms	(formula, terms) object speciying a model to be fitted to Data
order.data	(logical) TRUE = variables will be increasingly ordered, FALSE = order of the variables remains as is
exclude.numeric	(logical) TRUE = numeric variables will not be included in the reordering, which is required whenever this variable serves as covariate in a LMM, FALSE = numeric variables will also be converted to factors, useful in VCA-analysis, where all variables are interpreted as class-variables
quiet	(logical) TRUE = omits any (potentially) informative output regarding re-ordering and type-casting of variables

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
# random ordering
data(dataEP05A2_1)
dat <- dataEP05A2_1
levels(dat$day) <- sample(levels(dat$day))
# this has direct impact e.g. on order of estimated effects
fit <- anovaVCA(y~day/run, dat, order.data=FALSE)
ranef(fit)
# to guarantee consistent analysis results
# independent of the any data orderings option
# 'order.data' is per default set to TRUE:
fit <- anovaVCA(y~day/run, dat)
ranef(fit)
# which is identical to:
fit2 <- anovaVCA(y~day/run, orderData(dat, y~day/run), order.data=FALSE)
ranef(fit2)

## End(Not run)
```

Orthodont *Orthodont dataset from R-package 'nlme'*

Description

The Orthodont data frame has 108 rows and 4 columns of the change in an orthdontic measurement over time for several young subjects.

This dataset was included to simplify its usage in automated unit-tests (see directory UnitTests) and examples.

Investigators at the University of North Carolina Dental School followed the growth of 27 children (16 males, 11 females) from age 8 until age 14. Every two years they measured the distance between the pituitary and the pterygomaxillary fissure, two points that are easily identified on x-ray exposures of the side of the head.

Usage

```
data(Orthodont)
```

Format

data.frame with 80 rows and 3 variables.

References

Pinheiro, J. C. and Bates, D. M. (2000), Mixed-Effects Models in S and S-PLUS, Springer, New York. (Appendix A.17)

Potthoff, R. F. and Roy, S. N. (1964), A generalized multivariate analysis of variance model useful especially for growth curve problems, Biometrika, 51, 313-326.

---

plot.VCA *Standard 'plot' Method for 'VCA' S3-Objects.*

Description

Create a variability chart from a 'VCA'-object, i.e. from a fitted model.

Usage

```
## S3 method for class 'VCA'
plot(x, ...)
```

Arguments

x	(VCA) object
...	additional arguments to be passed to or from methods.

Details

This function extracts the data and the model-formula from a fitted 'VCA'-object and calls function varPlot accepting all its arguments. Please see the documention of function varPlot for a detailed description.

It will not be differentiated between fixed and random effects when calling this function on a fitted linear mixed model.

Value

nothing, instead a plot is generated

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

varPlot, anovaVCA,remlVCA, anovaMM,remlMM

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)

# standard plot without any extras
plot(fit)

# plot with some additional features
plot(fit, MeanLine=list(var=c("int", "day"), col=c("cyan", "blue"), lwd=c(2,2)))

# more complex model
data(realData)
Data <- realData[realData$PID == 1,]
fit2 <- anovaVCA(y~(calibration+lot)/day/run, Data)
plot(fit2,
BG=list(var="calibration",
col=c("#f7fcfd","#e5f5f9","#ccece6","#99d8c9",
      "#66c2a4","#41ae76","#238b45","#006d2c","#00441b"),
col.table=TRUE),
VLine=list(var=c("calibration", "lot"),
   col=c("black", "darkgray"), lwd=c(2,1), col.table=TRUE),
JoinLevels=list(var="lot", col=c("#ffffb2","orangered","#feb24c"),
        lwd=c(2,2,2)),
MeanLine=list(var="lot", col="blue", lwd=2))


## End(Not run)
```

---

plotRandVar *Plot Random Variates of a Mixed Model ('VCA' Object).*

Description

Plots, possibly transformed, random variates of a linear mixed model (random effects, contitional or marginal residuals).

Usage

```
plotRandVar(
  obj,
  term = NULL,
  mode = c("raw", "student", "standard", "pearson"),
  main = NULL,
  Xlabels = list(),
  Points = list(),
  Vlines = list(),
  pick = FALSE,
  ...
)
```

Arguments

obj	(VCA) object
term	(character, integer) specifying a type of residuals if one of c("conditional", "marginal"), or, the name of a random term (one of obj$re.assign$terms). If 'term' is a integer, it is interpreted as the i-th random term in 'obj$re.assign$terms'.
mode	(character) string specifying a possible transformation of random effects or residuals (see residuals.VCA and ranef.VCAfor details)
main	(character) string used as main title of the plot, if NULL, it will be automatically generated
Xlabels	(list) passed to function text adding labels to the bottom margin at x-coordinates 1:N, where N is the number of random variates. Useful for customization.
Points	(list) passed to function points for customization of plotting symbols
Vlines	(list) passed to function (abline) adding vertical lines, separating random variates for better visual separation, set to NULL for omitting vertical lines.
pick	(logical) TRUE = lets the user identify single points using the mouse, useful, when many, points were drawn where the X-labels are not readable.
...	additional arguments to be passed to methods, such as graphical parameters (see par)

Details

Function plots either random effects of a 'VCA' object or residuals. Parameter 'term' is used to specify either one. If 'term' is one of c("conditional", "marginal") corresponding residuals will be plotted (see resid for details). If 'term' is either the name of a random term in the formula of the 'VCA' object or an integer specifying the i-th random term, corresponding random effects will be plotted. Both types of random variates (random effects, residuals) can be plotted untransformed ("raw"), "studentized" or "standardized". In case of residuals, one can also use the "Pearson"-type transformation.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
 ## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
# solve mixed model equations including random effects
fit <- solveMME(fit)
plotRandVar(fit, "cond", "stand")
plotRandVar(fit, 1, "stud") # 1st random term 'day'
plotRandVar(fit, "day", "stud") # equivalent to the above

# for larger datasets residuals can hardly be identified
# pick out interesting points with the mouse

plotRandVar(fit, "marg", "stud", pick=TRUE)

# customize the appearance
plotRandVar( fit, 1, "stud", Vlines=list(col=c("red", "darkgreen")),
Xlabels=list(offset=.5, srt=60, cex=1, col="blue"),
Points=list(col=c("black", "red", rep("black", 18)),
pch=c(3,17,rep(3,18)), cex=c(1,2,rep(1,18))))

## End(Not run)
```

`predict.VCA` *Predictions from a Model Fitted by* `fitLMM`

Description

Model returns fitted values in case `newdata` is NULL or evaluates the fitted model employing user-specified data `newdata`. The default is that fitted values incorporate fixed effects and random effects, leaving out the (conditional) residuals only. If the interest lies in constraining predictions to the fixed effects only set `re=NA` or incorporate just part of the random variability specifying distinct random effects (see `re`.

Usage

```
## S3 method for class 'VCA'
predict(object, newdata = NULL, re = NULL, allow.new.levels = FALSE, ...)
```

Arguments

`object`	(VCA) object fitted via function `fitLMM`
`newdata`	(data.frame) with all variables required for the specified prediction, i.e. the default settings require all variables of the original model, in case of `re=NA`, only variables corresponding to fixed effects are required.
`re`	(character) if NULL (default) all random effects will be included, to restrict predictions to the fixed effects use `re=NA`, for a subset of random effects included in predictions use any valid random effects specification, i.e. `object$random`

allow.new.levels
: (logical) if new levels (no part of the original fitted model) in newdata are allowed. If FALSE (default), such new values in newdata will trigger an error; if TRUE, then the prediction will use the unconditional (population-level) values for data with previously unobserved levels (or NAs).

...
: additional arguments passdo or from other methods

Value

(numeric) vector of prediction results

Author(s)

Andre Schuetzenmeister <andre.schuetzeneister@roche.com>

Examples

```
## Not run:
# fit LMM with fixed lot and device effects and test for lot-differences
data(VCAdata1)
datS5 <- subset(VCAdata1, sample==5)
fitS5 <- fitLMM(y~(lot+device)/(day)/(run), datS5, "anova")
fitS5

# fitted values including fixed and random effects
pred0 <- predict(fitS5)
pred0
# sanity check:
all(round(pred0 + resid(fitS5) - datS5$y, 12) == 0)
# restrict to fixed effects
predict(fitS5, re=NA)
# restrict to fixed effects and dayly random effects
# see required names
fitS5$random
predict(fitS5, re="lot:device:day")

# check against original 'lmer'-predictions
# use version from VCA-package (ordinary data.frame)
data(Orthodont, package="VCA")
Ortho <- Orthodont
Ortho$age2 <- Ortho$age-11
# use exactly the same data, same ordering
Ortho <- orderData(Ortho, distance ~ Sex * age2 + (Subject) * age2)
fit.fitLMM <- fitLMM(distance ~ Sex * age2 + (Subject) * age2, Ortho, "reml")
library(lme4)
fit.lmer <- lmer(distance ~ Sex + age2 + Sex:age2 + (age2 | Subject), Ortho)
# check fitted value first (fixed + random effects)
predict(fit.lmer)
predict(fit.fitLMM)
# restrict to fixed part only
predict(fit.lmer, re.form=NA)
predict(fit.fitLMM, re=NA)
```

```
# user-specified 'newdata'
newdata <- Ortho[45:54,]
newdata$age2 <- newdata$age2 + 5
# include fixed and random effects
predict(fit.lmer, newdata)
predict(fit.fitLMM, newdata)
# generate new data
newdata <- Ortho[45:54,]
newdata$age2 <- newdata$age2 + 5
# predict on newdata using fixed and random effects
predict(fit.lmer, newdata)
predict(fit.fitLMM, newdata)
# restrict prediction to random Subject effects
predict(fit.lmer, newdata, re.form=~(1|Subject))
predict(fit.fitLMM, newdata, re="Subject")
# restrict prediction to random per-Subject slope
predict(fit.lmer, newdata, re.form=~(age2-1|Subject))
predict(fit.fitLMM, newdata, re="age2:Subject")

## End(Not run)
```

`print.VCA` *Standard Printing Method for Objects of Class 'VCA'*

Description

Function prints 'VCA' objects as returned e.g. by function anovaVCA.

Usage

```
## S3 method for class 'VCA'
print(x, digits = 6L, ...)
```

Arguments

`x`	(VCA) object of class 'VCA' as returned by function 'anovaVCA'.
`digits`	(integer) number of digits numeric values are rounded to before printing.
`...`	additional arguments to be passed to or from methods.

print.VCAinference *Standard Print Method for Objects of Class 'VCAinference'*

Description

Prints the list-type 'VCAinference'-object as tabulated output.

Usage

```
## S3 method for class 'VCAinference'
print(x, digits = 4L, what = c("all", "VC", "SD", "CV", "VCA"), ...)
```

Arguments

x	(VCAinference) object
digits	(integer) number of decimal digits.
what	(character) one of "all", "VC", "SD", "CV", "VCA" specifying which part of the 'VCA'-object is to be printed.
...	additional arguments to be passed to or from methods.

Details

Formats the list-type objects of class 'VCAinference' for a more comprehensive presentation of results, which are easier to grasp. The default is to show the complete object (VCA ANOVA-table, VC-, SD-, and CV-CIs). Using parameter 'what' allows to restrict the printed output to certain parts. Print-function invisibly returns a matrix or a list of matrices, depending on the values of 'what', i.e. it can be used as for packing the inference-information in one or multiple matrix-objects and extracting it/them.

Value

invisibly returns sub-elements of 'x' specified via 'what'

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

VCAinference, anovaVCA

Examples

