Package 'sommer'

October 25, 2026

Type Package

Title Solving Mixed Model Equations in R

Version 4.5.12

Date 2026-09-30

Maintainer Giovanny Covarrubias-Pazaran <cova_ruber@live.com.mx>

Description Structural multivariate-univariate linear mixed model solver for estimation of multiple random effects with unknown variance-covariance structures (e.g., unstructured, diagonal, autoregressive, factor analytic, reduced rank, among others) and known covariance among levels of random effects (e.g., pedigree and genomic relationship matrices) (Covarrubias-Pazaran, 2016 <doi:10.1371/journal.pone.0156744>; Maier et al., 2015 <doi:10.1016/j.ajhg.2014.12.006>; Jensen et al., 19
timates can be obtained using the Direct-Inversion Newton-Raphson and Direct-Inversion Average Information algorithms for the problems r x r (r being the number of records) or using the Henderson-based average information algorithm for the problem c x c (c being the number of coefficients to estimate).

Depends R (>= 3.5.0), Matrix (>= 1.6-2), methods, stats, enhancer

LazyLoad yes

License GPL-3

Imports Rcpp (>= 0.12.19), MASS, crayon

BugReports https://github.com/covaruber/sommer/issues

URL https://github.com/covaruber/sommer

LinkingTo Rcpp, RcppArmadillo, RcppProgress, RcppEigen, Matrix

SystemRequirements METIS graph partitioning library (optional; enables nested-dissection sparse ordering for large relationship matrices, auto-detected at configure time, falls back to AMD ordering when absent), NVIDIA CUDA toolkit with cuBLAS and cuSOLVER (optional; enables GPU kernels on Linux through SOMMER_CUDA_HOME; CUDA libraries are loaded dynamically)

Suggests RcppArmadillo, flexiblas, RhpcBLASctl, parallelly, rmarkdown, knitr, plyr, parallel, orthopolynom, RSpectra, lattice, emmeans (>= 1.4), estimability, testthat (>= 3.0.0), litedown

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation yes

Author Giovanny Covarrubias-Pazaran [aut, cre] (ORCID:
<https://orcid.org/0000-0002-7194-3837>)

Repository CRAN

Date/Publication 2026-10-25 14:40:02 UTC

Contents

sommer-package 4
A.mat 12
analyze_mme_crossovers 14
anova.mmes 15
antem 15
APY 16
ar1m 19
ar2m 20
ar3m 21
atm 22
atr 23
binm 24
car 25
coef.mmes 26
corgm 27
corImputation 28
covm 30
covmatrix_mmes 33
covparams_mmes 35
covparams_mmes_se 36
csm 37
csr 39
D.mat 40
dfToMatrix 42
dsm 43
dsr 44
dsumm 45
E.mat 47
fam 48
familym 50
fitted.mmes 51
fixm 52
generate_mme_workload 53
GWAS 54
H.mat 60
ism 62

loadings_mmes 63
mam 64
maternm 65
meffects_mmes 66
metricm 68
mmer 70
mmes 79
mmes_control 91
mme_workload_grid 96
MNR 97
ownm 101
plot.mmes 102
pmonitor 103
postPEV 104
postVarU 106
predict.mmes 108
print.sommer_mmes_control 114
print.sommer_mme_benchmark 114
print.sommer_mme_crossover 115
print.sommer_mme_profile 115
profile_mme_workload 116
r2 116
randef 117
residuals.mmes 118
rotate_back_mmes 119
rrm 119
run_mme_benchmark 122
sar 123
scores_mmes 124
spl2Dc 125
spl2Dmats 128
stackTraits 131
strm 132
summary.mmer 133
summary.mmes 134
toeplitzm 135
tpsmmbwrapper 136
unsm 139
usm 140
usr 141
vcc 142
vpredict 143
vs 145
vsm 148
vsr 150
wald_mmes 152

Index **155**

sommer-package **So***lving* **M***ixed* **M***odel* **E***quations in* **R**

Description

Sommer is a structural univariate and multivariate linear mixed-model package for fitting models with multiple random effects and flexible covariance structures. Variance parameters are estimated by restricted maximum likelihood (REML). The package provides two complementary numerical formulations.

`mmes` uses Henderson's mixed model equations with an Average Information REML algorithm. The mixed-model coefficient matrix is represented as a sparse system and factorized with Eigen/Armadillo routines. Sparse LDLT factorizations are reused throughout the REML calculations. Selected inverse elements required for likelihood derivatives and prediction error variances can be obtained with Takahashi sparse-inverse recursions, so a complete dense coefficient-matrix inverse is not routinely formed.

`mmer` provides the marginal-covariance MNR formulation. Its REML calculations are organized around the covariance matrix of the observations and the corresponding REML projection matrix, with Newton-Raphson/Fisher-scoring and Average Information updates for variance parameters.

Both formulations support multiple random effects and structured covariance models. Through `vsm`, covariance structures can be composed from an arbitrary number of covariance factors, including identity, diagonal, unstructured, autoregressive, moving-average, Toeplitz, factor-analytic, reduced-rank, antedependence, correlation and spatial structures, as well as user-defined covariance functions.

The marginal formulation in `mmer` performs its principal REML calculations in observation space, whereas `mmes` performs them through the sparse mixed-model coefficient system. Their relative computational performance therefore depends on the number of observations, the number of mixed-model coefficients, sparsity, and the covariance structures in the fitted model.

The numerical algorithms are coded primarily in C++ using Armadillo and Eigen. Sommer returns REML variance-covariance estimates, BLUEs, BLUPs, residuals, fitted values, information matrices and, when requested, prediction error variance information.

Functions for genetic analysis

The package provides kernels to estimate additive (`A.mat`), dominance (`D.mat`), epistatic (`E.mat`), single-step (`H.mat`) relationship matrices for diploid and polyploid organisms. It also provides flexibility to fit other genetic models such as full and half diallel models and random regression models.

A good converter from letter code to numeric format is implemented in the function `atcg1234`, which supports higher ploidy levels than diploid. Additional functions for genetic analysis have been included such as build a genotypic hybrid marker matrix (`build.HMM`), plot of genetic maps (`map.plot`), creation of manhattan plots (`manhattan`). If you need to use pedigree you need to convert your pedigree into a relationship matrix (use the 'getA' function from the pedigreemm package).

Functions for statistical analysis and S3 methods

The vpredict function can be used to estimate standard errors for linear combinations of variance components (e.g. ratios like h2). The r2 function calculates reliability. S3 methods are available for some parameter extraction such as:

+ predict.mmes

+ fitted.mmes

+ residuals.mmes

+ summary.mmes

+ coef.mmes

+ anova.mmes

+ plot.mmes

Functions for trial analysis

Recently, spatial modeling has been added added to sommer using the two-dimensional spline (spl2Dc).

Keeping sommer updated

The sommer package is updated on CRAN every 4-months due to CRAN policies but you can find the latest source at https://github.com/covaruber/sommer. This can be easily installed typing the following in the R console:

```
library(devtools)
install_github("covaruber/sommer")
```

This is recommended if you reported a bug, was fixed and was immediately pushed to GitHub but not in CRAN until the next update.

Tutorials

For tutorials on how to perform different analysis with sommer please look at the vignettes by typing in the terminal:

```
vignette("sommer.qg")
vignette("sommer.gxe")
vignette("sommer.vs.lme4")
vignette("sommer.spatial")
```

Getting started

The package has been equiped with several datasets to learn how to use the sommer package (and almost to learn all sort of quantitative genetic analysis):

* DT_halfdiallel, DT_fulldiallel and DT_mohring datasets have examples to fit half and full diallel designs.

* DT_h2 to calculate heritability

* DT_cornhybrids and DT_technow datasets to perform genomic prediction in hybrid single crosses

* DT_wheat dataset to do genomic prediction in single crosses in species displaying only additive effects.

* DT_cpdata dataset to fit genomic prediction models within a biparental population coming from 2 highly heterozygous parents including additive, dominance and epistatic effects.

* DT_polyploid to fit genomic prediction and GWAS analysis in polyploids.

* DT_gryphon data contains an example of an animal model including pedigree information.

* DT_btdata dataset contains an animal (birds) model.

* DT_legendre simulated dataset for random regression model.

* DT_sleepstudy dataset to know how to translate lme4 models to sommer models.

Differences of sommer >= 4.4.1 with previous versions

Since version 4.4.1, I have unified the use of the two different solving algorithms into the mmes function by just using the new argument henderson which by default is set to FALSE. Other than that the rest is the same with the addition that now the identity terms needs to be encapsulated in the ism function. In addition, now the multi-trait models need to be fitted in the long format. This are few but major changes to the way sommer models are fitted.

Differences of sommer >= 4.1.7 with previous versions

Since version 4.1.7 I have introduced the mmes-based average information function 'mmec' which is much faster when dealing with the r > c problem (more records than coefficients to estimate). This introduces its own covariance structure functons such as vsc(), usc(), dsc(), atc(), csc(). Please give it a try, although is in early phase of development.

Differences of sommer >= 3.7.0 with previous versions

Since version 3.7 I have completly redefined the specification of the variance-covariance structures to provide more flexibility to the user. This has particularly helped the residual covariance structures and the easier combination of custom random effects and overlay models. I think that although this will bring some uncomfortable situations at the beggining, in the long term this will help users to fit better models. In esence, I have abandoned the asreml formulation (not the structures available) given it's limitations to combine some of the sommer structures but all covariance structures can now be fitted using the 'vsm' functions.

Differences of sommer >= 3.0.0 with previous versions

Since version 3.0 I have decided to focus in developing the multivariate solver and for doing this I have decided to remove the M argument (for GWAS analysis) from the mmes function and move it to it's own function GWAS.

Before the mmes solver had implemented the usm(trait), diag(trait), at(trait) asreml formulation for multivariate models that allow to specify the structure of the trait in multivariate models. Therefore the MVM argument was no longer needed. After version 3.7 now the multi-trait structures can be specified in the Gt and Gtc arguments of the vsm function.

The Average Information algorithm had been removed in the past from the package because of its instability to deal with very complex models without good initial values. Now after 3.7 I have

brought it back after I noticed that starting with NR the first three iterations gives enough flexibility to the AI algorithm.

Keep in mind that sommer uses direct inversion (DI) algorithm which can be very slow for datasets with many observations (big 'n'). The package is focused in problems of the type p > n (more random effect(s) levels than observations) and models with dense covariance structures. For example, for experiment with dense covariance structures with low-replication (i.e. 2000 records from 1000 individuals replicated twice with a covariance structure of 1000x1000) sommer will be faster than MME-based software. Also for genomic problems with large number of random effect levels, i.e. 300 individuals (n) with 100,000 genetic markers (p). On the other hand, for highly replicated trials with small covariance structures or n > p (i.e. 2000 records from 200 individuals replicated 10 times with covariance structure of 200x200) asreml or other MME-based algorithms will be much faster and I recommend you to use that software.

Models Enabled

General linear mixed model

Both numerical engines fit the Gaussian linear mixed-model family

$$y = X\beta + Zu + e,$$

with

$$u \sim N(0, G), \qquad e \sim N(0, R),$$

and therefore

$$V = \mathrm{Var}(y) = ZGZ' + R.$$

For several independent random terms this becomes

$$V = \sum_{k=1}^{q} Z_k G_k Z_k' + R.$$

Here y is the response vector, X and Z are design matrices, β contains fixed effects, u contains random effects, and e contains residual effects. The same representation covers univariate and multivariate models after the corresponding responses, design matrices and covariance structures are assembled.

Covariance models specified through `vsm` use a product-level variance scale and an arbitrary number of covariance-shaping factors:

$$\Sigma = \sigma^2(K_1 \otimes K_2 \otimes \cdots \otimes K_m).$$

For a random effect with known relationship covariance A,

$$G = \sigma^2(K_1 \otimes K_2 \otimes \cdots \otimes K_m) \otimes A.$$

The CovarianceFactor interface separates evaluation of each K_j, its derivatives, parameter reporting and trust limits from the REML solver. New covariance structures can therefore be added without changing the core `mmes` optimization algorithm.

Marginal MNR formulation in mmer

The MNR engine used by `mmer` organizes the REML calculations around the marginal covariance V. Define the REML projection matrix

$$P = V^{-1} - V^{-1}X(X'V^{-1}X)^{-1}X'V^{-1}.$$

Apart from constants independent of the variance parameters, the Gaussian REML log-likelihood is

$$\ell_R(\theta) = -\frac{1}{2}\{\log|V| + \log|X'V^{-1}X| + y'Py\}.$$

For variance parameter θ_i, let

$$V_i = \frac{\partial V}{\partial \theta_i}.$$

For covariance models linear in the fitted variance parameters, the REML score has the standard form

$$s_i = \frac{1}{2}\{y'PV_iPy - \mathrm{tr}(PV_i)\}.$$

The MNR C++ implementation forms the projected covariance derivatives PV_i. Its Average Information matrix is

$$AI_{ij} = \frac{1}{2}y'PV_iPV_jPy,$$

while its Newton-Raphson/Fisher-scoring path uses trace products of the projected covariance derivatives. The resulting information system determines the variance-parameter search direction, and the implementation can blend the information calculation with an EM-type diagonal stabilization during early iterations.

This formulation is referred to as *marginal* because its principal covariance calculations are carried out in observation space through V and P. This describes the numerical formulation more accurately than naming it after a particular matrix operation.

Sparse Henderson Average Information formulation in mmes

The `mmes` engine uses Henderson's mixed model equations. With $W = [X \;\; Z]$, the coefficient system is

$$C = \begin{bmatrix} X'R^{-1}X & X'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z + G^{-1} \end{bmatrix},$$

and the right-hand side is

$$r = \begin{bmatrix} X'R^{-1}y \\ Z'R^{-1}y \end{bmatrix}.$$

The BLUE and BLUP solutions satisfy

$$Cb = r, \qquad b = (\widehat{\beta}', \widehat{u}')'.$$

The current C++ implementation stores C sparsely and uses an Eigen `SimplicialLDLT` factorization. Symbolic factorization information is reused when the sparsity pattern remains unchanged. Random-effect precision contributions are inserted blockwise. Structured residual covariance models use optimized diagonal or repeated-block/Kronecker paths when applicable and a generic sparse residual-factorization path otherwise.

The REML criterion is evaluated from determinant contributions for the random covariance structures, residual covariance and mixed-model coefficient matrix, together with the quadratic term $y'Py$. This is a mixed-model-equation representation of the same REML objective optimized by the marginal formulation.

For covariance parameter η_i, the generic covariance engine supplies

$$\Sigma_i = \frac{\partial \Sigma}{\partial \eta_i}.$$

For

$$\Sigma = \sigma^2(K_1 \otimes \cdots \otimes K_m),$$

a parameter belonging to factor K_j has derivative

$$\frac{\partial \Sigma}{\partial \eta_{jk}} = \sigma^2 K_1 \otimes \cdots \otimes \frac{\partial K_j}{\partial \eta_{jk}} \otimes \cdots \otimes K_m.$$

For the working scale $\tau = \log(\sigma^2)$,

$$\frac{\partial \Sigma}{\partial \tau} = \Sigma.$$

The score trace terms use selected elements associated with the sparse coefficient system. These are obtained from the LDLT factors with Takahashi sparse-inverse recursions; a complete C^{-1} is not required during REML iterations.

Average Information is calculated by analytically differentiating the factorized mixed-model equations. Differentiating

$$Cb = r$$

with respect to η_i gives

$$Cb_i = r_i - C_i b,$$

where

$$C_i = \frac{\partial C}{\partial \eta_i}, \qquad r_i = \frac{\partial r}{\partial \eta_i}.$$

The sensitivity systems therefore reuse the same sparse factorization of C. For covariance Σ with precision $\Lambda = \Sigma^{-1}$ and derivative $B_i = \partial\Sigma/\partial\eta_i$,

$$\Lambda_i = -\Lambda B_i \Lambda.$$

This factor-differentiation formulation avoids constructing an observation-space working variate for every random covariance parameter. Residual/residual and random/residual Average Information terms use the corresponding mixed-model/Schur-complement identities.

Covariance constructors may provide analytic derivatives in C++ or R. If a constructor uses numerical differentiation, only its small covariance factor $K_j(\eta)$ is differentiated by central finite differences; the REML likelihood and downstream score and Average Information calculations are not numerically differentiated.

Sparse inverse subsets and prediction error variances

During optimization, `mmes` does not routinely materialize the complete C^{-1}. Takahashi recursions provide the selected inverse subset required for REML trace calculations. After convergence, `control$output$inverse=1` obtains the diagonal information needed for prediction error variances through the sparse inverse subset, whereas `control$output$inverse=2` requests the complete coefficient-matrix inverse.

Parameter updates and numerical safeguards

Covariance parameters are optimized in working coordinates chosen to respect their parameter spaces, such as logarithms for positive scales and hyperbolic-tangent or bounded-logit mappings for correlations. Each CovarianceFactor descriptor supplies its natural-scale reporting transformation and parameter-specific trust limit.

The information system determines a proposed REML update. In the current `mmes` implementation, the proposal is limited in working-coordinate space and then checked against the exact REML likelihood. If the proposed global step decreases the accepted likelihood beyond numerical tolerance, the step is geometrically reduced and reevaluated. A likelihood decrease is therefore not interpreted as convergence. Boundary handling and robust information solves are retained for difficult or weakly identified variance components.

Choosing between the two formulations

Both engines estimate the same class of linear mixed models but organize the numerical work differently. `mmer` uses the marginal covariance representation and is attractive when observation-space calculations are moderate in size. `mmes` uses the sparse Henderson coefficient system and is particularly attractive when the mixed-model equations are sparse and their factorization can be reused efficiently. The best choice depends on the dimensions, sparsity and covariance structures of the fitted model rather than on a universal rule based only on the number of records or coefficients.

See `mmes`, `mmer`, and `vsm` for the interfaces and available covariance structures.

Bug report and contact

If you have any questions or suggestions please post it in https://stackoverflow.com or https://stats.stackexchange.com

I'll be glad to help or answer any question. I have spent a valuable amount of time developing this package. Please cite this package in your publication. Type 'citation("sommer")' to know how to cite it.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G. 2016. Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Covarrubias-Pazaran G. 2018. Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639

Sanderson, C., & Curtin, R. (2025). Armadillo: An Efficient Framework for Numerical Linear Algebra. arXiv preprint arXiv:2502.03000.

Bernardo Rex. 2010. Breeding for quantitative traits in plants. Second edition. Stemma Press. 390 pp.

Gilmour et al. 1995. Average Information REML: An efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51(4):1440-1450.

Henderson C.R. 1975. Best Linear Unbiased Estimation and Prediction under a Selection Model. Biometrics vol. 31(2):423-447.

Kang et al. 2008. Efficient control of population structure in model organism association mapping. Genetics 178:1709-1723.

Lee et al. 2015. MTG2: An efficient algorithm for multivariate linear mixed model analysis based on genomic information. Cold Spring Harbor. doi: https://doi.org/10.1101/027201.

Maier et al. 2015. Joint analysis of psychiatric disorders increases accuracy of risk prediction for schizophrenia, bipolar disorder, and major depressive disorder. Am J Hum Genet; 96(2):283-294.

Searle. 1993. Applying the EM algorithm to calculating ML and REML estimates of variance components. Paper invited for the 1993 American Statistical Association Meeting, San Francisco.

Yu et al. 2006. A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. Genetics 38:203-208.

Tunnicliffe W. 1989. On the use of marginal likelihood in time series model estimation. JRSS 51(1):15-27.

Examples

```
####============================================####
#### For CRAN time limitations most lines in the
#### examples are silenced with one '#' mark,
#### remove them and run the examples
####============================================####
```

```
####==========================================####
#### EXAMPLES
#### Different models with sommer
####==========================================####

data(DT_example, package="enhancer")




DT <- DT_example
DT=DT[with(DT, order(Env)), ]
head(DT)

####==========================================####
#### Univariate homogeneous variance models  ####
####==========================================####

## Compound simmetry (CS) model
ans1 <- mmes(Yield~Env,
             random= ~ Name + Env:Name,
             rcov= ~ units,
             data=DT)
summary(ans1)

####==========================================####
#### Univariate heterogeneous variance models  ####
####==========================================####
## Compound simmetry (CS) + Diagonal (DIAG) model
ans3 <- mmes(Yield~Env,
             random= ~Name + vsm(dsm(Env),ism(Name)),
             rcov= ~ vsm(dsm(Env),ism(units)),
             data=DT)
summary(ans3)
```

A.mat *Additive relationship matrix*

Description

Calculates the realized additive relationship matrix. Currently is the C++ implementation of van Raden (2008).

Usage

```
A.mat(X,min.MAF=0,return.imputed=FALSE)
```

Arguments

X	Matrix ($n \times m$) of unphased genotypes for n lines and m biallelic markers, coded as {-1,0,1}. Fractional (imputed) and missing values (NA) are allowed.
min.MAF	Minimum minor allele frequency, a finite value between zero and 0.5. Markers at or below this threshold and all constant markers are removed.
return.imputed	When TRUE, the imputed marker matrix is returned.