```
## Not run:
# load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_1)

# perform ANOVA-estimation of variance components for a nested design
res <- anovaVCA(y~day/run, Data=dataEP05A2_1)
res
inf <- VCAinference(res)
inf

# show certain parts and extract them invisibly
CVmat <- print(inf, what="CV")
CVmat

# show numerical values with more digits
print(inf, digit=12)

## End(Not run)
```

`protectedCall` *Wrap Function-Calls to Execute Additional Checks.*

Description

Function can be used to wrap function-calls, here, intended for model fitting functions `anovaVCA`, `anovaMM`, `remlVCA`, `remlMM`, `fitVCA`, and `fitLMM`. When wrapped, there is the option to perform additional checks and reporting back identified problems by setting 'ErrorType="Detailed"'. There is no error-handling provided by this function, i.e. any error issued will remain an error. It would need to be handled by `try`, `tryCatch` or similar. Note, that inline definition of datasets within 'expr' is not supported and will issue an error.

Usage

```
protectedCall(expr, ErrorType = c("Simple", "Detailed"))
```

Arguments

`expr`	(expression) to be protected, typically, a call to a model-fitting function from this package (see details)
`ErrorType`	(ErrorType) "Simple"=default error-messages, "Detailed"= additional data consistency checks will be performed

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
# nothing happens if no error occurs
data(dataEP05A2_1)
res <- protectedCall(anovaVCA(form=y~day/run, Data=dataEP05A2_1))
res
# error message without additional consistency checks (default)
dat3 <- data.frame( y=rnorm(8,10),
day=rep(c(1,2),c(4,4)),
run=rep(c(2,1), c(4,4)))
protectedCall(anovaVCA(form=y~day/run, Data=dat3), ErrorType="Simple")
# error message with additional consistency checks hopefully helpful for the user
protectedCall(anovaVCA(form=y~day/run, Data=dat3), ErrorType="Detailed")

# handle error
res <- try(protectedCall(anovaVCA(form=y~day/run, Data=dat3), ErrorType="Detailed"), silent=TRUE)
if(is(res, "try-error"))
cat(sub(", ErrorType .*\\)", "", sub("protectedCall\\(", "", res)))

# inline-definition of data.frames issues an error
protectedCall(anovaVCA( form=y~day/run,
Data=data.frame(y=rnorm(8,10),
day=rep(c(1,2),c(4,4)),
run=rep(c(2,1), c(4,4)))))

## End(Not run)
```

`ranef` *Generic Method for Extracting Random Effects from a Fitted Model*

Description

Generic Method for Extracting Random Effects from a Fitted Model

Usage

```
ranef(object, ...)
```

Arguments

`object`	(object)
`...`	additional parameters

See Also

`ranef.VCA`

`ranef.VCA` *Extract Random Effects from 'VCA' Object*

Description

Extract random effects and possibly apply a transformation to them (standardization, studentization).

Usage

```
## S3 method for class 'VCA'
ranef(
  object,
  term = NULL,
  mode = c("raw", "student", "standard"),
  quiet = FALSE,
  ...
)
```

Arguments

`object`	(VCA) object from which random effects shall be extracted
`term`	(character) string specifying a term (factor) for which random effects should be extracted, one can also specify an integer which is interpreted as i-th element of 'obj$res.assign$terms'
`mode`	(character) string or abbreviation specifying whether "raw" residuals should be returned or a transformed version c("student" or "standard")
`quiet`	(logical) TRUE = will suppress any warning, which will be issued otherwise
`...`	additional parameters

Details

Extracting the 'RandomEffects' element of an 'VCA' object if this exists and applying standardization (mean 0, sd 1) or studentization. For studentized random effects the i-th random effects is divided by the i-th main diagonal element of matrix $O = GZ^TQZG$, where G is the covariance-matrix of random effects, Z is a design matrix assigning random effects to observations and matrix $Q = V^{-1}(I - H)$ (see `residuals.VCA` for further details).

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Laird, N.M., Ware, J.H., 1982. Random effects models for longitudinal data. Biometrics 38, 963-974.

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1)
ranef(fit)

# get variable-specific random effects (REs)
# both extract the same REs
ranef(fit, "day")
ranef(fit, 1)

# get standardized REs
ranef(fit, "day:run", "standard")

# or studentized REs
ranef(fit, 2, "stu")

## End(Not run)
```

---

realData *Real-World Data*

Description

This dataset is meant to serve as real-world representative completing the collection of datasets coming with this package. There are 6 variables, one response variable ('y') corresponding to concentration values of the measurand, and 5 factor variables. Variable "calibration" corresponds to the day a (re-) calibration was performed, all other variables are more or less self-explaining.

Usage

```
data(realData)
```

Format

data.frame with 2268 rows and 6 variables.

---

remlMM *Fit Linear Mixed Models via REML*

Description

Function fits Linear Mixed Models (LMM) using Restricted Maximum Likelihood (REML).

Usage

```
remlMM(
  form,
  Data,
  by = NULL,
  VarVC = TRUE,
  cov = TRUE,
  quiet = FALSE,
  order.data = TRUE
)
```

Arguments

form	(formula) specifying the model to be fit, a response variable left of the '~' is mandatory, random terms have to be enclosed in brackets (see details for definition of valid model terms)
Data	(data.frame) containing all variables referenced in 'form'
by	(factor, character) variable specifying groups for which the analysis should be performed individually, i.e. by-processing
VarVC	(logical) TRUE = the variance-covariance matrix of variance components will be approximated using the method found in Giesbrecht & Burns (1985), which also serves as basis for applying a Satterthwaite approximation of the degrees of freedom for each variance component, FALSE = leaves out this step, no confidence intervals for VC will be available
cov	(logical) TRUE = in case of non-zero covariances a block diagonal matrix will be constructed, FALSE = a diagonal matrix with all off-diagonal element being equal to zero will be contructed
quiet	(logical) TRUE = will suppress any messages or warning, which will be issued otherwise
order.data	(logical) TRUE = class-variables will be ordered increasingly, FALSE = ordering of class-variables will remain as is

Details

The model is formulated exactly as in function anovaMM, i.e. random terms need be enclosed by round brackets. All terms appearing in the model (fixed or random) need to be compliant with the regular expression "^[^[\.]]?[[:alnum:]_\.]*$", i.e. they may not start with a dot and may then only consist of alpha-numeric characters, dot and underscore. Otherwise, an error will be issued.

Here, a LMM is fitted by REML using the lmer function of the lme4-package. For all models the Giesbrechnt & Burns (1985) approximation of the variance-covariance matrix of variance components (VC) can be applied ('VarVC=TRUE'). A Satterthwaite approximation of the degrees of freedom for all VC and total variance is based on this approximated matrix using $df = 2Z^2$, where Z is the Wald statistic $Z = \sigma^2/se(\sigma^2)$, and σ^2 is here used for an estimated variance. The variance of total variability, i.e. the sum of all VC is computed via summing up all elements of the variance-covariance matrix of the VC. One can constrain the variance-covariance matrix of random effects G to be either diagonal ('cov=FALSE'), i.e. all random effects are indpendent of each other

(covariance is 0). If 'cov=TRUE' (the default) matrix G will be constructed as implied by the model returned by function lmer.

As for objects returned by function anovaMM linear hypotheses of fixed effects or LS Means can be tested with functions test.fixef and test.lsmeans. Note, that option "contain" does not work for LMM fitted via REML.

Note, that for large datasets approximating the variance-covariance matrix of VC is computationally expensive and may take very long. There is no Fisher-information matrix available for 'merMod' objects, which can serve as approximation. To avoid this time-consuming step, use argument 'VarVC=FALSE' but remember, that no confidence intervals for any VC will be available. If you use Microsoft's R Open, formerly known as Revolution-R, which comes with Intel's Math Kernel Library (MKL), this will be automatically detected and an environment-optimized version will be used, reducing the computational time considerably (see examples).

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

remlVCA, VCAinference, ranef.VCA, residuals.VCA, anovaVCA, anovaMM, plotRandVar, test.fixef, test.lsmeans, lmer

Examples

```
## Not run:
data(dataEP05A2_2)

# assuming 'day' as fixed, 'run' as random
remlMM(y~day/(run), dataEP05A2_2)

# assuming both as random leads to same results as
# calling anovaVCA
remlMM(y~(day)/(run), dataEP05A2_2)
anovaVCA(y~day/run, dataEP05A2_2)
remlVCA(y~day/run, dataEP05A2_2)

# fit a larger random model
data(VCAdata1)
fitMM1 <- remlMM(y~((lot)+(device))/(day)/(run), VCAdata1[VCAdata1$sample==1,])
fitMM1
# now use function tailored for random models
fitRM1 <- anovaVCA(y~(lot+device)/day/run, VCAdata1[VCAdata1$sample==1,])
fitRM1

# there are only 3 lots, take 'lot' as fixed
fitMM2 <- remlMM(y~(lot+(device))/(day)/(run), VCAdata1[VCAdata1$sample==2,])

# the following model definition is equivalent to the one above,
# since a single random term in an interaction makes the interaction
# random (see the 3rd reference for details on this topic)
fitMM3 <- remlMM(y~(lot+(device))/day/run, VCAdata1[VCAdata1$sample==2,])
```

```
# fit same model for each sample using by-processing
lst <- remlMM(y~(lot+(device))/day/run, VCAdata1, by="sample")
lst

# fit mixed model originally from 'nlme' package

library(nlme)
data(Orthodont)
fit.lme <- lme(distance~Sex*I(age-11), random=~I(age-11)|Subject, Orthodont)

# re-organize data for using 'remlMM'
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))
fit.remlMM1 <- remlMM(distance~Sex*age2+(Subject)*age2, Ortho)

# use simplified formula avoiding unnecessary terms
fit.remlMM2 <- remlMM(distance~Sex+age2+Sex:age2+(Subject)+age2:(Subject), Ortho)

# and exclude intercept
fit.remlMM3 <- remlMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)

# now use exclude covariance of per-subject intercept and slope
# as for models fitted by function 'anovaMM'
fit.remlMM4 <- remlMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho, cov=FALSE)

# compare results
fit.lme
fit.remlMM1
fit.remlMM2
fit.remlMM3
fit.remlMM4

# are there a sex-specific differences?
cmat <- getL(fit.remlMM3, c("SexMale-SexFemale", "SexMale:age2-SexFemale:age2"))
cmat

test.fixef(fit.remlMM3, L=cmat)

## End(Not run)
```

`remlVCA` *Perform (V)ariance (C)omponent (A)nalysis via REML-Estimation*

Description

Function performs a Variance Component Analysis (VCA) using Restricted Maximum Likelihood (REML) to fit the random model, i.e. a linear mixed model (LMM) where the intercept is the only fixed effect.

Usage

```
remlVCA(form, Data, by = NULL, VarVC = TRUE, quiet = FALSE, order.data = TRUE)
```

Arguments

form	(formula) specifying the model to be fit, a response variable left of the '~' is mandatory
Data	(data.frame) containing all variables referenced in 'form'
by	(factor, character) variable specifying groups for which the analysis should be performed individually, i.e. by-processing
VarVC	(logical) TRUE = the variance-covariance matrix of variance components will be approximated using the method found in Giesbrecht & Burns (1985), which also serves as basis for applying a Satterthwaite approximation of the degrees of freedom for each variance component, FALSE = leaves out this step, no confidence intervals for VC will be available
quiet	(logical) TRUE = will suppress any messages or warnings, which will be issued otherwise
order.data	(logical) TRUE = class-variables will be ordered increasingly, FALSE = ordering of class-variables will remain as is

Details

Here, a variance component model is fitted by REML using the lmer function of the lme4-package. For all models the Giesbrechnt & Burns (1985) approximation of the variance-covariance matrix of variance components (VC) is applied. A Satterthwaite approximation of the degrees of freedom for all VC and total variance is based on this approximated matrix using $df = 2Z^2$, where Z is the Wald statistic $Z = \sigma^2/se(\sigma^2)$, and σ^2 is here used for an estimated variance. The variance of total variability, i.e. the sum of all VC is computed via summing up all elements of the variance-covariance matrix of the VC. Note, that for large datasets approximating the variance-covariance matrix of VC is computationally expensive and may take very long. There is no Fisher-information matrix available for 'merMod' objects, which can serve as approximation. To avoid this time-consuming step, use argument 'VarVC=FALSE' but remember, that no confidence intervals for any VC will be available. If you use Microsoft's R Open, formerly known as Revolution-R, which comes with Intel's Math Kernel Library (MKL), this will be automatically detected and an environment-optimized version will be used, reducing the computational time very much (see examples).

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

remlMM, VCAinference, ranef.VCA, residuals.VCA, anovaVCA, anovaMM, plotRandVar, lmer

Examples

```
## Not run:

# a VCA standard example
data(dataEP05A2_3)

# fit it by ANOVA first, then by REML
fit0 <- anovaVCA(y~day/run, dataEP05A2_3)
fit1 <- remlVCA(y~day/run, dataEP05A2_3)
fit0
fit1

# make example unbalanced
set.seed(107)
dat.ub <- dataEP05A2_3[-sample(1:80, 7),]
fit0ub <- anovaVCA(y~day/run, dat.ub)
fit1ub <- remlVCA(y~day/run, dat.ub)

# not that ANOVA- and REML-results now differ
fit0ub
fit1ub

### Use the six sample reproducibility data from CLSI EP5-A3
### and fit per sample reproducibility model
data(CA19_9)
fit.all <- remlVCA(result~site/day, CA19_9, by="sample")

reproMat <- data.frame(
Sample=c("P1", "P2", "Q3", "Q4", "P5", "Q6"),
Mean= c(fit.all[[1]]$Mean, fit.all[[2]]$Mean, fit.all[[3]]$Mean,
fit.all[[4]]$Mean, fit.all[[5]]$Mean, fit.all[[6]]$Mean),
Rep_SD=c(fit.all[[1]]$aov.tab["error","SD"], fit.all[[2]]$aov.tab["error","SD"],
fit.all[[3]]$aov.tab["error","SD"], fit.all[[4]]$aov.tab["error","SD"],
fit.all[[5]]$aov.tab["error","SD"], fit.all[[6]]$aov.tab["error","SD"]),
Rep_CV=c(fit.all[[1]]$aov.tab["error","CV[%]"],fit.all[[2]]$aov.tab["error","CV[%]"],
fit.all[[3]]$aov.tab["error","CV[%]"],fit.all[[4]]$aov.tab["error","CV[%]"],
fit.all[[5]]$aov.tab["error","CV[%]"],fit.all[[6]]$aov.tab["error","CV[%]"]),
WLP_SD=c(sqrt(sum(fit.all[[1]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[2]]$aov.tab[3:4, "VC"])),
sqrt(sum(fit.all[[3]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[4]]$aov.tab[3:4, "VC"])),
sqrt(sum(fit.all[[5]]$aov.tab[3:4,"VC"])),sqrt(sum(fit.all[[6]]$aov.tab[3:4, "VC"]))),
WLP_CV=c(sqrt(sum(fit.all[[1]]$aov.tab[3:4,"VC"]))/fit.all[[1]]$Mean*100,
sqrt(sum(fit.all[[2]]$aov.tab[3:4,"VC"]))/fit.all[[2]]$Mean*100,
sqrt(sum(fit.all[[3]]$aov.tab[3:4,"VC"]))/fit.all[[3]]$Mean*100,
sqrt(sum(fit.all[[4]]$aov.tab[3:4,"VC"]))/fit.all[[4]]$Mean*100,
sqrt(sum(fit.all[[5]]$aov.tab[3:4,"VC"]))/fit.all[[5]]$Mean*100,
sqrt(sum(fit.all[[6]]$aov.tab[3:4,"VC"]))/fit.all[[6]]$Mean*100),
Repro_SD=c(fit.all[[1]]$aov.tab["total","SD"],fit.all[[2]]$aov.tab["total","SD"],
fit.all[[3]]$aov.tab["total","SD"],fit.all[[4]]$aov.tab["total","SD"],
fit.all[[5]]$aov.tab["total","SD"],fit.all[[6]]$aov.tab["total","SD"]),
Repro_CV=c(fit.all[[1]]$aov.tab["total","CV[%]"],fit.all[[2]]$aov.tab["total","CV[%]"],
fit.all[[3]]$aov.tab["total","CV[%]"],fit.all[[4]]$aov.tab["total","CV[%]"],
fit.all[[5]]$aov.tab["total","CV[%]"],fit.all[[6]]$aov.tab["total","CV[%]"]))
```

```
for(i in 3:8) reproMat[,i] <- round(reproMat[,i],digits=ifelse(i%%2==0,1,3))
reproMat

# now plot the precision profile over all samples
plot(reproMat[,"Mean"], reproMat[,"Rep_CV"], type="l", main="Precision Profile CA19-9",
xlab="Mean CA19-9 Value", ylab="CV[%]")
grid()
points(reproMat[,"Mean"], reproMat[,"Rep_CV"], pch=16)

# REML-estimation not yes optimzed to the same degree as
# ANOVA-estimation. Note, that no variance-covariance matrix
# for the REML-fit is computed (VarVC=FALSE)!
# Note: A correct analysis would be done per-sample, this is just
#       for illustration.
data(VCAdata1)
# with complete sweeping implemented as FORTRAN-routine fit
system.time(fit0 <- anovaVCA(y~sample+(device+lot)/day/run, VCAdata1))
system.time(fit1 <- remlVCA(y~sample+(device+lot)/day/run, VCAdata1, VarVC=FALSE))

# The previous example will also be interesting for environments using MKL.
# Run it once in a GNU-R environment and once in a MKL-environment
# and compare computational time of both. Note, that 'VarVC' is now set to TRUE
# and variable "sample" is put into the brackets increasing the number of random
# effects by factor 10. On my Intel Xeon E5-2687W 3.1 GHz workstation it takes
# ~ 400s with GNU-R and ~25s with MKL support (MRO) both run under Windows.
system.time(fit2 <- remlVCA(y~(sample+device+lot)/day/run, VCAdata1, VarVC=TRUE))

# using the SWEEP-Operator is even faster
system.time(fit3 <- anovaVCA(y~(sample+device+lot)/day/run, VCAdata1))
fit2
fit3

## End(Not run)
```

ReproData1 *Multi-Site Data for Estimating Reproducibility Precision*

Description

This data set consists of real-world measurements of a multi-site study aiming at quantifying reproducibility precision. Unlike in the CLSI EP05-A3 guideline, there are two runs per day with three replicated measurements per run.

Usage

```
data(ReproData1)
```

Format

data.frame with 120 rows and 4 variables.

References

Approved Guideline CLSI EP05-A3 - Evaluation of Precision Performance of Quantitative Measurement Methods. CLSI

`reScale` *Re-Scale results of 'VCA' or 'VCAinference'*

Description

Function adjusts variance components (VC) and standard deviations (SD) and their respective confidence intervals of 'VCAinference' objects, and the 'VCAobj' sub-element. For 'VCA' objects the VC and SD values are adjusted as well as the fixed and random effects and the covariance-matrix of fixed effects.

Usage

```
reScale(obj, VarVC = TRUE)
```

Arguments

`obj`	(object) either of class 'VCA' or 'VCAinference'
`VarVC`	(logical) TRUE = variance-covariance matrix of the fitted model 'obj' will be computed and automatically re-scaled, FALSE = variance-covariance matrix will not be computed and re-scaled. This might cause wrong results in downstream analyses which require this matrix on the correct scale! Only use this option if computation time really matters!

Value

(object) either of class 'VCA' or 'VCAinference', where results have been transformed back to the original scale of the response variable

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

`Scale`

Examples

```
## Not run:
data(dataEP05A2_3)

# reference values
fit0 <- anovaVCA(y~day/run, dataEP05A2_3, MME=TRUE)
inf0 <- VCAinference(fit0, VarVC=TRUE)

fit1 <- Scale("anovaVCA", y~day/run, dataEP05A2_3, MME=TRUE)
inf1 <- VCAinference(fit1, VarVC=TRUE)
inf1 <- reScale(inf1)

# compare to reference
print(inf0, what="VC")
print(inf1, what="VC")
print(inf0, what="SD")
print(inf1, what="SD")
print(inf0, what="CV")
print(inf1, what="CV")

# now use REML-based estimation
fit0 <- remlVCA(y~day/run, dataEP05A2_3)
inf0 <- VCAinference(fit0)

fit1 <- Scale("remlVCA", y~day/run, dataEP05A2_3, MME=TRUE)
inf1 <- VCAinference(fit1)
inf1 <- reScale(inf1)

# compare to reference
print(inf0, what="VC")
print(inf1, what="VC")
print(inf0, what="SD")
print(inf1, what="SD")
print(inf0, what="CV")
print(inf1, what="CV")

## End(Not run)
```

`residuals.VCA` *Extract Residuals of a 'VCA' Object*

Description

Function extracts marginal or conditional residuals from a 'VCA' object, representing a linear mixed model.

Usage