Details

Missing values are replaced by the column (marker) mean. At least two individuals and one variable marker surviving the frequency filter are required. Genotypes must lie in $[-1, 1]$; a marker with no observed genotypes cannot be imputed.

For the VanRaden method, let p_k be the column mean of $(X + 1)/2$ for each retained marker and $W_{ik} = X_{ik} + 1 - 2p_k$. Then $A = WW'/c$, where $c = 2\sum_k p_k(1 - p_k)$.

Mean imputation is explicit. Earlier implementations inadvertently used the imputation helper's default median despite reporting mean imputation; results with missing genotypes can therefore differ.

Value

If return.imputed = FALSE, the $n \times n$ additive relationship matrix is returned.

If return.imputed = TRUE, the function returns a list containing

$A the A matrix

$X the imputed marker matrix

References

Endelman, J.B., and J.-L. Jannink. 2012. Shrinkage estimation of the realized relationship matrix. G3:Genes, Genomes, Genetics. 2:1405-1413. doi: 10.1534/g3.112.004259

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
####============================================####
#### random population of 200 lines with 1000 markers
####============================================####
X <- matrix(rep(0,200*1000),200,1000)
for (i in 1:200) {
  X[i,] <- ifelse(runif(1000)<0.5,-1,1)
}

A <- A.mat(X)
```

```
####============================================####
#### take a look at the Genomic relationship matrix
#### (just a small part)
####============================================####
# colfunc <- colorRampPalette(c("steelblue4","springgreen","yellow"))
# hv <- heatmap(A[1:15,1:15], col = colfunc(100),Colv = "Rowv")
# str(hv)
```

`analyze_mme_crossovers` *Analyze Crossover Frontiers from MME Benchmarks*

Description

Identifies problem configurations (factors, levels, dimensions) where experimental solver paths provide speedups over baseline methods, fitting empirical crossover conditions.

Usage

```
analyze_mme_crossovers(
  benchmark,
  baseline_engine = NULL,
  candidate_engine = NULL
)
```

Arguments

`benchmark`	An object of class `sommer_mme_benchmark`.
`baseline_engine`	Name of the reference baseline engine. Default is the first engine tested.
`candidate_engine`	Name of the experimental engine to evaluate.

Value

An object of class `"sommer_mme_crossover"`.

anova.mmes *anova form a GLMM fitted with mmes*

Description

anova method for class "mmes".

Usage

```
## S3 method for class 'mmes'
anova(object, object2=NULL, ...)
```

Arguments

object	an object of class "mmes"
object2	an object of class "mmes". If supplied, a likelihood ratio test between the two models is returned (not available for PQL fits). If NULL, Wald tests for the fixed effects of object are returned by wald_mmes.
...	Further arguments passed to wald_mmes (e.g. ssType, denDF) when object2 is NULL.

Value

vector of anova

Author(s)

Giovanny Covarrubias

See Also

anova, mmes, wald_mmes

antem *Antedependence covariance structure*

Description

Antedependence covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
antem(x, order = 1L, beta = NULL, innovations = NULL, fixed = NULL)
```

Arguments

x	Ordered variable defining q covariance levels.
order	Antedependence order, 1 <= order < q.
beta	Optional starting regression coefficients for allowed subdiagonals.
innovations	Optional q positive innovation variances.
fixed	Logical vector controlling all beta coefficients and q-1 innovation-variance ratios.

Details

The modified-Cholesky representation is

$$Ty = e, \qquad \mathrm{Cov}(e) = D, \qquad K = T^{-1} D T^{-\mathsf{T}}.$$

The matrix T is unit lower triangular, with nonzero regression coefficients only within the requested number of preceding levels. The first innovation variance is the scale reference; the remaining innovation variances are positive ratios.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

ar1m, toeplitzm, vsm.

Examples

```
## Not run:
vsm(antem(time, order=2), ism(id))

## End(Not run)
```

APY *Construct an APY Relationship Precision Matrix*

Description

Constructs the Algorithm for Proven and Young (APY) approximation for a positive-definite relationship matrix and a user-specified core set. In the genomic evaluation literature, the input is the genomic relationship matrix G for genotyped animals, usually calculated from SNP marker data. The core and non-core partition is within that genotyped set; it is not a pedigree-versus-genomic partition. The non-core conditional covariance is approximated by its diagonal. The returned matrix is a sparse precision matrix suitable for the Gu argument of mmes() in a genomic BLUP model.

Usage

```
APY(G, core, tol = 1e-10, return.details = FALSE)
```

Arguments

G	A finite, symmetric, positive-definite numeric relationship matrix. For the standard APY genetic evaluation use, this is the genomic relationship matrix among genotyped animals, commonly obtained from marker genotypes with A.mat(). Row and column names, when present, must agree exactly. This implementation receives G as a dense matrix.
core	Unique one-based row indices or row names selecting the core individuals.
tol	Positive tolerance below one, used for symmetry validation and the minimum acceptable conditional variance relative to the relationship scale. Small or non-positive conditional variances cause an error; they are not silently regularized.
return.details	If TRUE, return a list containing the sparse precision matrix as Gu, core and non-core indices, regression matrix B, and non-core conditional variances.

Details

Partition G into core and non-core blocks. With $B = G_{nc}G_{cc}^{-1}$ and $D = \mathrm{diag}(G_{nn} - BG_{cn})$, APY uses the precision $\begin{bmatrix} G_{cc}^{-1}+B'D^{-1}B & -B'D^{-1} \\ -D^{-1}B & D^{-1} \end{bmatrix}$. It preserves the core block and core-to-non-core relationships while replacing non-core conditional correlations with zero. Thus APY is exact when the non-core conditional covariance is diagonal; otherwise it is an approximation to the supplied relationship model.

APY was developed to provide a computationally economical approximation to the inverse genomic relationship matrix for large genomic evaluations (Misztal, 2016; Fragomeni et al., 2015). The algorithm partitions genotyped animals into core and non-core individuals. The core contains the relationships that are modeled jointly; each non-core animal is regressed on the core, with its remaining conditional variance treated as independent of other non-core animals. Thus non-core animals are not independent of the core: their core-to-non-core relationships are retained.

Pedigree relationships are a distinct input. For pedigree-only models, the pedigree relationship matrix A and its sparse inverse A^{-1} are normally used directly; APY is not required. In single-step genomic BLUP, pedigree and genomic information are combined, conventionally through $H^{-1} = A^{-1} + \begin{bmatrix} 0 & 0 \\ 0 & G^{-1}-A_{22}^{-1} \end{bmatrix}$, where genotyped animals are ordered last and A_{22} is their pedigree relationship block. APY approximates the genomic term G^{-1} in that correction; it does not replace the pedigree term or construct H^{-1}. This function returns only the APY genomic precision. Do not use it as the complete single-step precision for a population that also contains non-genotyped animals.

This function constructs APY from an already available relationship matrix. It does not avoid the cost of constructing a full dense relationship matrix from markers, and it does not change the solver algorithm. Marker-based block construction and single-step matrix assembly are not implemented here.

Value

A sparse symmetric APY precision matrix with individual dimnames and `attr(x, "inverse") = TRUE`, ready for `Gu`. Metadata describing the core and conditional variances is stored in `attr(x, "APY")`. If `return.details = TRUE`, returns the diagnostic list described above.

References

Misztal I (2016). Inexpensive Computation of the Inverse of the Genomic Relationship Matrix in Populations with Small Effective Population Size. *Genetics*, 202(2), 401–409. doi:10.1534/genetics.115.182089.

Fragomeni BO, Lourenco DA, Tsuruta S, Masuda Y, Aguilar I, Legarra A, Lawlor TJ, Misztal I (2015). Use of genomic recursions and algorithm for proven and young animals for single-step genomic BLUP analyses–a simulation study. *Journal of Animal Breeding and Genetics*, 132(5), 340–345. doi:10.1111/jbg.12161.

See Also

`mmes`, `A.mat`, `H.mat`

Examples

```
## Marker-derived GBLUP for genotyped animals.
set.seed(2)
ids <- paste0("animal", seq_len(12))
markers <- matrix(sample(c(-1, 0, 1), 12 * 80, replace = TRUE), nrow = 12,
                  dimnames = list(ids, NULL))
G <- A.mat(markers)

## Blend slightly with the identity so this small illustrative G is positive
## definite, then construct APY precision for a six-animal core.
G <- 0.95 * G + 0.05 * diag(nrow(G))
dimnames(G) <- list(ids, ids)
Ginv.apy <- APY(G, core = ids[seq_len(6)])

phenotypes <- data.frame(
  id = factor(rep(ids, each = 3), levels = ids),
  y = rnorm(length(ids) * 3)
)
fit <- mmes(y ~ 1,
  random = ~vsm(ism(id), Gu = Ginv.apy),
  rcov = ~units,
  data = phenotypes,
  control = mmes_control(diagnostics = list(verbose = FALSE)))

## In a single-step evaluation, this APY approximation supplies G^{-1} for
## the genotyped block correction; it is not itself the complete H^{-1}.
```

ar1m *First-order autoregressive covariance structure*

Description

First-order autoregressive covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
ar1m(x, rho = 0.30, fixed = FALSE,
     variance = c("homogeneous", "heterogeneous"), values = NULL)
```

Arguments

x	Ordered factor or design defining q ordered covariance levels.
rho	Starting AR(1) correlation in (-1,1).
fixed	For homogeneous variance, a logical indicating whether rho is fixed. For heterogeneous variance, a logical vector of length q: rho followed by q-1 variance-ratio parameters.
variance	Whether the AR correlation has homogeneous or heterogeneous marginal variances.
values	Optional q positive starting variances when variance = "heterogeneous".

Details

The homogeneous AR(1) shape is

$$K_{ij} = \rho^{|i-j|}.$$

Heterogeneous mode uses

$$K = DR(\rho)D,$$

with the first variance fixed as the relative-scale reference. The working correlation coordinate is $\eta = \operatorname{atanh}(\rho)$, ensuring $|\rho| < 1$.

Factor levels define the lag order, so numeric-looking levels must be in increasing numeric order; ar1m() warns otherwise. Inside vsm() one correlation is shared by every level of the other Kronecker factors. For a separate correlation per trial, wrap the vsm() term in dsumm.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

ar2m, ar3m, toeplitzm, vsm, dsumm.

Examples

```
## Not run:
vsm(ar1m(time), ism(id))

# Separable row-by-range residual shared across trials:
# rcov = ~ vsm(dsm(trial), ar1m(range), ar1m(row), ism(units))
# Trial-specific correlations and variances:
# rcov = ~ dsumm(vsm(ar1m(range), ar1m(row), ism(units)), by=trial)

## End(Not run)
```

`ar2m` *Second-order autoregressive covariance structure*

Description

Second-order autoregressive covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
ar2m(x, pacf = c(0.20, 0.10), fixed = NULL,
     variance = c("homogeneous", "heterogeneous"), values = NULL)
```

Arguments

`x`	Ordered factor defining covariance levels.
`pacf`	Length-two vector of starting partial autocorrelations, each strictly between -1 and 1.
`fixed`	For homogeneous variance, a logical vector of length two. For heterogeneous variance, a logical vector of length q+1: two PACFs followed by q-1 variance-ratio parameters.
`variance`	Whether the AR correlation has homogeneous or heterogeneous marginal variances.
`values`	Optional q positive starting variances when `variance = "heterogeneous"`.

Details

A stationary AR(2) covariance is parameterized through reflection coefficients / partial autocorrelations. Each PACF is mapped from an unconstrained working coordinate by `tanh`, then converted to stable AR coefficients by the Levinson/Durbin recursion. Heterogeneous mode uses

$$K = DRD$$

and estimates q-1 positive variance ratios in addition to the PACFs.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`ar1m`, `ar3m`, `toeplitzm`, `vsm`.

Examples

```
## Not run:
vsm(ar2m(time), ism(id))

## End(Not run)
```

`ar3m` *Third-order autoregressive covariance structure*

Description

Third-order autoregressive covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
ar3m(x, pacf = c(0.20, 0.10, 0.05), fixed = NULL,
     variance = c("homogeneous", "heterogeneous"), values = NULL)
```

Arguments

`x`	Ordered factor defining covariance levels.
`pacf`	Length-three vector of starting partial autocorrelations, each strictly between -1 and 1.
`fixed`	For homogeneous variance, a logical vector of length three. For heterogeneous variance, a logical vector of length q+2: three PACFs followed by q-1 variance-ratio parameters.
`variance`	Whether the AR correlation has homogeneous or heterogeneous marginal variances.
`values`	Optional q positive starting variances when `variance = "heterogeneous"`.

Details

A stationary AR(3) covariance is parameterized through three partial autocorrelations. The PACF parameterization guarantees stationarity while allowing unconstrained working coordinates. Heterogeneous mode uses

$$K = DRD$$

and estimates q-1 positive variance ratios in addition to the PACFs.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`ar1m`, `ar2m`, `toeplitzm`, `vsm`.

Examples

```
## Not run:
vsm(ar3m(time), ism(id))

## End(Not run)
```

`atm` *Selected-level diagonal covariance structure*

Description

Selected-level diagonal covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
atm(x, levs, values = NULL, fixed = NULL)
```

Arguments

`x`	Variable defining all available levels.
`levs`	Character names or numeric positions of the levels retained in this covariance factor.
`values`	Optional positive starting variances for the selected levels.
`fixed`	Logical vector of length length(levs)-1 controlling the selected variance ratios.

Details

`atm` is a selected-level version of `dsm`. The design is restricted to `levs`; observations outside those levels receive zeros for this factor. The first selected level is the scale reference and the remaining selected levels are positive variance ratios. This representation is most naturally used for random effects; residual factors must still assign exactly one covariance-product coordinate to every observation.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

dsm, vsm, mmes.

Examples

```
## Not run:
vsm(atm(environment, c("E1","E3")), ism(genotype))

## End(Not run)
```

---

atr *Selected-level diagonal covariance structure for* mmer *and* vsr

---

Description

Creates a diagonal covariance-component selector for specified columns or levels of a covariance dimension. It is intended for the mmer covariance-model interface, typically inside vsr.

Usage

```
atr(x, levs)
```

Arguments

x	A factor, character vector, numeric vector, or design/incidence matrix defining the covariance dimension. Factor and character inputs are expanded to incidence columns. A matrix is used directly.
levs	Character vector identifying the columns or levels whose diagonal entries are selected. If omitted, all available column names are selected.

Details

atr() constructs a design matrix Z together with a diagonal thetaC selector. Let q be the number of columns of Z, and define

$$a_j = \begin{cases} 1, & \text{if level } j \text{ is included in levs,} \\ 0, & \text{otherwise.} \end{cases}$$

The returned constraint matrix is

$$\text{thetaC} = \text{diag}(a_1, \ldots, a_q).$$

Thus atr() is useful when a covariance component is to be associated only with a specified subset of levels or design columns. Levels not selected in levs receive zero on the corresponding diagonal of thetaC.

This function belongs to the mmer/vsr covariance-structure interface. The related atm function belongs to the newer mmes/vsm CovarianceFactor interface and uses a different parameterization.

For a matrix input, its column names define the available levels. For factor or character input, the incidence-matrix column names define the levels. If levs is missing, all available columns are selected.

Value

A list with components:

- Z: the design/incidence representation of x.
- thetaC: a diagonal zero/one matrix selecting the levels specified by levs.

See Also

vsr, dsr, usr, csr, atm, mmer

Examples

```
# Select levels B and C
A <- atr(factor(c("A","B","A","C")), levs = c("B","C"))
A$Z
A$thetaC
```

---

binm *Binomial proportion response with its number of trials*

Description

Builds a single proportion response for binomial mmes fits from counts of successes and failures (or trials). The number of trials travels with the proportion as an attribute and is used as the binomial prior weight, so the single-response interface of mmes is preserved.

Usage

```
binm(successes, failures = NULL, trials = NULL)
```

Arguments

successes	Numeric vector of successes.
failures	Numeric vector of failures. Provide either failures or trials.
trials	Numeric vector with the number of trials.

Details

The result is successes/trials with class "binm" and attribute "trials". Subsetting keeps the trials aligned, so the vector can be stored as a data-frame column. Missing counts give a missing response, handled by the naMethodY rules of mmes. Fitting a proportion without binm treats every value as one trial, which is wrong unless all trials equal one.

Value

A numeric vector of proportions of class "binm".

See Also

mmes

Examples

```
set.seed(1)
d <- data.frame(x = rnorm(60), g = factor(rep(1:12, each = 5)))
d$n <- sample(5:10, 60, TRUE)
d$s <- rbinom(60, d$n, plogis(0.3 * d$x))
fit <- mmes(binm(s, trials = n) ~ x,
  random = ~ g,
  data = d,
  family = binomial(),
  control = mmes_control(update = list(blendWeight = 0),
  diagnostics = list(verbose = FALSE)))
fit$b
```

---

car *Proper conditional autoregressive spatial covariance structure*

Description

Proper conditional autoregressive spatial covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
car(x, W, rho = 0.10, fixed = FALSE)
```

Arguments

x	Factor or design defining q spatial levels.
W	Symmetric nonnegative adjacency/weights matrix with zero diagonal, aligned to the levels of x.
rho	Starting proper-CAR dependence parameter.
fixed	Logical indicating whether rho is fixed.

Details

Let $D = \mathrm{diag}(W\mathbf{1})$. The proper CAR precision and covariance are

$$Q = D - \rho W, \qquad M = Q^{-1}, \qquad K = M/M_{11}.$$

The constructor requires positive row sums, so isolated levels are not allowed. The admissible open interval for rho is obtained from the eigenvalues of $D^{-1/2}WD^{-1/2}$ and is enforced through a bounded-logit working coordinate. This is a proper, nonsingular CAR model; an intrinsic singular CAR is not used by the current precision-based Henderson implementation. The covariance derivative is analytic.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`sar`, `maternm`, `vsm`.

Examples

```
## Not run:
vsm(car(location, W), ism(genotype))

## End(Not run)
```

`coef.mmes` *coef form a GLMM fitted with mmes*

Description

`coef` method for class `"mmes"`.

Usage

```
## S3 method for class 'mmes'
coef(object, ...)
```

Arguments

`object`	an object of class `"mmes"`
`...`	Further arguments to be passed

Value

vector of coef

Author(s)

Giovanny Covarrubias

See Also

`coef`, `mmes`

corgm *General positive-definite correlation structure*

Description

General positive-definite correlation structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
corgm(x, theta = NULL, fixed = NULL,
      variance = c("homogeneous", "heterogeneous"), values = NULL)
```

Arguments

x	Variable defining q covariance levels.
theta	Optional positive-definite q by q starting covariance or correlation matrix. With variance="heterogeneous" its diagonal is used as starting variances when values is not given.
fixed	Logical vector flagging parameters held at their starting value: length q(q-1)/2 (correlations) for the homogeneous mode and q(q-1)/2 + q-1 (correlations, then variance ratios of levels 2..q) for the heterogeneous mode.
variance	"homogeneous" (default) gives a pure correlation matrix scaled by the single variance of vsm. "heterogeneous" adds one variance per level.
values	Optional positive starting variances (length q) for the heterogeneous mode. Only their ratios to the first level matter; the first level's variance is the vsm scale. When omitted, the starting ratios are taken from the data.

Details

An unrestricted SPD correlation matrix is represented using a unit-diagonal lower factor A. The intermediate matrix $S = AA^{\mathsf{T}}$ is standardized to correlation scale,

$$K_{ij} = S_{ij}/\sqrt{S_{ii}S_{jj}}.$$

The unrestricted working parameters are the strict lower-triangular entries of A.

With variance="heterogeneous" the shape is $K = DRD$ with $D = \mathrm{diag}(1, \sqrt{r_2}, \ldots, \sqrt{r_q})$, where $r_l = \sigma_l^2/\sigma_1^2 = \exp(\phi_l)$ are variance ratios relative to the first level; the vsm scale is σ_1^2. The model is the same as usm but parameterized as correlations and variances, which allows individual variances to be fixed. This is what mixed-family multi-trait models (familym) need: the residual variance of a binomial or Poisson trait is fixed to one while the correlations and the variances of Gaussian traits are estimated. Reported parameters (covparams_mmes) are the correlations and the variances of each level.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

csm, usm, vsm, covmatrix_mmes, familym.

Examples

```
data(DT_example, package = "enhancer")
DT <- DT_example
# genetic correlations between environments with a homogeneous variance
m1 <- mmes(Yield ~ Env,
  random = ~ vsm(corgm(Env), ism(Name)),
  rcov = ~ units,
  data = DT,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
covparams_mmes(m1, 1)

# heterogeneous variances (equivalent to usm, parameterized as corr/var)
m2 <- mmes(Yield ~ Env,
  random = ~ vsm(corgm(Env, variance = "heterogeneous"), ism(Name)),
  rcov = ~ units,
  data = DT,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
covparams_mmes(m2, 1)
covmatrix_mmes(m2, 1)$correlation
```

---

corImputation *Imputing a matrix using correlations*

Description

corImputation imputes missing data based on the correlation that exists between row levels.