```
## S3 method for class 'VCA'
```

```
residuals(
  object,
  type = c("conditional", "marginal"),
  mode = c("raw", "student", "standard", "pearson"),
  quiet = FALSE,
  ...
)
```

Arguments

object	(VCA) object
type	(character) string specifying the type of residuals to be returned, valid options are "marginal" and "conditional" or abbreviations
mode	(character) string or abbreviation specifying the specific transformation applied to a certain type of residuals. There are "raw" (untransformed), "standardized", "studentized" and "pearson" (see details) residuals.
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise
...	additional parameters

Details

There are two types of residuals which can be extraced from a 'VCA' object. Marginal residuals correspond to $e_m = y - \hat{y}$, where $\hat{y} = Xb$ with X being the design matrix of fixed effects and b being the column vector of fixed effects parameter estimates. Conditional residuals are defined as $e_c = y - Xb - Zg$, where Z corresponds to the designs matrix of random effects g. Whenever 'obj' is a pure-error model, e.g. 'y~1' both options will return the same values $y - Xb$ and b corresponds to the intercept. Each type of residuals can be standardized, studentized, or transformed to pearson-type residuals. The former corresponds to a transformation of residuals to have mean 0 and variance equal to 1 ($(r-\bar{r})/\sigma_r$). Studentized residuals emerge from dividing raw residuals by the square-root of diagonal elements of the corresponding variance-covariance matrix. For conditional residuals, this is $Var(c) = P = RQR$, with $Q = V^{-1}(I - H)$, $H = XT$ being the hat-matrix, and $T = (X^T V^{-1} X)^{-1} X^T V^{-1}$. For marginal residuals, this matrix is $Var(m) = O = V - Q$. Here, $>^T<$ denotes the matrix transpose operator, and $>^{-1}<$ the regular matrix inverse. Pearson-type residuals are computed in the same manner as studentized, only the variance-covariance matrices differ. For marginal residuals this is equal to $Var(y) = V$, for conditional residuals this is $Var(c) = R$ (see getV for details).

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Hilden-Minton, J. A. (1995). Multilevel diagnostics for mixed and hierarchical linear models. Dissertation, University of California, Los Angeles.

Nobre, J. S. & Singer, J. M. (2007). Residual analysis for linear mixed models. Biometrical Journal, 49, 863-875.

Schuetzenmeister, A. and Piepho, H.P. (2012). Residual analysis of linear mixed models using a simulation approach. Computational Statistics and Data Analysis, 56, 1405-1416

See Also

ranef, anovaVCA, anovaMM

Examples

```
## Not run:
data(VCAdata1)
datS1 <- VCAdata1[VCAdata1$sample==1,]
fit1  <- anovaVCA(y~(lot+device)/(day)/(run), datS1)

# default is conditional (raw) residuals
resid(fit1)
resid(fit1, "m")

# get standardized version
resid(fit1, mode="stand") # conditional residuals (default)
resid(fit1, "marg", "stand") # marginal residuals

# get studentized version, taking their
# covariances into account
resid(fit1, mode="stud") # conditional residuals (default)
resid(fit1, "marg", "stud") # marginal residuals

## End(Not run)
```

SattDF — *Satterthwaite Approximation for Total Degrees of Freedom and for Single Variance Components*

Description

This function estimates degrees of freedom of the total variance (type="total") in random models or individual variance components (type="individual"). It bases on the results of the unified approach to ANOVA-type estimation of variance components as implemented in functions anovaVCA and anovaMM.

Usage

```
SattDF(MS, Ci, DF, type = c("total", "individual"))
```

Arguments

MS	(numeric) vector of sequential mean squares (ANOVA type-1).
Ci	(matrix) where elements are numeric values representing the inverse of the coefficient matrix for calculation of expected mean squares (see anovaVCA).
DF	(numeric) vector with the degrees of freedom for each factor in a ANOVA type-1 model.
type	(character) string specifying whether "total" degrees of freedom should be approximated or those of individual variance components

Details

Function is used internally, thus, it is not exported. Option 'type="total"' is used in functions anovaVCA and anovaMM for approximating total DF. Option 'type="individual"' is used in function VCAinference when choosing 'ci.method="satterthwaite"' for approximating DFs for individual variance components.

Value

numeric value representing the Satterthwaite DFs of the total variance.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
\donttest{
data(dataEP05A2_2)
res <- anovaVCA(y~day/run, dataEP05A2_2)
VCA::SattDF(res$aov.tab[-1,"MS"], getMat(res, "Ci.MS"), res$aov.tab[-1,"DF"], type="tot")

# now approximating individual DF for variance components
VCA::SattDF(res$aov.tab[-1,"MS"], getMat(res, "Ci.MS"), res$aov.tab[-1,"DF"], type="i")
}
## End(Not run)
```

Scale *Automatically Scale Data Calling these Functions: 'anovaVCA', 'anovaMM', 'remlVCA' or 'remlMM'*

Description

This function scales data before fitting a linear mixed model aiming to avoid numerical problems when numbers of the response variable are either very small or very large. It adds attribute "scale" to the resulting 'VCA'-object, which is used by function reScale to transform back the VCA-results of a VCA or VCAinference object that was previously scaled.

Usage

```
Scale(Fun, form, Data, ...)
```

Arguments

Fun	(expr, function, character) either a complete function call to one of "anovaVCA", "anovaMM", "remlVCA", "remlMM", a character string or just the function name without quotes (see example)
form	(formula) specifying the model to fitted by 'Fun'
Data	(data.frame) with all variables specified via 'Fun'
...	additional arguments applying to one of the four functions anovaVCA,anovaMM, remlVCA, remlMM

Details

NOTE: Scaling is applied on the complete data set, without checking whether there are incomplete observations or not!

Value

(object) of class 'VCA' which can be used as input for function VCAinference

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

reScale

Examples

```
## Not run:
data(dataEP05A2_3)

# simulate very large numbers of the response
dat3   <- dataEP05A2_3
dat3$y <- dat3$y * 1e8

# now try to fit 21-day model to this data
fit <- anovaVCA(y~day/run, dat3)

# now use 'Scale' function
fit1 <- Scale("anovaVCA", y~day/run, dat3)
fit2 <- Scale(anovaVCA, y~day/run, dat3) # also works
fit3 <- Scale(anovaVCA(y~day/run, dat3)) # works as well

# back to original scale
(fit1 <- reScale(fit1))
(fit2 <- reScale(fit2))
```

```
(fit3 <- reScale(fit3))

# reference values
fit0 <- anovaVCA(y~day/run, dataEP05A2_3, MME=TRUE)
inf0 <- VCAinference(fit0, VarVC=TRUE)

fit1 <- Scale(anovaVCA(y~day/run, dataEP05A2_3, MME=TRUE))
inf1 <- VCAinference(fit1, VarVC=TRUE)
inf1 <- reScale(inf1)

# compare to reference
print(inf0, what="VC")
print(inf1, what="VC")
print(inf0, what="SD")
print(inf1, what="SD")
print(inf0, what="CV")
print(inf1, what="CV")

# now use REML-based estimation
fit0 <- remlVCA(y~day/run, dataEP05A2_3)
inf0 <- VCAinference(fit0)

fit1 <- Scale("remlVCA", y~day/run, dataEP05A2_3)
inf1 <- VCAinference(fit1)
inf1 <- reScale(inf1)

# compare to reference
print(inf0, what="VC")
print(inf1, what="VC")
print(inf0, what="SD")
print(inf1, what="SD")
print(inf0, what="CV")
print(inf1, what="CV")

# scaling also works with by-processing
data(VCAdata1)
fit <- Scale(anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample"))
reScale(fit)

## End(Not run)
```

`scaleData` *Scale Response Variable to Ensure Robust Numerical Calculations*

Description

Function determines scaling factor for transforming the mean of the response to a range between 0.1 and 1, applies scaling of the response and binds the scaling factor to the data as attribute 'scale'.

Usage

```
scaleData(Data = NULL, resp = NULL)
```

Arguments

Data	(data.frame) with the data to be fitted and the response to be scaled
resp	(character) name of the (numeric) response variable

Value

(data.frame) with the response scaled according to the scaling-factor, which is recorded in the attribute `scale` of the data set

Author(s)

Andre Schuetzenmeister <andre.schuetzenmester@roche.com>

---

sleepstudy *sleepstudy dataset from R-package 'lme4'*

Description

The average reaction time per day for subjects in a sleep deprivation study. On day 0 the subjects had their normal amount of sleep. Starting that night they were restricted to 3 hours of sleep per night. The observations represent the average reaction time on a series of tests given each day to each subject.

Usage

```
data(sleepstudy)
```

Format

A data frame with 180 observations on the following 3 variables.

- Reaction

  Average reaction time (ms)

- Days

  Number of days of sleep deprivation

- Subject

  Subject number on which the observation was made.

References

Gregory Belenky, Nancy J. Wesensten, David R. Thorne, Maria L. Thomas, Helen C. Sing, Daniel P. Redmond, Michael B. Russo and Thomas J. Balkin (2003) Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: a sleep dose-response study. Journal of Sleep Research 12, 1-12.

`Solve` *Solve System of Linear Equations using Inverse of Cholesky-Root*

Description

Function solves a system of linear equations, respectively, inverts a matrix by means of the inverse Cholesky-root.

Usage

```
Solve(X, quiet = FALSE)
```

Arguments

`X`	(matrix, Matrix) object to be inverted
`quiet`	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

This function is intended to reduce the computational time in function `solveMME` which computes the inverse of the square variance- covariance Matrix of observations. It is considerably faster than function `solve` (see example). Whenever an error occurs, which is the case for non positive definite matrices 'X', function `MPinv` is called automatically yielding a generalized inverse (Moore-Penrose inverse) of 'X'.

Value

(matrix, Matrix) corresponding to the inverse of X

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
# following complex (nonsense) model takes pretty long to fit
system.time(res.sw <- anovaVCA(y~(sample+lot+device)/day/run, VCAdata1))
# solve mixed model equations (not automatically done to be more efficient)
system.time(res.sw <- solveMME(res.sw))
# extract covariance matrix of observations V
V1 <- getMat(res.sw, "V")
```

```
V2 <- as.matrix(V1)
system.time(V2i <- solve(V2))
system.time(V1i <- VCA:::Solve(V1))
V1i <- as.matrix(V1i)
dimnames(V1i) <- NULL
dimnames(V2i) <- NULL
all.equal(V1i, V2i)

## End(Not run)
```

`solveMME` *Solve Mixed Model Equations*

Description

Function solves the Mixed Model Equations (MME) to estimate fixed and random effects.

Usage

```
solveMME(obj)
```

Arguments

`obj`	... (VCA) object

Details

This function is for internal use only, thus, not exported.

Value

(VCA) object, which has additional elements "RandomEffects" corresponding to the column vector of estimated random effects, "FixedEffects" being the column vector of estimated fixed effects. Element "Matrices" has additional elements referring to the elements of the MMEs and element "VarFixed" corresponds to the variance-covariance matrix of fixed effects.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(dataEP05A2_1)
fit <- anovaVCA(y~day/run, dataEP05A2_1, NegVC=TRUE)
fit <- solveMME(fit)
ranef(fit)

## End(Not run)
```

`stepwiseVCA` *Bottom-Up Step-Wise VCA-Analysis of the Complete Dataset*

Description

Function performs step-wise VCA-analysis on a fitted VCA-object by leaving out N-1 to 0 top-level variance components (VC).

Usage

```
stepwiseVCA(obj, VarVC.method = c("scm", "gb"))
```

Arguments

`obj`	(VCA) object representing the complete analysis
`VarVC.method`	(character) string specifying the algorithm to be used for estimating variance-covariance matrix of VCs (see `anovaMM` for details).

Details

This function uses the complete data to quantify sub-sets of variance components. In each step the current total variance is estimated by subtracting the sum of all left-out VCs from the total variance of the initial VCA object. Doing this guarantees that the contribution to the total variance which is due to left-out VCs is accounted for, i.e. it is estimated but not included/reported. The degrees of freedom (DFs) of the emerging total variances of sub-sets are determined using the Satterthwaite approximation. This is achieved by extracting the corresponding sub-matrix from the coefficient matrix C of the 'VCA' object, the sub-vector of ANOVA mean squares, and the sub-vector of degrees of freedom and calling function `SattDF` method="total".

This step-wise procedure starts one-level above error (repeatability) and ends at the level of the upper-most VC. It can only be used on models fitted by ANOVA Type-1, i.e. by function `anovaVCA`.

Value

(list) of (simplified) 'VCA' objects representing analysis-result of sub-models

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(VCAdata1)
datS7L1 <- VCAdata1[VCAdata1$sample == 7 & VCAdata1$lot == 1, ]
fit0 <- anovaVCA(y~device/day/run, datS7L1, MME=TRUE)

# complete VCA-analysis result
fit0
```

```
# perform step-wise (bottom-up) VCA-analyses
sw.res <- stepwiseVCA(fit0)
sw.res

# get CIs on intermediate precision
VCAinference(sw.res[["device:day"]])

## End(Not run)
```

---

summarize.VCA *Summarize Outcome of a Variance Component Analysis.*

Description

If a single 'VCA'-object is passed, the first step is to call 'VCAinference' for CI estimation. For each variance component (VC) the result of the VCA is summarized and can be configured by arguments 'type', 'tail', 'ends', and 'conf.level'. These define which information is returned by this summary function. In case of passing a list of 'VCA'- or 'VCAinference'-objects, a matrix will be returned where columns correspond to list-elements, usually samples, and rows to estimated values. This is done as the number of estimated values usually exceeds the number of samples.

Usage

```
summarize.VCA(
  object,
  type = c("sd", "cv"),
  tail = "one-sided",
  ends = "upper",
  conf.level = 0.95,
  DF = TRUE,
  as.df = FALSE,
  print = TRUE
)
```

Arguments

object	(object) of class VCA or VCAinference or a list of these objects to be summarized.
type	(character) "sd" for standard devation, "cv" for coefficient of variation, and "vc" for variance defining on which scale results shall be returned. Multiple can be specified.
tail	(character) "one-sided" for one-sided CI, "two-sided" for two-sided CI, can be abbreviated
ends	(character) "upper" or "lower" bounds of a e.g. 95% CI, can be both
conf.level	(numeric) confidence level of the CI

DF	(logical) TRUE to include degrees of freedom, FALSE to omit them
as.df	(logical) TRUE to transpose the returned object and convert into a data.frame, FALSE leve
print	(logical) TRUE print summary, FALSE omit printing and just return matrix or data.frame

Value

(matrix, data.frame) with VCA-results either with estimates in rows and sample(s) in columns, or vice versa

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:
data(CA19_9)
fit.all <- anovaVCA(result~site/day, CA19_9, by="sample")
summarize.VCA(fit.all)
# complete set of results
summarize.VCA( fit.all, type=c("vc", "sd", "cv"), tail=c("one", "two"),
ends=c("lower", "upper"))
# summarizing a single VCA-object
summarize.VCA(fit.all[[1]])

### summarizing list of 'VCAinference' objects
infs <- VCAinference(fit.all)
summarize.VCAinference(infs)

## End(Not run)
```

---

test.fixef *Perform t-Tests for Linear Contrasts on Fixed Effects*

Description

This function performs t-Tests for one or multiple linear combinations (contrasts) of estimated fixed effects.

Usage

```
test.fixef(
  obj,
  L,
  ddfm = c("contain", "residual", "satterthwaite"),
  method.grad = "simple",
```

```
  tol = 1e-12,
  quiet = FALSE,
  opt = TRUE,
  onlyDF = FALSE,
  ...
)
```

Arguments

obj	(VCA) object
L	(numeric) vector or matrix, specifying linear combinations of the fixed effects, in the latter case, each line represents a disctinct linear contrast
ddfm	(character) string specifying the method used for computing the denominator degrees of freedom for tests of fixed effects or LS Means. Available methods are "contain", "residual", and "satterthwaite".
method.grad	(character) string specifying the method to be used for approximating the gradient of the variance-covariance matrix of fixed effects at the estimated covariance parameter estimates (see function 'grad' (numDeriv) for details)
tol	(numeric) value specifying the numeric tolerance for testing equality to zero
quiet	(logical) TRUE = suppress warning messages, e.g. for non-estimable contrasts
opt	(logical) TRUE = tries to optimize computation time by avoiding unnecessary computations for balanced datasets (see details).
onlyDF	(logical) TRUE = only the specified type of degrees of freedom are determined without carrying out the actual hypothesis test(s)
...	further parameters (for internal use actually)

Details

Here, the same procedure as in SAS PROC MIXED ddfm=satterthwaite (sat) is implemented. This implementation was inspired by the code of function 'calcSatterth' of R-package 'lmerTest'. Thanks to the authors for this nice implementation.
Note, that approximated Satterthwaite degrees of freedom might differ from 'lmerTest' and SAS PROC MIXED. Both use the inverse Fisher-information matrix as approximation of the variance-covariance matrix of variance components (covariance parameters). Here, either the exact algorithm for ANOVA-estimators of variance components, described in Searle et. al (1992) p. 176, or the approximation presented in Giesbrecht and Burns (19985) are used. For balanced designs their will be no differences, usually. In case of balanced designs, the Satterthwaite approximation is equal to the degrees of freedom of the highest order random term in the model (see examples).

Value

(numeric) vector or matrix with 4 elements/columns corresponding to "Estimate", "t Value", "DF", and "Pr > |t|".

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com> inspired by authors of R-package 'lmerTest'

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

SAS Help and Documentation PROC MIXED (MODEL-statement, Option 'ddfm'), SAS Institute Inc., Cary, NC, USA

See Also

test.lsmeans, getL

Examples

```
## Not run:
data(dataEP05A2_2)
ub.dat <- dataEP05A2_2[-c(11,12,23,32,40,41,42),]
fit1 <- anovaMM(y~day/(run), ub.dat)
fit2 <- remlMM(y~day/(run), ub.dat)
fe1 <- fixef(fit1)
fe1
fe2 <- fixef(fit2)
fe2
lc.mat <- getL( fit1, c("day1-day2", "day3-day6"))
lc.mat
test.fixef(fit1, lc.mat, ddfm="satt")
test.fixef(fit2, lc.mat, ddfm="satt")

# some inferential statistics about fixed effects estimates
L <- diag(nrow(fe1))
rownames(L) <- colnames(L) <- rownames(fe1)
test.fixef(fit1, L)
test.fixef(fit2, L)

# using different "residual" method determining DFs
test.fixef(fit1, L, ddfm="res")
test.fixef(fit2, L, ddfm="res")

# having 'opt=TRUE' is a good idea to save time
# (in case of balanced designs)
data(VCAdata1)
datS3 <- VCAdata1[VCAdata1$sample==3,]
fit3 <- anovaMM(y~(lot+device)/(day)/run, datS3)
fit4 <- remlMM(y~(lot+device)/(day)/run, datS3)
fit3$VarCov <- vcovVC(fit3)
fe3 <- fixef(fit3)
fe4 <- fixef(fit4)
L <- diag(nrow(fe3))
rownames(L) <- colnames(L) <- rownames(fe3)
system.time(tst1 <- test.fixef(fit3, L))
system.time(tst2 <- test.fixef(fit3, L, opt=FALSE))
system.time(tst3 <- test.fixef(fit4, L, opt=FALSE))
```

```
tst1
tst2
tst3

## End(Not run)
```

`test.lsmeans` *Perform t-Tests for Linear Contrasts on LS Means*

Description

Perform custom hypothesis tests on Least Squares Means (LS Means) of fixed effect.

Usage

```
test.lsmeans(
  obj,
  L,
  ddfm = c("contain", "residual", "satterthwaite"),
  quiet = FALSE
)
```

Arguments

`obj`	(VCA) object
`L`	(matrix) specifying one or multiple custom hypothesis tests as linear contrasts of LS Means. Which LS Means have to be used is inferred from the column names of matrix L, which need to be in line with the naming of LS Means in function `lsmeans`.
`ddfm`	(character) string specifying the method used for computing the denominator degrees of freedom of t-tests of LS Means. Available methods are "contain", "residual", and "satterthwaite".
`quiet`	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

This function is similar to function `test.fixef` and represents a convenient way of specifying linear contrasts of LS Means.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

See Also

`test.fixef`, `lsmeans`

Examples

```
## Not run:
data(dataEP05A2_2)
ub.dat <- dataEP05A2_2[-c(11,12,23,32,40,41,42),]
fit1 <- anovaMM(y~day/(run), ub.dat)
fit2 <- remlMM(y~day/(run), ub.dat)
lsm1 <- lsmeans(fit1)
lsm2 <- lsmeans(fit2)
lsm1
lsm2

lc.mat <- getL(fit1, c("day1-day2", "day3-day6"), "lsm")
lc.mat[1,c(1,2)] <- c(1,-1)
lc.mat[2,c(3,6)] <- c(1,-1)
lc.mat
test.lsmeans(fit1, lc.mat)
test.lsmeans(fit2, lc.mat)

# fit mixed model from the 'nlme' package

library(nlme)
data(Orthodont)
fit.lme <- lme(distance~Sex*I(age-11), random=~I(age-11)|Subject, Orthodont)

# re-organize data for using 'anovaMM'
Ortho <- Orthodont
Ortho$age2 <- Ortho$age - 11
Ortho$Subject <- factor(as.character(Ortho$Subject))

# model without intercept
fit.anovaMM <- anovaMM(distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)
fit.remlMM1 <- remlMM( distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho)
fit.remlMM2 <- remlMM( distance~Sex+Sex:age2+(Subject)+(Subject):age2-1, Ortho, cov=FALSE)
lsm0 <- lsmeans(fit.anovaMM)
lsm1 <- lsmeans(fit.remlMM1)
lsm2 <- lsmeans(fit.remlMM2)
lsm0
lsm1
lsm2

lc.mat <- matrix(c(1,-1), nrow=1, dimnames=list("int.Male-int.Female", c("SexMale", "SexFemale")))
lc.mat
test.lsmeans(fit.anovaMM, lc.mat)
test.lsmeans(fit.remlMM1, lc.mat)
test.lsmeans(fit.remlMM2, lc.mat)

## End(Not run)
```

`Trace` *Compute the Trace of a Matrix*

Description

Function computes the sum of main-diagonal elements of a square matrix.

Usage

```
Trace(x, quiet = FALSE)
```

Arguments

x	(matrix, Matrix) object
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise

Value

(numeric) value, the trace of the matrix

---

varPlot *Variability Chart for Hierarchical Models.*

Description

Function varPlot determines the sequence of variables in the model formula and uses this information to construct the variability chart.