Usage

```
corImputation(wide, Gu=NULL, nearest=10, roundR=FALSE)
```

Arguments

wide	numeric matrix with individuals in rows and time variable in columns (e.g., environments, genetic markers, etc.).
Gu	optional correlation matrix between the individuals or row levels. If NULL it will be computed as the correlation of t(wide).
nearest	integer value describing how many nearest neighbours (the ones showing the highest correlation) should be used to average and return the imputed value.
roundR	a TRUE/FALSE statement describing if the average result should be rounded or not. This may be specifically useful for categorical data in the form of numbers (e.g., -1,0,1).

Value

$res a list with the imputed matrix and the original matrix.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
###################################
### imputing genotype data example
###################################
# data(DT_cpdata, package="enhancer")
# X <- GT_cpdata
# # add missing data
# v <- sample(1:length(X), 500)
# Xna <- X
# Xna[v]<- NA
# ## impute (can take some time)
# Y <- corImputation(wide=Xna, Gu=NULL, nearest=20, roundR=TRUE)
# cm <- table(Y$imputed[v],X[v])
# ## calculate accuracy
# sum(diag(cm))/length(v)
###################################
### imputing phenotypic data example
###################################
# data(DT_h2, package="enhancer")
# X <- reshape(DT_h2[,c("Name","Env","y")], direction = "wide", idvar = "Name",
#                 timevar = "Env", v.names = "y", sep= "_")
# rownames(X) <- X$Name
# X <- as.matrix(X[,-1])
# head(X)
# # add missing data
# v <- sample(1:length(X), 50)
# Xna <- X
# Xna[v]<- NA
# ## impute
# Y <- corImputation(wide=Xna, Gu=NULL, nearest=20, roundR=TRUE)
# plot(y=Y$imputed[v],x=X[v], xlab="true",ylab="predicted")
# cor(Y$imputed[v],X[v], use = "complete.obs")
```

covm *Covariance Between Two Random Effects*

Description

covm combines two random-effect structures created with vsm into a single correlated random-effect structure. It fits an unstructured 2×2 covariance matrix between the two random effects while allowing them to have different incidence matrices. The two effects must act on the same coefficient space and use the same relationship or precision matrix. The resulting structure can be fitted with the mmes solver.

Usage

```
covm(ran1, ran2, thetaC = NULL, theta = NULL,
fixed = NULL, fixedSigma2 = FALSE,
labels = c("ran1", "ran2"), tol = 1e-10)
```

Arguments

ran1	A random-effect structure returned by vsm for the first random effect. Currently, covm requires a simple vsm structure with one covariance-product coordinate, for example vsm(ism(focal)) or vsm(ism(focal), Gu = Ai).
ran2	A random-effect structure returned by vsm for the second random effect. It must have the same coefficient dimension and coefficient levels as ran1, although its incidence matrix may differ.
thetaC	Deprecated and not supported by the current CovarianceFactor parameterization. The former cell-wise constraint matrix cannot in general be translated exactly into the normalized-Cholesky parameterization used by covm. Use fixedSigma2 and fixed instead.
theta	An optional symmetric 2×2 positive-definite matrix containing initial values for the variance-covariance matrix of the two random effects. Values are supplied directly on the natural variance-covariance scale. “‘ The diagonal elements define the initial variances of the two random effects and the off-diagonal element defines their initial covariance. If theta = NULL, the default starting matrix is ``` diag(2) * 0.15 + matrix(0.015, 2, 2) “‘ giving initial variances of 0.165 and an initial covariance of 0.015. “‘
fixed	An optional logical vector of length two indicating whether the two normalized-Cholesky coordinates describing the covariance structure should be fixed at their starting values. The coordinates correspond to the off-diagonal Cholesky element and the second diagonal Cholesky element, respectively. The default is c(FALSE, FALSE), so both are estimated.

`fixedSigma2`	Logical value indicating whether the variance scale associated with the first random effect should be fixed at its starting value. The default is `FALSE`.
`labels`	Character vector of length two giving labels for the two random effects in the covariance descriptor. The default is `c("ran1", "ran2")`. The labels must be different and non-empty.
`tol`	Numerical tolerance used when checking positive definiteness and whether the relationship/precision matrices supplied by the two random effects are equal. The default is `1e-10`.

Details

`covm` is intended for models in which two distinct random effects are expected to be correlated. A common example is the joint modeling of direct and indirect genetic effects.

If the coefficient vector for the first random effect is u_1 and that for the second is u_2, `covm` represents their covariance structure as

$$\mathrm{Var}([u_1', u_2']') = \sigma^2 K_{\text{effect}} \otimes A,$$

where A represents the covariance structure associated with the coefficient levels, and K_{effect} is a 2×2 unstructured covariance shape describing the relationship between the two random effects.

Internally, `covm` uses the same CovarianceFactor interface as `vsm`. The 2×2 effect covariance is represented using a normalized-Cholesky parameterization. This guarantees a positive-definite covariance matrix during optimization.

The variance of the first random effect is represented by the overall scale parameter `sigma2`. The remaining variance ratio and covariance are represented through the normalized-Cholesky factor.

The two random effects may have different incidence matrices. However, they must refer to identical coefficient levels in the same order and must use the same relationship/precision matrix.

When a relationship matrix is supplied through `Gu`, the Henderson solver requires it to be an inverse/precision matrix with `attr(Gu, "inverse") = TRUE`. For example:

```
attr(Ai, "inverse") <- TRUE

covm(
vsm(ism(focal), Gu = Ai),
vsm(ism(neighbour), Gu = Ai)
)
```

The current implementation combines simple `vsm` random effects only. Each input must contain one covariance-product coordinate. More complex covariance-factor products should therefore not currently be supplied independently inside `ran1` and `ran2`.

Value

A list describing a correlated random-effect structure compatible with the CovarianceFactor-v2 interface used by `vsm` and `mmes`. The main elements are:

‘‘‘

`Z` A list containing the two incidence matrices, one for each random effect.

`Gu` The common sparse relationship precision matrix associated with the coefficient levels. It is stored with `attr(Gu, "inverse") = TRUE` for use by the Henderson solver.

`covStruct` A CovarianceFactor-v2 descriptor containing the overall variance scale and the normalized-Cholesky representation of the 2×2 unstructured covariance between the two random effects.

`residualLocalIndex` `NULL`, since `covm` describes a random-effect structure rather than a residual covariance structure.

`productDesign` `NULL` for the current implementation.

`partitionsR` `NULL` for the current implementation.

`covm` Logical value equal to `TRUE`, identifying the object as one constructed by `covm`.

`covm_labels` The two labels used for the correlated random effects.
‘‘‘

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016). Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6). doi:10.1371/journal.pone.0156744

Bijma, P. (2014). The quantitative genetics of indirect genetic effects: a selective review of modelling issues. Heredity, 112(1), 61–69.

See Also

`vsm` for constructing random-effect covariance structures and `mmes` for fitting mixed models using the Henderson solver.

Examples

```
data(DT_ige, package = "enhancer")
DT <- DT_ige

## Correlate two random effects with identity relationship matrices

covRes <- with(
DT,
covm(
vsm(ism(focal)),
vsm(ism(neighbour))
)
)

str(covRes)

## Custom initial variance-covariance matrix
```

```
covRes2 <- with(
DT,
covm(
vsm(ism(focal)),
vsm(ism(neighbour)),
theta = matrix(
c(0.5, 0.1,
0.1, 0.3),
2, 2
)
)
)

## Relationship precision matrix

## Ai must have matching row and column names and be marked as inverse.

##

## attr(Ai, "inverse") <- TRUE

##

## covRes3 <- with(

## DT,

## covm(

## vsm(ism(focal), Gu = Ai),

## vsm(ism(neighbour), Gu = Ai)

## )

## )

## See the DT_ige help page for a complete model-fitting example.
```

`covmatrix_mmes` *Covariance and correlation matrix of a fitted term with standard errors*

Description

Returns the full covariance matrix of a random or residual term of a fitted `mmes` model over its covariance coordinates (for example the genetic trait-by-trait covariance matrix of a multi-trait model), together with the implied correlation matrix and delta-method standard errors of every element.

Usage

```
covmatrix_mmes(object, term, se = TRUE, max.dim = 500L, rel_step = 1e-5)
```

Arguments

object	A fitted mmes model.
term	Name (as in names(object$covStruct)) or position of the term. The residual term is the last one.
se	If TRUE, add standard errors.
max.dim	Refuse terms whose covariance coordinate space is larger than this (the matrices are dense).
rel_step	Relative step of the central differences used for the Jacobian.

Details

The matrix is $\sigma^2 K_1 \otimes \cdots \otimes K_m$ built from all the covariance factors of the vsm term (the relationship matrix Gu is not included), so for vsm(usm(trait), ism(id)) it is the trait covariance matrix, and for vsm(dsm(env), usm(trait), ism(id)) the environment-by-trait matrix. Row and column names are the coordinate levels.

Standard errors use the delta method, $\mathrm{Var}(g(\hat{\theta})) \approx J V J^\top$, where V is object$theta_se (the covariance of the reported parameters, as used by covparams_mmes_se) and J the Jacobian of each element with respect to the reported parameters. Fixed parameters contribute no variance. Diagonal standard errors of the correlation matrix are zero. Terms split by dsumm sections are not supported.

Value

A list with covariance and correlation and, if se=TRUE, covariance.se and correlation.se.

See Also

covparams_mmes, covparams_mmes_se, vpredict, stackTraits

Examples

```
set.seed(17)
n <- 60
DT <- data.frame(id = factor(rep(seq_len(n), each = 3)))
u <- matrix(rnorm(n * 2), n)
DT$trait_a <- u[DT$id, 1] + rnorm(nrow(DT), sd = 0.7)
DT$trait_b <- u[DT$id, 2] + rnorm(nrow(DT), sd = 1)
DTL <- stackTraits(DT, traits = c("trait_a", "trait_b"), keep = "id")
fit <- mmes(value ~ trait,
  random = ~ vsm(usm(trait), ism(id)),
  rcov = ~ vsm(usm(trait), ism(record)),
  data = DTL,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
G <- covmatrix_mmes(fit, 1)       # genetic
G$covariance
G$correlation
```