Usage

```
varPlot(
  form,
  Data,
  keep.order = TRUE,
  type = c(1L, 2L, 3L)[1],
  VARtype = "SD",
  htab = 0.5,
  Title = NULL,
  VSpace = NULL,
  VarLab = list(cex = 0.75, adj = c(0.5, 0.5)),
  YLabel = list(text = "Value", side = 2, line = 3.5, cex = 1.5),
  SDYLabel = list(side = 2, line = 2.5),
  Points = list(pch = 16, cex = 0.5, col = "black"),
  SDs = list(pch = 16, col = "blue", cex = 0.75),
  SDline = list(lwd = 1, lty = 1, col = "blue"),
  BG = list(border = "lightgray", col.table = FALSE),
  VLine = list(lty = 1, lwd = 1, col = "gray90"),
  HLine = NULL,
  Join = list(lty = 1, lwd = 1, col = "gray"),
  JoinLevels = NULL,
  Mean = list(pch = 3, col = "red", cex = 0.5),
```

```
  MeanLine = NULL,
  Boxplot = NULL,
  VCnam = list(cex = 0.75, col = "black", line = 0.25),
  useVarNam = FALSE,
  ylim = NULL,
  max.level = 25,
  ...
)
```

Arguments

form	(formula) object specifying the model, NOTE: any crossed factors are reduced to last term of the crossing structure, i.e. "a:b" is reduced to "b", "a:b:c" is reduced to "c".
Data	(data.frame) with the data
keep.order	(logical) TRUE = the ordering of factor-levels is kept as provided by 'Data', FALSE = factor-levels are sorted on and within each level of nesting.
type	(integer) specifying the type of plot to be used, options are 1 = regular scatter-plot, 2 = plot of the standard deviation, 3 = both type of plots.
VARtype	(character) either "SD" (standard deviation) or "CV" (coefficient of variation), controls which type of measures is used to report variability in plots when 'type' is set to either 2 or (see 'type' above). Note that all parameters which apply to the SD-plot will be used for the CV-plot in case 'VARtype="CV"'.
htab	(numeric) value 0 < htab < 1 specifying the height of the table representing the experimental design. This value represents the proportion in relation to the actual plotting area, i.e. htab=1 mean 50% of the vertical space is reserved for the table.
Title	(list) specifying all parameters applicable in function title for printing main- or sub-titles to plots. If 'type==3', these settings will apply to each plot. For individual settings specify a list with two elements, where each element is a list itself specifying all parameters of function 'title'. The first one is used for the variability chart, the second one for the SD or CV plot. Set to NULL to omit any titles.
VSpace	(numeric) vector of the same length as there are variance components, specifying the proportion of vertical space assigned to each variance component in the tabular indicating the model structure. These elements have to sum to 1, otherwise equal sizes will be used for each VC.
VarLab	(list) specifying all parameters applicable in function text, used to add labels within the table environment refering to the nesting structure. This can be a list of lists, where the i-th list corresponds to the i-th variance component, counted in bottom-up direction, i.e. starting from the most general variance component ('day' in the 1st example).
YLabel	(list) specifying all parameters applicable in function mtext, used for labelling the Y-axis.
SDYLabel	(list) specifying all parameters applicable in function mtext, used for labelling the Y-axis.

`Points` (list) specifying all parameters applicable in function `points`, used to specify scatterplots per lower-end factor-level (e.g. 'run' in formula run/day). If list-elements "col", "pch", "bg" and "cex" are lists themselves with elements "var" and "col"/"pch"/"bg"/"cex", where the former specifies a variable used for assigning colors/symbols/backgrounds/sizes according to the class-level of variable "var", point-colors/plotting-symbols/plotting-symbol backgrounds/plotting-symbol sizes can be used for indicating specific sub-classes not addressed by the model/design or indicate any sort of information (see examples). Note the i-th element of 'col'/'pch' refers of the i-th element of unique(Data$var), even if 'var' is an integer variable.

`SDs` (list) specifying all parameters applicable in function `points`, used to specify the appearance of SD-plots.

`SDline` (list) specifying all parameters applicable in function `lines`, used to specify the (optional) line joining individual SDs, Set to NULL to omit.

`BG` (list) specifying the background for factor-levels of a nested factor. This list is passed on to function `rect` after element 'var', which identifies the factor to be used for coloring, has been removed. If not set to NULL and no factor has been specified by the user, the top-level factor is selected by default. If this list contains element 'col.table=TRUE', the same coloring schema is used in the table below at the corresponding row/factor (see examples). Addionally, list-elment 'col.bg=FALSE' can be used to turn off BG-coloring, e.g. if only the the respective row in the table below should be color-coded (defaults to 'col.bg=TRUE'). When specifying as many colors as there are factor-levels, the same color will be applied to a factor-level automatically. This is relevant for factors, which are not top-level (bottom in the table). Example: BG=list(var="run", col=c("white", "lightgray"), border=NA) draws the background for alternating levels of factor "run" white and gray for better visual differentiation. Set to NULL to omit. Use list(..., col="white", border="gray") for using gray vertical lines for separation. See argument 'VLine' for additional highlighting options of factor-levels.

`VLine` (list) specifying all parameters applicable in `lines` optionally separating levels of one or multiple variables as vertical lines. This is useful in addition to 'BG' (see examples), where automatically 'border=NA' will be set that 'VLine' will take full effect. If this list contains element 'col.table=TRUE', vertical lines will be extended to the table below the plot.

`HLine` (list) specifying all parameters applicable in function `abline` to add horizontal lines. Only horizontal lines can be set specifying the 'h' parameter. 'HLine=list()' will use default settings. 'HLine=NULL' will omit horizontal lines. In case 'type=3', two separate lists can be specified where the first list applies to the variability chart and the second list to the SD-/CV-chart.

`Join` (list) specifying all parameter applicable in function `lines` controlling how observed values within lower-level factor-levels, are joined. Set to NULL to omit.

`JoinLevels` (list) specifying all arguments applicable in function `lines`, joining factor-levels nested within higher order factor levels, list-element "var" specifies this variable

`Mean` (list) passed to function `points` specifying plotting symbols used to indicate mean values per lower-level factor-level, set equal to NULL to omit.

`MeanLine` (list) passed to function `lines` specifying the appearance of horizontal lines indicating mean values of factor levels. The factor variable for which mean-values of factor-levels are plotted can be specified via list-element "var" accepting any factor variable specified in 'form'. List element "mar" takes values in [0;.5] setting the left and right margin size of mean-lines. Set equal to NULL to omit. Use 'var="int"' for specifying the overall mean (grand mean, intercept). If this list contains logical 'join' which is set to TRUE, these mean lines will be joined. If list-element "top" is set to TRUE, these lines will be plotted on top, which is particularily useful for very large datasets.

`Boxplot` (list) if not NULL, a boxplot of all values within the smallest possible subgroup (replicates) will be added to the plot, On can set list-elements 'col.box="gray65"', 'col.median="white"', 'col.whiskers="gray65"' specifying different colors and 'lwd=3' for the line width of the median-line and whiskers-lines as well as 'jitter=1e3' controlling the jittering of points around the center of the box in horizontal direction, smallest possible value is 5 meaning the largest amount of jittering (1/5 in both directions) value is)

`VCnam` (list) specifying the text-labels (names of variance components) appearing as axis-labels. These parameters are passed to function `mtext`. Parameter 'side' can only be set to 2 (left) or 4 (right) controlling where names of variance components appear. Set to NULL to omit VC-names.

`useVarNam` (logical) TRUE = each factor-level specifier is pasted to the variable name of the current variable and used as list-element name, FALSE = factor-level specifiers are used as names of list-elements; the former is useful when factor levels are indicated as integers, e.g. days as 1,2,..., the latter is useful when factor levels are already unique, e.g. day1, day2, ...

`ylim` (numeric) vector of length two, specifying the limits in Y-direction, if not set these values will be determined automatically. In case of plot 'type=3' this can also be a list of two ylim-vectors, first corresponding to the variability chart, second to the plot of error variability per replicate group

`max.level` (integer) specifying the max. number of levels of a nested factor in order to draw vertical lines. If there are too many levels a black area will be generated by many vertical lines. Level names will also be omitted.

`...` further graphical parameters passed on to function 'par', e.g. use 'mar' for specification of margin widths. Note, that not all of them will have an effect, because some are fixed ensuring that a variability chart is drawn.

Details

This function implements a variability-chart, known from, e.g. JMP (JMP, SAS Institute Inc., Cary, NC). Arbitrary models can be specified via parameter 'form'. Formulas will be reduced to a simple hierarchical structure ordering factor-variables according to the order of appearance in 'form'. This is done to make function `varPlot` applicable to any random model considered in this package. Even if there are main factors, neither one being above or below another main factor, these are forced into a hierachy. Besides the classic scatterplot, where observations are plotted in sub-classes emerging from the model formula, a plot of standard deviations (SD) or coefficients of variation (CV) is provided (type=2) or both types of plots together (type=3).

Value

(invisibly) returns 'Data' with additional variable 'Xcoord' giving X-coordinates of each observation