```
G$correlation.se
R <- covmatrix_mmes(fit, 2)       # residual
R$correlation
```

`covparams_mmes` *Extract native-scale covariance parameters from an mmes fit*

Description

Returns covariance parameters in the native, model-facing scale defined by each CovarianceFactor descriptor, rather than the unconstrained working coordinates or normalized variance ratios used internally by the optimizer.

Usage

```
covparams_mmes(object, term = NULL)
```

Arguments

`object`	A fitted object of class `"mmes"`.
`term`	Optional covariance-term name or numeric index. By default, all random and residual covariance terms are returned.

Details

Every CovarianceFactor descriptor carries a native reporting callback. Built-in structures use it to report meaningful quantities such as level-specific variances for `dsm()`, variances and covariances for `usm()`, and variance-correlation parameters for correlation structures. Structural zeros and internal coordinates such as log standard deviations, normalized Cholesky coefficients, and variance ratios are not returned where the covariance model provides a more direct native representation.

All built-in CovarianceFactor families provide native reporters, including identity, diagonal, unstructured, AR(1–3), compound-symmetry, moving-average, general-correlation, factor-analytic, antedependence, reduced-rank, Matern, Toeplitz, SAR, and CAR structures. Heterogeneous AR and compound-symmetry models report their dependence parameter together with level-specific variances rather than an overall variance plus variance ratios.

Both fixed and estimated covariance parameters are included. For an arbitrary Kronecker product, the product-level variance scale is absorbed by the first covariance-shaping factor. Subsequent factors are necessarily reported on their normalized relative scale because the product has only one identifiable overall scale.

User-defined `ownm()` structures can supply a `native_report` callback. If none is supplied, their overall variance and naturally transformed parameter values are returned.

Use `covparams_mmes_se()` when native-scale delta-method standard errors are also required.

Value

A data frame with columns `term`, `factor`, `section`, `parameter`, and `estimate`. `section` identifies the `dsumm` section a parameter belongs to and is `NA` otherwise.

See Also

covparams_mmes_se, mmes, vsm, ownm

Examples

```
## Not run:
fit <- mmes(Yield ~ Env,
            random = ~ vsm(dsm(Env), ism(Name)),
            rcov = ~ units, data = DT_example)
covparams_mmes(fit)
fit$covParNative

## End(Not run)
```

---

covparams_mmes_se *Native-scale covariance parameters and standard errors*

Description

Returns descriptor-defined native covariance parameters with delta-method standard errors computed from the fitted covariance-parameter uncertainty matrix.

Usage

```
covparams_mmes_se(object, term = NULL, rel_step = 1e-6)
```

Arguments

object	A fitted object of class "mmes".
term	Optional covariance-term name or numeric index. By default, all random and residual covariance terms are returned.
rel_step	Positive relative step used to numerically differentiate each CovarianceFactor native reporting callback.

Details

The fitted theta_se matrix describes uncertainty in the reported covPar coordinates. For each covariance structure, covparams_mmes_se numerically evaluates the Jacobian of its descriptor's native reporting callback and applies the multivariate delta method,

$$V_{native} = JV_{covPar}J^{\mathsf{T}}.$$

The Jacobian is assembled jointly for all selected terms, preserving cross-term covariance in the returned vcov attribute rather than assuming that terms are independent. Consequently, coupled transformations use the complete covariance information. For example, the standard error of a dsm() level variance accounts for uncertainty and covariance in both the overall variance and its variance ratio. Likewise, usm() standard errors account for all relevant normalized Cholesky parameters.

The transformation is available for every built-in CovarianceFactor family supported by covparams_mmes(); user-defined ownm() structures use their descriptor-provided native_report callback.

Parameters marked fixed in the covariance descriptor contribute no independent sampling uncertainty: their rows and columns in theta_se are excluded before transformation. A native quantity involving both fixed and free parameters can still have a nonzero standard error through its free parameters. Both fixed and estimated native quantities are returned. Unavailable uncertainty affects only native quantities that depend on the corresponding parameters, not independent reported terms.

The same table is stored automatically in object$covParNativeSE by mmes().

Value

A data frame containing term, factor, parameter, estimate, StdError, and Zratio. A zero standard error is reported for a fully fixed native quantity, with Zratio=NA. The vcov attribute contains the full native-scale covariance matrix in the same order as the returned rows, including covariance between terms.

See Also

covparams_mmes, mmes, vsm

Examples

```
## Not run:
fit <- mmes(Yield ~ Env,
            random = ~ vsm(dsm(Env), ism(Name)),
            rcov = ~ units, data = DT_example)
covparams_mmes_se(fit)
fit$covParNativeSE

## End(Not run)
```

---

csm *Compound-symmetry covariance structure*

Description

Compound-symmetry covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
csm(x, rho = 0.10, fixed = FALSE,
    variance = c("homogeneous", "heterogeneous"), values = NULL)
```

Arguments

x	Variable defining q covariance levels.
rho	Starting common off-diagonal correlation.
fixed	For homogeneous variance, a logical indicating whether rho is fixed. For heterogeneous variance, a logical vector of length q: rho followed by q-1 variance-ratio parameters.
variance	Whether the correlation has homogeneous or heterogeneous marginal variances.
values	Optional q positive starting variances when variance = "heterogeneous".

Details

With variance = "homogeneous", csm uses unit diagonal and a common off-diagonal correlation,

$$K_{ii} = 1, \quad K_{ij} = \rho \; (i \neq j).$$

With variance = "heterogeneous", it uses

$$K = DC(\rho)D,$$

where the first variance is the reference and the remaining q-1 variances are estimated as positive ratios. Positive definiteness requires $-1/(q-1) < \rho < 1$. The product-level variance remains owned by vsm.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

corgm, dsm, vsm.

Examples

```
## Not run:
vsm(csm(environment), ism(genotype))

## End(Not run)
```

csr *User-specified covariance-component structure for* mmer *and* vsr

Description

Combines a design/incidence representation with a user-supplied covariance-component constraint matrix for use by the mmer covariance-model interface, typically inside vsr.

Usage

```
csr(x, mm)
```

Arguments

x	A factor, character vector, numeric vector, or design/incidence matrix defining the covariance dimension. Factor and character inputs are expanded to incidence columns. A matrix is used directly.
mm	A square covariance-component constraint matrix. Its dimensions must correspond to the columns of the design matrix generated from x. The matrix is returned as thetaC, with its row and column names set to the column names of the resulting design matrix.

Details

csr() provides the general constraint-matrix interface underlying custom covariance patterns for mmer/vsr. Unlike dsr and usr, it does not construct a predefined diagonal or unstructured pattern. Instead, the supplied matrix mm determines the covariance-component structure through thetaC.

Conceptually, thetaC identifies which elements of the covariance matrix are associated with estimated variance-covariance components and which constraints are shared among elements. The interpretation of the entries therefore follows the thetaC convention used by vsr.

When x is a matrix it is used directly as Z. Otherwise, factors and character vectors are converted to incidence matrices, while numeric non-factor inputs are treated as a single design column.

The function assigns the column names of Z to both dimensions of mm; therefore mm should be conformable with the number of columns of the resulting design matrix.

Value

A list with components:

- Z: the design/incidence matrix associated with the covariance dimension.
- thetaC: the user-supplied covariance-component constraint matrix, with dimensions named according to Z.

See Also

vsr, dsr, usr, atr, mmer

Examples

```
# A custom two-level covariance-component pattern
x <- factor(c("A","B","A","B"))
mm <- matrix(c(1, 2,
               2, 1), 2, 2, byrow = TRUE)
C <- csr(x, mm)
C$Z
C$thetaC
```

`D.mat` *Dominance relationship matrix*

Description

C++ implementation of the dominance matrix. Calculates the realized dominance relationship matrix. Can help to increase the prediction accuracy when 2 conditions are met; 1) The trait has intermediate to high heritability, 2) The population contains a big number of individuals that are half or full sibs (HS & FS).

Usage

```
D.mat(X,nishio=TRUE,min.MAF=0,return.imputed=FALSE)
```

Arguments

`X`	Matrix ($n \times m$) of unphased genotypes for n lines and m biallelic markers, coded as {-1,0,1}. Fractional (imputed) and missing values (NA) are allowed.
`nishio`	If TRUE, Nishio and Satoh (2014), which uses the orthogonal parameterization of Vitezica et al. (2013). Otherwise Su et al. (2012). See Details and references.
`min.MAF`	Minimum minor allele frequency, a finite value between zero and 0.5. Markers at or below this threshold and all constant markers are removed.
`return.imputed`	When TRUE, the imputed marker matrix is returned.

Details

Missing values are replaced by the column (marker) mean before the matrix is computed. At least two individuals and one variable marker surviving the frequency filter are required. Genotypes must lie in $[-1, 1]$; a marker with no observed genotypes cannot be imputed. Markers with minor allele frequency less than or equal to `min.MAF` and all constant markers are removed. For each marker k, p_k is the frequency of the allele counted by the code 1, estimated as the column mean of $(X+1)/2$, and $q_k = 1 - p_k$.

For the nishio method (Nishio and Satoh, 2014; Vitezica et al., 2013): genotypes are coded as $w_{ik} = -2q_k^2$ for $x_{ik} = 1$, $2p_kq_k$ for $x_{ik} = 0$ and $-2p_k^2$ for $x_{ik} = -1$, computed as $w_{ik} = -x_{ik}^2 + (p_k - q_k)x_{ik} + 2p_kq_k$. Fractional values are passed through the same formula; note that a mean-imputed value $x_{ik} = p_k - q_k$ receives the heterozygote code $2p_kq_k$. Then $D = WW'/c$,

where $c = \sum_k (2p_k q_k)^2$. Under Hardy-Weinberg equilibrium this coding is orthogonal to the additive coding.

For the Su method (Su et al., 2012): heterozygosity is coded as $h_{ik} = 1 - |x_{ik}|$ and centered as $m_{ik} = h_{ik} - 2p_k q_k$. Then $D = MM'/c$, where $c = \sum_k 2p_k q_k(1 - 2p_k q_k)$.

Versions of sommer earlier than 4.4.9 used a different formula for `nishio=TRUE`: h_{ik} centered by its column mean and divided by $\sum_k (2p_k q_k)^2$. That was not the Nishio and Satoh (2014) method, so results from those versions will differ.

Value

If return.imputed = FALSE, the $n \times n$ dominance relationship matrix is returned.