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

Examples

```
## Not run:

# load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_3)

# two additional classification variables (without real interpretation)
dataEP05A2_3$user <- sample(rep(c(1,2), 40))
dataEP05A2_3$cls2 <- sample(rep(c(1,2), 40))

# plot data as variability-chart, using automatically determined parameter
# settings (see 'dynParmSet')
varPlot(y~day/run, dataEP05A2_3)

# display intercept (total mean)
varPlot(y~day/run, dataEP05A2_3, MeanLine=list(var="int"))

# use custom VC-names
varPlot(y~day/run, dataEP05A2_3, VCnam=list(text=c("_Day", "_Run")))

# re-plot now also indicating dayly means as blue horizontal lines
varPlot(y~day/run, dataEP05A2_3, MeanLine=list(var=c("day", "int"), col="blue"))

# now use variable-names in names of individual factor-levels and use a different
# notation of the nesting structure
varPlot(y~day+day:run, dataEP05A2_3, useVarNam=TRUE)

# rotate names of VCs to fit into cells
varPlot( y~day+day:run, dataEP05A2_3, useVarNam=TRUE,
VarLab=list(list(font=2, srt=60), list(srt=90)))

# use alternating backgrounds for each level of factor "day"
# (top-level factor is default)
# use a simplified model formula (NOTE: only valid for function 'varPlot')
varPlot(y~day+run, dataEP05A2_3, BG=list(col=c("gray70", "gray90"), border=NA))

# now also color the corresponding row in the table accordingly
varPlot( y~day+run, dataEP05A2_3,
         BG=list(col=c("gray70", "gray90"), border=NA, col.table=TRUE))

# assign different point-colors according to a classification variable
# not part of the model (artificial example in this case)
varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,7), VCnam=list(side=4),
```

```
         Points=list(col=list(var="user", col=c("red", "green"))) )

# always check order of factor levels before annotating
order(unique(dataEP05A2_3$user))

# add legend to right margin
legend.m(fill=c("green", "red"), legend=c("User 1", "User 2"))

# assign different plotting symbols according to a classification
# variable not part of the model
varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,7), VCnam=list(side=4),
         Points=list(pch=list(var="user", pch=c(2, 8))) )

# add legend to right margin
legend.m(pch=c(8,2), legend=c("User 1", "User 2"))

# assign custom plotting symbols by combining 'pch' and 'bg'
varPlot( y~day+day:run, dataEP05A2_3,
         Points=list(pch=list(var="user", pch=c(21, 24)),
                     bg=list( var="user", bg=c("lightblue", "yellow"))) )

# assign custom plotting symbols by combining 'pch', 'bg', and 'cex'
varPlot( y~day+day:run, dataEP05A2_3,
         Points=list(pch=list(var="user", pch=c(21, 24)),
                     bg =list(var="user", bg=c("lightblue", "yellow")),
                     cex=list(var="user",  cex=c(2,1))) )

# now combine point-coloring and plotting symbols
# to indicate two additional classification variables
varPlot( y~day+day:run, dataEP05A2_3, mar=c(1,5,1,10),
         VCnam=list(side=4, cex=1.5),
         Points=list(col=list(var="user", col=c("red", "darkgreen")),
                     pch=list(var="cls2", pch=c(21, 22)),
                     bg =list(var="user", bg =c("orange", "green"))) )

# add legend to (right) margin
 legend.m( margin="right", pch=c(21, 22, 22, 22),
           pt.bg=c("white", "white", "orange", "green"),
           col=c("black", "black", "white", "white"),
           pt.cex=c(1.75, 1.75, 2, 2),
           legend=c("Cls2=1", "Cls2=2", "User=2", "User=1"),
           cex=1.5)

# use blue lines between each level of factor "run"
varPlot(y~day/run, dataEP05A2_3, BG=list(var="run", border="blue"))

# plot SDs for each run
varPlot(y~day+day:run, dataEP05A2_3, type=2)

# use CV instead of SD
varPlot(y~day/run, dataEP05A2_3, type=2, VARtype="CV")

# now plot variability-chart and SD-plot in one window
```

```
varPlot(y~day/run, dataEP05A2_3, type=3, useVarNam=TRUE)

# now further customize the plot
varPlot( y~day/run, dataEP05A2_3, BG=list(col=c("lightgray", "gray")),
         YLabel=list(font=2, col="blue", cex=1.75, text="Custom Y-Axis Label"),
         VCnam=list(col="red", font=4, cex=2),
         VarLab=list(list(col="blue", font=3, cex=2), list(cex=1.25, srt=-15)))

# create variability-chart of the example dataset in the CLSI EP05-A2
# guideline (listed on p.25)
data(Glucose,package="VCA")
varPlot(result~day/run, Glucose, type=3)

# use individual settings of 'VarLab' and 'VSpace' for each variance component
varPlot(result~day/run, Glucose, type=3,
        VarLab=list(list(srt=45, col="red", font=2),
        list(srt=90, col="blue", font=3)), VSpace=c(.25, .75))

# set individual titles for both plot when 'type=3'
# and individual 'ylim' specifications
 varPlot(result~day/run, Glucose, type=3,
Title=list( list(main="Variability Chart"),
list(main="Plot of SD-Values")),
ylim=list( c(230, 260), c(0, 10)))

# more complex experimental design
data(realData)
Data <- realData[realData$PID == 1,]
varPlot(y~lot/calibration/day/run, Data, type=3)

# order levels in the tablular environment
varPlot(y~lot/calibration/day/run, Data, keep.order=FALSE)
# keeping the order as in the data set (default) was different
varPlot(y~lot/calibration/day/run, Data, keep.order=TRUE)

# improve visual appearance of the plot by alternating bg-colors
# for variable "calibration"
varPlot(y~lot/calibration/day/run, Data, type=3, keep.order=FALSE,
        BG=list(var="calibration", col=c("white", "lightgray")))

# add horizontal lines indicating mean-value for each factor-level of all variables
varPlot(y~lot/calibration/day/run, Data, type=3, keep.order=FALSE,
        BG=list(var="calibration",
                col=c("lightgray","antiquewhite2","antiquewhite4",
                      "antiquewhite1","aliceblue","antiquewhite3",
                      "white","antiquewhite","wheat" ),
                col.table=TRUE),
        MeanLine=list(var=c("lot", "calibration", "day", "int"),
                      col=c("orange", "blue", "green", "magenta"),
                      lwd=c(2,2,2,2)))

# now also highlight bounds between factor levels of "lot" and "day"
# as vertical lines and extend them into the table (note that each
```

```
# variable needs its specific value for 'col.table')
 varPlot(y~lot/calibration/day/run, Data, type=3, keep.order=FALSE,
BG=list(var="calibration",
col=c( "aquamarine","antiquewhite2","antiquewhite4",
"antiquewhite1","aliceblue","antiquewhite3",
                      "white","antiquewhite","wheat" ),
col.table=TRUE),
MeanLine=list( var=c("lot", "calibration", "day", "int"),
col=c("orange", "blue", "darkgreen", "magenta"),
lwd=c(2,2,2,2)),
VLine=list( var=c("lot", "day"), col=c("black", "skyblue1"),
lwd=c(2, 1), col.table=c(TRUE, TRUE)))

# one can use argument 'JoinLevels' to join factor-levels or a variable
# nested within a higher-level factor, 'VLine' is used to separate levels
# of variables "calibration" and "lot" with different colors
 varPlot(y~calibration/lot/day/run, Data,
         BG=list(var="calibration",
                 col=c("#f7fcfd","#e5f5f9","#ccece6","#99d8c9",
                       "#66c2a4","#41ae76","#238b45","#006d2c","#00441b"),
                 col.table=TRUE),
         VLine=list(var=c("calibration", "lot"),
                    col=c("black", "darkgray"), lwd=c(2,1), col.table=TRUE),
         JoinLevels=list(var="lot", col=c("#ffffb2","orangered","#feb24c"),
                         lwd=c(2,2,2)),
         MeanLine=list(var="lot", col="blue", lwd=2))

# same plot demonstrating additional features applicable via 'Points'
 varPlot(y~calibration/lot/day/run, Data,
         BG=list(var="calibration",
                 col=c("#f7fcfd","#e5f5f9","#ccece6","#99d8c9",
                       "#66c2a4","#41ae76","#238b45","#006d2c","#00441b"),
                 col.table=TRUE),
         VLine=list(var=c("calibration", "lot"),
                    col=c("black", "mediumseagreen"), lwd=c(2,1),
                    col.table=c(TRUE,TRUE)),
         JoinLevels=list(var="lot", col=c("lightblue", "cyan", "yellow"),
                         lwd=c(2,2,2)),
         MeanLine=list(var="lot", col="blue", lwd=2),
         Points=list(pch=list(var="lot", pch=c(21, 22, 24)),
                     bg =list(var="lot", bg=c("lightblue", "cyan", "yellow")),
                     cex=1.25))

# depict measurements as boxplots
data(VCAdata1)
datS5 <- subset(VCAdata1, sample==5)
varPlot(y~device/day, datS5, Boxplot=list())

# present points as jitter-plot around box-center
varPlot(y~device/day, datS5,
Boxplot=list(jitter=1, col.box="darkgreen"),
BG=list(var="device", col=paste0("gray", c(60, 70, 80)),
col.table=TRUE),
```

```
Points=list(pch=16,
col=list(var="run", col=c("blue", "red"))),
Mean=list(col="black", cex=1, lwd=2),
VLine=list(var="day", col="white"))
# add legend
legend( "topright", legend=c("run 1", "run 2"),
        fill=c("blue", "red"), box.lty=0, border="white")

## End(Not run)
```

VCAdata1 *Simulated Data for Variance Component Analysis.*

Description

This data set consists of 2520 observations. There are 3 lots (lot), 10 samples, 21 days, 2 runs within day. This simulated dataset is used in examples and unit-tests (see subdir 'UnitTests' of the package-dir).

Usage

```
data(VCAdata1)
```

Format

data.frame with 2520 rows and 5 variables.

VCAinference *Inferential Statistics for VCA-Results*

Description

Function VCAinference constructs one- and two-sided confidence intervals, and performs Chi-Squared tests for total and error variance against claimed values for 'VCA' objects.

Usage

```
VCAinference(
  obj,
  alpha = 0.05,
  total.claim = NA,
  error.claim = NA,
  claim.type = "VC",
  VarVC = FALSE,
  excludeNeg = TRUE,
  constrainCI = TRUE,
  ci.method = "sas",
  quiet = FALSE
)
```

Arguments

`obj`	(object) of class 'VCA' or, alternatively, a list of 'VCA' objects, where all other arguments can be specified as vectors, where the i-th vector element applies to the i-th element of 'obj' (see examples)
`alpha`	(numeric) value specifying the significance level for $100 * (1 - alpha)\%$ confidence intervals.
`total.claim`	(numeric) value specifying the claim-value for the Chi-Squared test for the total variance (SD or CV, see `claim.type`).
`error.claim`	(numeric) value specifying the claim-value for the Chi-Squared test for the error variance (SD or CV, see `claim.type`).
`claim.type`	(character) one of "VC", "SD", "CV" specifying how claim-values have to be interpreted: "VC" (Default) = claim-value(s) specified in terms of variance(s), "SD" = claim-values specified in terms of standard deviations (SD), "CV" = claim-values specified in terms of coefficient(s) of variation (CV) and are specified as percentages. If set to "SD" or "CV", claim-values will be converted to variances before applying the Chi-Squared test (see examples).
`VarVC`	(logical) TRUE = the covariance matrix of the estimated VCs will be computed (see `vcovVC`), where diagonal elements correspond to the variances of the individual VCs. This matrix is required for estimation of CIs for intermediate VCs if 'method.ci="sas"'. FALSE (Default) = computing covariance matrix of VCs is omitted, as well as CIs for intermediate VCs.
`excludeNeg`	(logical) TRUE = confidence intervals of negative variance estimates will not be reported. FALSE = confidence intervals for all VCs will be reported including those with negative VCs. See the details section for a thorough explanation.
`constrainCI`	(logical) TRUE = CI-limits for all variance components are constrained to be >= 0. FALSE = unconstrained CIs with potentially negative CI-limits will be reported. which will preserve the original width of CIs. See the details section for a thorough explanation.
`ci.method`	(character) string or abbreviation specifying which approach to use for computing confidence intervals of variance components (VC). "sas" (default) uses Chi-Squared based CIs for total and error and normal approximation for all other VCs (Wald-limits, option "NOBOUND" in SAS PROC MIXED); "satterthwaite" will approximate DFs for each VC using the Satterthwaite approach (see `SattDF` for models fitted by ANOVA) and all Cis are based on the Chi-Squared distribution. This approach is conservative but avoids negative values for the lower bounds.
`quiet`	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

This function computes confidence intervals (CI) for variance components (VC), standard deviations (SD) and coefficients of variation (CV). VCs 'total' and 'error' can be tested against claimed

values specifying parameters 'total.claim' and 'error.claim'. One can also specify claim-values in terms of SD or CV (see `claim.type`).
Confidence intervals for VCs are constructed either following the same rules as in SAS 9.2 PROC MIXED with option 'method=type1' (ci.method="sas") or using Satterthwaite methodology throughout (ci.method="satterthwaite"). In the former approach for VC total and error, which are constrained to be $>= 0$, CIs are based on the Chi-Squared distribution. Degrees of freedom (DF) for total variance are approximated using the Satterthwaite approximation (which is not available in either SAS procedure). For all other VCs, the CI is $[sigma^2 - QNorm(alpha/2) * SE(sigma^2); sigma^2 + QNorm(1 - alpha/2) * SE(sigma^2)]$, where QNorm(x) indicates the x-quantile of the standard normal distribution. The second method approximates DFs for all VCs using the Satterthwaite approximation and CIs are based on the corresponding Chi-Squared distribution for all VCs (see examples). Note that in the computation of the covariance-matrix of the VCs, the estimated VCs will be used. If these are requested to be set to 0 (`NegVC=FALSE` in `anovaVCA`), the result might not be conformable with theory given in the first reference. The validity of this implementation was checked against SAS 9.2 PROC MIXED (method=type1), where VCs are not constrained to be >= 0. The sampling variances for VCs are obtained assuming normality throughout based on $Var(\sigma^2 = C^{-1} * Var(m_{SS} * (C^{-1})^T))$, where C^{-1} is the inverse of the coefficient matrix equating observed Sum of Squares (SS) to their expected values, and $(C^{-1})^T$ indicating the transpose of C^{-1} (see Searle et al. 1992, pg. 176).

An input `VCA`-object can be in one of three states:

State (1) corresponds to the situation, where all VC > 0.
State (2) corresponds to the situation, where at least one VC < 0.
State (3) corresponds to situations, where negative VC estimates occured but were set to 0, i.e. `NegVC=FALSE` - the Default.

State (2) occurs when parameter `NegVC` was set to TRUE in `anovaVCA`, state (3) represents the default-setting in function `anovaVCA`. If a `VCA`-object is in state (1), parameter `excludeNeg` has no effect (there are no negative VCs), only parameter `constrainCI` is evaluated. For `VCA`-objects in state(2), `constrainCI` has no effect, because constraining CIs for unconstrained VCs makes no sense. State (3) forces parameter `constrainCI` to be set to TRUE and one can only choose whether to exclude CIs of negative VC estimates or not. Whenever VCs have to be constrained, it is straight forward to apply constraining also to any CI. Note that situations outlined above only occur when parameter `VarVC` is set to TRUE, which causes estimation of the covariance-matrix of variance components. The default is only to compute and report CIs for total and error variance, which cannot become negative.

Value

(VCAinference) object, a list with elements:

`ChiSqTest`	(data.frame) with results of the Chi-Squared test
`ConfInt`	(list) with elements `VC`, `SD`, `CV`, all lists themselves containing (data.frame) objects `OneSided` and `TwoSided`
`VCAobj`	(VCA) object specified as input, if `VarVC=TRUE`, the 'aov.tab' element will have an extra column "Var(VC)" storing variances of VC-estimates"

Note

Original CIs will always be available independent of parameter-settings of `excludeNeg` and `constrainCI`.
Original CIs are stored in attribute "CIoriginal" of the returned 'VCAinference'-object, e.g. 'attr(obj$ConfInt$SD$OneSided, "CIoriginal")' or 'attr(obj$ConfInt$CV$TwoSided, "CIoriginal")'.

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components., Wiley New York

Burdick, R., Graybill, F. (1992), Confidence Intervals on Variance Components. Marcel Dekker, Inc.

Satterthwaite, F.E. (1946), An Approximate Distribution of Estimates of Variance Components., Biometrics Bulletin 2, 110-114

See Also

print.VCAinference, anovaVCA

Examples

```
## Not run:

# load data (CLSI EP05-A2 Within-Lab Precision Experiment)
data(dataEP05A2_1)

# perform (V)variance (C)component (A)nalysis (also compute A-matrices)
res <- anovaVCA(y~day/run, dataEP05A2_1)

# get confidence intervals for total and error (VC, SD, CV)
VCAinference(res)

# additionally request CIs for all other VCs; default is to constrain
# CI-limits to be >= 0
# first solve MME
res <- solveMME(res)
VCAinference(res, VarVC=TRUE)

# now using Satterthwaite methodology for CIs
VCAinference(res, VarVC=TRUE, ci.method="satt")

# request unconstrained CIs
VCAinference(res, VarVC=TRUE, constrainCI=FALSE)

# additionally request Chi-Squared Tests of total and error, default
# is that claim values are specified as variances (claim.type="VC")
VCAinference(res, total.claim=4.5, error.claim=3.5)
```

```
# perform Chi-Squared Tests, where claim-values are given as SD,
# compare p-values to former example
VCAinference(res, total.claim=sqrt(4.5), error.claim=sqrt(3.5), claim.type="SD")

# now using Satterthwaite methodology for CIs
VCAinference(res, total.claim=sqrt(4.5), error.claim=sqrt(3.5),
claim.type="SD", ci.method="satt")

# now add random error to example data forcing the ANOVA-estimate of the
# day-variance to be negative
set.seed(121)
tmpData <- dataEP05A2_1
tmpData$y <- tmpData$y + rnorm(80,,3)
res2 <- anovaVCA(y~day/run, tmpData)

# call 'VCAinference' with default settings
VCAinference(res2)

# extract components of the returned 'VCAinference' object
inf <- VCAinference(res2, total.claim=12)
inf$ConfInt$VC$OneSided # one-sided CIs for variance components
inf$ConfInt$VC$TwoSided # two-sided CI for variance components
inf$ChiSqTest

# request CIs for all VCs, default is to exclude CIs of negative VCs (excludeNeg=TRUE)
# solve MMEs first (or set MME=TRUE when calling anovaVCA)
res2 <- solveMME(res2)
VCAinference(res2, VarVC=TRUE)

# request CIs for all VCs, including those for negative VCs, note that all CI-limits
# are constrained to be >= 0
VCAinference(res2, VarVC=TRUE, excludeNeg=FALSE)

# request unconstrained CIs for all VCs, including those for negative VCS
# one has to re-fit the model allowing the VCs to be negative
res3 <- anovaVCA(y~day/run, tmpData, NegVC=TRUE, MME=TRUE)
VCAinference(res3, VarVC=TRUE, excludeNeg=FALSE, constrainCI=FALSE)

### use the numerical example from the CLSI EP05-A2 guideline (p.25)
data(Glucose,package="VCA")
res.ex <- anovaVCA(result~day/run, Glucose)

### also perform Chi-Squared tests
### Note: in guideline claimed SD-values are used, here, claimed variances are used
VCAinference(res.ex, total.claim=3.4^2, error.claim=2.5^2)


# load another example dataset and extract the "sample_1" subset
data(VCAdata1)
sample1 <- VCAdata1[which(VCAdata1$sample==1),]

# generate an additional factor variable and random errors according to its levels
sample1$device <- gl(3,28,252)
```

```
set.seed(505)
sample1$y <- sample1$y + rep(rep(rnorm(3,,.25), c(28,28,28)),3)

# fit a crossed-nested design with main factors 'lot' and 'device'
# and nested factors 'day' and 'run' nested below, also request A-matrices
res1 <- anovaVCA(y~(lot+device)/day/run, sample1)

# get confidence intervals, covariance-matrix of VCs, ...,
# explicitly request the covariance-matrix of variance components
# solve MMEs first
res1 <- solveMME(res1)
inf1 <- VCAinference(res1, VarVC=TRUE, constrainCI=FALSE)
inf1

# print numerical values with more digits
print(inf1, digit=12)

# print only parts of the 'VCAinference' object (see \code{\link{print.VCAinference}})
print(inf1, digit=12, what=c("VCA", "VC"))

# extract complete covariance matrix of variance components
# (main diagonal is part of standard output -> "Var(VC")
VarCovVC <- vcovVC(inf1$VCAobj)
round(VarCovVC, 12)

# use by-processing and specific argument-values for each level of the by-variable
data(VCAdata1)
fit.all <- anovaVCA(y~(device+lot)/day/run, VCAdata1, by="sample", NegVC=TRUE)
inf.all <- VCAinference(fit.all, total.claim=c(.1,.75,.8,1,.5,.5,2.5,20,.1,1))
print.VCAinference(inf.all, what="VC")

## End(Not run)
```

`vcov.VCA` *Calculate Variance-Covariance Matrix of Fixed Effects for an 'VCA' Object*

Description

Return the variance-covariance matrix of fixed effects for a linear mixed model applicable for objects of class 'VCA'.

Usage

```
## S3 method for class 'VCA'
vcov(object, quiet = FALSE, ...)
```

Arguments

object	(VCA) object for which the variance-covariance matrix of fixed effects shall be calculated
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise
...	additional parameters

Details

Actually this function only extracts this matrix or, if not available, calls function vcovFixed which performs calculations. It exists for compatibility reasons, i.e. for coneniently using objects of class 'VCA' with other packages expecting this function, e.g. the 'multcomp' package for general linear hypotheses for parametric models.

Value

(matrix) corresponding to the variance-covariance matrix of fixed effects

Examples

```
## Not run:
data(dataEP05A2_1)
fit1 <- anovaMM(y~day/(run), dataEP05A2_1)
vcov(fit1)

fit2 <- anovaVCA(y~day/run, dataEP05A2_1)
vcov(fit2)

## End(Not run)
```

---

vcovFixed *Calculate Variance-Covariance Matrix and Standard Errors of Fixed Effects for an 'VCA' Object*

Description

The variance-covariance matrix of fixed effects for the linear mixed model in 'obj' is calculated.

Usage

```
vcovFixed(obj, quiet = FALSE)
```

Arguments

obj	(VCA) object for which the variance-covariance matrix of fixed effects shall be calculated
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

The variance-covariance matrix of fixed effects for a linear mixed model corresponds to matrix $(X^T V^{-1} X)^-$, where >^{T}< denotes the transpose operator, >$^{-1}$< the regular matrix inverse, and >$^-$< the generalized (Moore-Penrose) inverse of a matrix.

Value

(matrix) corresponding to the variance-covariance matrix of fixed effects

Examples

```
## Not run:
data(dataEP05A2_1)
fit1 <- anovaMM(y~day/(run), dataEP05A2_1)
vcov(fit1)

fit2 <- anovaVCA(y~day/run, dataEP05A2_1)
vcov(fit2)

## End(Not run)
```

---

vcovVC — *Calculate Variance-Covariance Matrix of Variance Components of 'VCA' objects*

Description

This function computes the variance-covariance matrix of variance components (VC) either applying the approach given in the 1^{st} reference ('method="scm"') or using the approximation given in the 2^{nd} reference ('method="gb"').

Usage

```
vcovVC(obj, method = NULL, quiet = FALSE)
```

Arguments

obj	(VCA) object
method	(character) string, optionally specifying whether to use the algorithm given in the 1st reference ("scm") or in the 2nd refernce ("gb"). If not not supplied, the option is used coming with the 'VCA' object.
quiet	(logical) TRUE = will suppress any warning, which will be issued otherwise

Details

This function is called on a 'VCA' object, which can be the sole argument. In this case the value assigned to element 'VarVC.method' of the 'VCA' object will be used.

Value

(matrix) corresponding to variance-covariance matrix of variance components

Author(s)

Andre Schuetzenmeister <andre.schuetzenmeister@roche.com>, Florian Dufey <florian.dufey@roche.com>

References

Searle, S.R, Casella, G., McCulloch, C.E. (1992), Variance Components, Wiley New York

Giesbrecht, F.G. and Burns, J.C. (1985), Two-Stage Analysis Based on a Mixed Model: Large-Sample Asymptotic Theory and Small-Sample Simulation Results, Biometrics 41, p. 477-486

Examples

```
## Not run:
data(realData)
dat1 <- realData[realData$PID==1,]
fit  <- anovaVCA(y~lot/calibration/day/run, dat1)
vcovVC(fit)
vcovVC(fit, "scm") # Searle-Casella-McCulloch method (1st reference)
vcovVC(fit, "gb") # Giesbrecht and Burns method (2nd reference)

## End(Not run)
```

Index