If return.imputed = TRUE, the function returns a list containing

$D the D matrix

$X the imputed marker matrix

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Nishio M and Satoh M. 2014. Including Dominance Effects in the Genomic BLUP Method for Genomic Evaluation. Plos One 9(1), doi:10.1371/journal.pone.0085792

Vitezica ZG, Varona L, Legarra A. 2013. On the Additive and Dominant Variance and Covariance of Individuals Within the Genomic Selection Scope. Genetics 195(4): 1223-1230. doi:10.1534/genetics.113.155176

Su G, Christensen OF, Ostersen T, Henryon M, Lund MS. 2012. Estimating Additive and Non-Additive Genetic Variances and Predicting Genetic Merits Using Genome-Wide Dense Single Nucleotide Polymorphism Markers. PLoS ONE 7(9): e45293. doi:10.1371/journal.pone.0045293

Examples

```
####============================================####
#### EXAMPLE 1
####============================================####
####random population of 200 lines with 1000 markers
X <- matrix(rep(0,200*1000),200,1000)
for (i in 1:200) {
  X[i,] <- sample(c(-1,0,0,1), size=1000, replace=TRUE)
}

D <- D.mat(X)
```

`dfToMatrix` *data frame to matrix*

Description

This function takes a matrix that is in data frame format and transforms it into a matrix. Other packages that allows you to obtain an additive relationship matrix from a pedigree is the 'pedigreemm' package.

Usage

```
dfToMatrix(x, row="Row",column="Column",
           value="Ainverse", returnInverse=FALSE,
           bend=1e-6)
```

Arguments

`x`	ginv element, output from the Ainverse function.
`row`	name of the column in x that indicates the row in the original relationship matrix.
`column`	name of the column in x that indicates the column in the original relationship matrix.
`value`	name of the column in x that indicates the value for a given row and column in the original relationship matrix.
`returnInverse`	a TRUE/FALSE value indicating if the inverse of the x matrix should be computed once the data frame x is converted into a matrix.
`bend`	a numeric value to add to the diagonal matrix in case matrix is singular for inversion.

Value

`K`	pedigree transformed in a relationship matrix.
`Kinv`	inverse of the pedigree transformed in a relationship matrix.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
library(Matrix)
m <- matrix(1:9,3,3)
m <- tcrossprod(m)

mdf <- as.data.frame(as.table(m))
mdf

dfToMatrix(mdf, row = "Var1", column = "Var2",
            value = "Freq",returnInverse=FALSE )
```

`dsm` *Diagonal heterogeneous covariance structure*

Description

Diagonal heterogeneous covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
dsm(x, values = NULL, fixed = NULL, theta = NULL)
```

Arguments

`x`	Variable or design defining the covariance levels.
`values`	Optional positive starting variances, one per level.
`fixed`	Logical vector of length q-1 indicating which variance-ratio parameters are fixed.
`theta`	Optional q by q diagonal starting covariance matrix; if supplied, its diagonal replaces values.

Details

For q levels, `dsm` uses the first variance as the internal scale reference and estimates $q - 1$ positive variance ratios,

$$K = \mathrm{diag}(1, r_2, \ldots, r_q).$$

The ratios are optimized on log scales and reported on their natural positive scale.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

vsm, mmes.

Examples

```
## Not run:
vsm(dsm(environment), ism(genotype))

## End(Not run)
```

---

dsr *Diagonal covariance structure for* mmer *and* vsr

Description

Creates a diagonal variance-component structure for use by the mmer covariance-model interface, typically inside vsr. Each column or level represented by x receives its own diagonal variance component, while all off-diagonal covariance components are constrained to zero.

Usage

```
dsr(x)
```

Arguments

x	A factor, character vector, numeric vector, or design/incidence matrix defining the covariance dimension. Factor and character inputs are expanded to incidence columns. A matrix is used directly.

Details

dsr() is part of the covariance-structure interface used by mmer/vsr; it is distinct from dsm, which belongs to the newer mmes/vsm CovarianceFactor interface.

The function returns a design matrix Z and a covariance-parameter constraint matrix thetaC. If q columns or levels are represented by x, then

$$\text{thetaC} = I_q.$$

Consequently, the associated covariance matrix has the form

$$\Sigma = \text{diag}(\sigma_1^2, \ldots, \sigma_q^2),$$

so that the q variances are estimated separately and all covariances are fixed to zero.

For a factor or character vector, missing observations are retained through the construction of the incidence matrix. If only one non-missing level is present, a one-column incidence matrix is constructed explicitly.

For a numeric non-factor input, x is treated as a single design column rather than expanded into factor levels.

Value

A list with components:

- Z: the design/incidence matrix associated with the covariance dimension.
- thetaC: a diagonal matrix defining the diagonal covariance-component pattern used by vsr.

See Also

vsr, usr, csr, atr, dsm, mmer

Examples

```
# Diagonal covariance structure across levels
D <- dsr(factor(c("A","B","A","C")))
D$Z
D$thetaC
```

---

dsumm *Direct sum of residual covariance structures across sections*

Description

dsumm replicates a residual vsm structure independently for every level (section) of a grouping variable, such as a trial, location, or experiment. Each section receives its own variance and its own copy of every covariance parameter of the inner structure.

Usage

```
dsumm(x, by, levels = NULL)
```

Arguments

x	A residual vsm() term, for example vsm(ar1m(range), ar1m(row), ism(units)). Its covariance-shaping factors define the structure fitted within each section.
by	Observation-level grouping variable defining the sections. Missing values of by exclude the observation.
levels	Optional character vector of by levels that receive the structured covariance of x. All other levels receive an independent and identically distributed residual with their own variance. The default NULL applies the structure to every level.

Details

Direct sum versus Kronecker product. A `vsm()` term is a Kronecker product with a single overall variance. Including a diagonal section factor,

$$\mathrm{vsm}(\mathrm{dsm}(t), \mathrm{ar1m}(r), \mathrm{ar1m}(c), \mathrm{ism}(units)): \quad R = \sigma^2\, D_t \otimes K_r(\rho_r) \otimes K_c(\rho_c),$$

gives each section its own variance, but a single correlation shared by all sections, because each factor of a Kronecker product appears only once. In contrast, `dsumm` builds the block-diagonal direct sum

$$R = \bigoplus_t \sigma_t^2\, K_r(\rho_{r,t}) \otimes K_c(\rho_{c,t}),$$

so every section has its own variance and its own correlations. Sections are mutually independent, so their parameters are estimated only from their own observations; with section-specific fixed effects the joint fit reproduces separate per-section fits.

Choosing between the two. Use the `vsm()` form when trials are small or similar and a common spatial correlation is adequate (fewer parameters). Use `dsumm` when correlations are expected to differ between trials. With T sections and p inner covariance parameters, `dsumm` estimates $T(p+1)$ residual parameters.

Unlisted levels. Levels omitted from `levels` are modelled as $\sigma_t^2 I$. Their observations are retained even when the variables of the inner structure (for example row or range) are missing, so trials without spatial coordinates can be analysed together with spatially designed trials.

Computation. Each section is embedded in the rectangle spanned by its observed levels of the inner factors. Covariance applications use the small factor inverses, and missing cells (e.g. plots with a missing response) are absorbed exactly by a Schur complement, so the dense observation-level covariance matrix is never formed. This requires sections that are close to complete rectangles and no `W` weights.

Reporting. `covparams_mmes` returns one row per parameter and section, with the section in the `section` column. The variance rows belong to the by factor; correlation rows are labelled by their inner factor, e.g. `ar1m(range)`.

Value

A residual covariance structure, in the same form as a residual `vsm()` term, for use in the `rcov` argument of `mmes`.

Limitations

`dsumm` is currently available only for residual (`rcov`) structures, without `W` weights, and cannot be nested.

See Also

`vsm`, `ar1m`, `dsm`, `mmes`, `covparams_mmes`.

Examples

```
## Not run:

mix <- mmes(yield ~ trial,
            rcov = ~ dsumm(vsm(ar1m(range), ar1m(row), ism(units)), by = trial),
            data = dat)
covparams_mmes(mix)

# Spatial structure only for two trials; remaining trials are iid.
mix2 <- mmes(yield ~ trial,
             rcov = ~ dsumm(vsm(ar1m(range), ar1m(row), ism(units)),
                            by = trial, levels = c("T1", "T2")),
             data = dat)

# Site-specific unstructured covariance among environments.
mix3 <- mmes(y ~ site,
             rcov = ~ dsumm(vsm(usm(env), ism(units)), by = site),
             data = dat2)

## End(Not run)
```

E.mat *Epistatic relationship matrix*

Description

Calculates the realized epistatic relationship matrix of second order (additive x additive, additive x dominance, or dominance x dominance) using hadamard products with the C++ Armadillo library.

Usage

```
E.mat(X,nishio=TRUE,type="A#A",min.MAF=0.02)
```

Arguments

X	Matrix ($n \times m$) of unphased genotypes for n lines and m biallelic markers, coded as {-1,0,1}. Fractional (imputed) and missing values (NA) are allowed.
nishio	If TRUE, Nishio and Satoh (2014) (orthogonal parameterization of Vitezica et al. 2013), otherwise Su et al. (2012) (see Details in the D.mat help page).
type	An argument specifying the type of epistatic relationship matrix desired. The default is the second order epistasis (additive x additive) type="A#A". Other options are additive x dominant (type="A#D"), or dominant by dominant (type="D#D").
min.MAF	Minimum minor allele frequency, a finite value between zero and 0.5. Markers at or below this threshold and all constant markers are removed. The default is 0.02.

Details

The result is the element-wise (Hadamard) product $E = A \circ A$, $E = A \circ D$, or $E = D \circ D$. Mean imputation, allele-frequency estimation and marker filtering are shared between the component matrices. Their definitions and input requirements are the same as in `A.mat` and `D.mat`. As in `A.mat`, the correction from inadvertent median imputation to mean imputation can change results for missing genotypes.

Value

The epistatic relationship matrix is returned.

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Endelman, J.B., and J.-L. Jannink. 2012. Shrinkage estimation of the realized relationship matrix. G3:Genes, Genomes, Genetics. 2:1405-1413. doi: 10.1534/g3.112.004259

Nishio M and Satoh M. 2014. Including Dominance Effects in the Genomic BLUP Method for Genomic Evaluation. Plos One 9(1), doi:10.1371/journal.pone.0085792

Su G, Christensen OF, Ostersen T, Henryon M, Lund MS. 2012. Estimating Additive and Non-Additive Genetic Variances and Predicting Genetic Merits Using Genome-Wide Dense Single Nucleotide Polymorphism Markers. PLoS ONE 7(9): e45293. doi:10.1371/journal.pone.0045293

Examples

```
####============================================####
####random population of 200 lines with 1000 markers
####============================================####
X <- matrix(rep(0,200*1000),200,1000)
for (i in 1:200) {
  X[i,] <- sample(c(-1,0,0,1), size=1000, replace=TRUE)
}

E <- E.mat(X, type="A#A")
# if heterozygote markers are present can be used "A#D" or "D#D"
```

`fam` *Factor-analytic covariance structure*

Description

Factor-analytic covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
fam(x, k = 1L, loadings = NULL, specific = NULL, fixed = NULL,
    variance = c("heterogeneous", "homogeneous"), shared = NULL)
```

Arguments

x	Variable defining q covariance levels.
k	Factor-analytic rank, satisfying 1 <= k < q.
loadings	Optional finite q by k starting loading matrix.
specific	Optional q positive starting specific variances. With variance="homogeneous", supply either one positive value or q equal values. The default is 0.5 for every level.
fixed	Logical vector controlling all loading and specific-variance-ratio parameters.
variance	Whether to estimate separate specific variances for each level ("heterogeneous", the default) or a common specific variance ("homogeneous").
shared	Optional non-empty loading-group name. Matching names on the single random and residual FA terms request common trait loadings with source-specific diagonal factor variances. Requires mmes_control(representation=list(equations="conditional")) for AI, or conditional EM through update=list(method="em") with that representation. The earlier update=list(method="conditional-fa") selector still selects EM. Ordinary equations reject shared tags rather than fitting independent loadings.

Details

The factor-analytic shape is based on

$$M = \Lambda\Lambda^{\mathsf{T}} + \Psi,$$

where Ψ is diagonal positive, followed by $K = M/M_{11}$. For rotational identification, the leading $k \times k$ loading block is lower triangular and its diagonal loadings are positive. The first specific variance is an internal reference; remaining specific variances are positive ratios. This removes the otherwise redundant internal scale because vsm supplies the overall σ^2.

With variance="heterogeneous", fixed has length $qk - k(k-1)/2 + q - 1$: triangular loadings followed by q-1 specific-variance ratios. With variance="homogeneous", all specific ratios are one, $\Psi = I_q$ internally, and fixed has length $qk - k(k-1)/2$. The common specific variance on the fitted scale is estimated, not fixed at one, and is reported as common_specific_variance. This homogeneous model reproduces the covariance family previously fitted by rrm(). For a genuine reduced-rank model with all specific variances fixed at zero, use rrm.

With matching shared groups, the conditional engine fits $G = LD_gL^{\mathsf{T}} + \Psi_g$ and $R = LD_eL^{\mathsf{T}} + \Psi_e$, with diagonal positive factor variances normalized to $D_g + D_e = I$. Both sources must be entirely free and have the same trait levels and rank. Starting loading columns, after applying each term's variance scale, must be positive scalar multiples across sources. The common loading matrix and factor variance allocations are returned in fit$conditionalFA$state. Marginal loading reports use independent triangular rotations of each source, not the common basis.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

rrm, usm, vsm.

Examples

```
## Not run:
vsm(fam(environment, k=2), ism(genotype))
vsm(fam(environment, k=2, variance="homogeneous"), ism(genotype))

## End(Not run)
```

`familym` *Different distribution families for the traits of a multi-trait model*

Description

Assigns a `stats::family()` object to each trait of a long-format multi-trait `mmes` model, e.g. a binary disease score together with a Gaussian yield, so that their genetic and residual correlations can be estimated jointly by penalized quasi-likelihood (PQL).

Usage

```
familym(by, ...)
```

Arguments

by	Name (character string) of the column of `data` that identifies the traits, typically `"trait"` as created by `stackTraits`.
...	Family objects named by the levels of the `by` column, e.g. `Disease = binomial()`, `Yield = gaussian()`. Every level present in the data needs a family.

Details

The object is passed as the `family` argument of `mmes`. Each PQL iteration builds the working response and working weights of every row with the family of its trait, and fits a weighted Gaussian multi-trait model.

The residual variance of binomial, Poisson and negative binomial traits is not a free parameter (their dispersion is one), while that of Gaussian and other families is estimated. The residual structure must therefore allow individual trait variances to be fixed: use `rcov = ~ vsm(corgm(trait, variance = "heterogeneous"), ism(record))` (trait correlations and heterogeneous variances, see `corgm`) or `rcov = ~ vsm(dsm(trait), ism(units))` (uncorrelated residuals). The first level of the trait factor must be a fixed-dispersion trait, because its variance is the `vsm` scale; reorder the levels otherwise. `mmes` then fixes the residual variance of every fixed-dispersion trait to one and estimates the rest.

As for single-family PQL fits, the likelihood, `AIC` and `BIC` are `NA`. The fitted object has a named `dispersion` vector (one per trait: one for fixed-dispersion traits, the REML residual variance for the others), and `fitted()` and `residuals()` apply the inverse link and variance function of each trait to its rows. Random-effect variances are on the link scale of each trait.

Value

An object of class "sommer_familym".

See Also

mmes, corgm, stackTraits, covmatrix_mmes

Examples

```
set.seed(12)
ng <- 100; nr <- 4
G <- matrix(c(1, 0.4, 0.4, 0.8), 2)
u <- matrix(rnorm(ng * 2), ng) %*% chol(G)
d <- data.frame(id = factor(rep(paste0("g", 1:ng), each = nr)))
d$disease <- rbinom(nrow(d), 1, plogis(-0.5 + u[as.integer(d$id), 1]))
d$yield <- 10 + u[as.integer(d$id), 2] + rnorm(nrow(d))
L <- stackTraits(d, traits = c("disease", "yield"))


fit <- mmes(value ~ trait,
  random = ~ vsm(usm(trait), ism(id)),
  rcov = ~ vsm(corgm(trait, variance = "heterogeneous"), ism(record)),
  family = familym("trait", disease = binomial(), yield = gaussian()),
  data = L,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
fit$dispersion
covmatrix_mmes(fit, 1)$correlation   # genetic correlation (link scale)
head(fitted(fit))                    # probabilities for disease rows
```

---

fitted.mmes *fitted form a LMM fitted with mmes*

Description

fitted method for class "mmes".

Usage

```
## S3 method for class 'mmes'
fitted(object, type=c("response", "link"), ...)
```

Arguments

object	an object of class "mmes"
type	For a non-Gaussian PQL fit, return fitted means on the response scale (default) or linear predictors on the link scale.
...	Further arguments to be passed to the mmes function

Value

For Gaussian fits, fitted values of the form y.hat = Xb + Zu. For non-Gaussian PQL fits, response-scale conditional means by default, or Xb + Zu with `type="link"`.

Author(s)

Giovanny Covarrubias

See Also

`fitted`, `mmes`

Examples

```
# data(DT_cpdata, package="enhancer")
# DT <- DT_cpdata
# GT <- GT_cpdata
# MP <- MP_cpdata
# #### create the variance-covariance matrix
# A <- A.mat(GT) # additive relationship matrix
# #### look at the data and fit the model
# head(DT)
# mix1 <- mmes(Yield~1,
#               random=~vsm(ism(id),Gu=A)
#                       + Rowf + Colf + spl2Dc(Row,Col),
#               rcov=~units,
#               data=DT)
#
# ff=fitted(mix1)
#
# colfunc <- colorRampPalette(c("steelblue4","springgreen","yellow"))
# lattice::wireframe(`u:Row.fitted`~Row*Col, data=ff$dataWithFitted,
#           aspect=c(61/87,0.4), drape=TRUE,# col.regions = colfunc,
#           light.source=c(10,0,10))
# lattice::levelplot(`u:Row.fitted`~Row*Col, data=ff$dataWithFitted, col.regions = colfunc)
```

`fixm` *fixed indication matrix*

Description

`fixm` creates a square matrix with 3's in the diagnals and off-diagonals to quickly specify a fixed constraint in the Gtc argument of the `vsm` function.

Usage

```
fixm(x, reps=NULL)
```

Arguments

x	integer specifying the number of traits to be fitted for a given random effect.
reps	integer specifying the number of times the matrix should be repeated in a list format to provide easily the constraints in complex models that use the ds(), us() or cs() structures.

Value

$res a matrix or a list of matrices with the constraints to be provided in the Gtc argument of the vsm function.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
fixm(4)
fixm(4,2)
```

---

generate_mme_workload *Generate Parameterized MME Workloads for Solver Evaluation*

Description

Instantiates reproducible synthetic mixed model datasets and formulas spanning factor dimensions, covariance structures, relationship densities, and tasks.

Usage

```
generate_mme_workload(name = NULL,
                      n_ind = 60L,
                      n_levels = 5L,
                      structure = c("ar1", "ar2", "us", "diag", "fa",
                                    "rr", "spatial", "sar", "car"),
                      relationship = c("none", "dense", "sparse"),
                      missing_rate = 0.0,
                      reml = TRUE,
                      computeCi = 0L,
                      seed = 42L)
```

Arguments

name	Descriptive label for the workload.
n_ind	Number of unique individuals / genotypes. Default is 60.
n_levels	Number of factor levels (e.g. environments, timepoints, or sites). Default is 5.
structure	Covariance structure type. Supported: "ar1", "ar2", "us", "diag", "fa", "rr", "spatial", "sar", "car".
relationship	Individual relationship matrix: "none" (identity), "dense" (genomic relationship inverse), or "sparse" (pedigree inverse).
missing_rate	Fraction of missing response values (0.0 to 0.5). Default is 0.0.
reml	Logical; whether to fit by REML (TRUE) or ML (FALSE). Default is TRUE.
computeCi	Final inverse demand: 0 (none), 1 (Takahashi selected inverse), 2 (full inverse).
seed	Random seed for reproducible dataset simulation.

Value

A list of class "mme_workload" containing data, formulas, REML and computeCi settings, and workload metadata.

See Also

mmes, mmes_control

Examples

```
# Generate a small 1D autoregressive workload
wl <- generate_mme_workload(
  name = "demo_ar1",
  n_ind = 20,
  n_levels = 4,
  structure = "ar1"
)
```

GWAS *Genome wide association study analysis*

Description

THIS FUNCTION IS DEPRECATED. Fits a multivariate/univariate linear mixed model GWAS by likelihood methods (REML), see the Details section below. It uses the mmer function and its core coded in C++ using the Armadillo library to optimize dense matrix operations common in the derect-inversion algorithms. After the model fit extracts the inverse of the phenotypic variance matrix to perform the association test for the "p" markers. Please check the Details section (Model enabled) if you have any issue with making the function run.

The sommer package is updated on CRAN every 3-months due to CRAN policies but you can find the latest source at https://github.com/covaruber/sommer . This can be easily installed typing the following in the R console:

library(devtools)

install_github("covaruber/sommer")

This is recommended since bugs fixes will be immediately available in the GitHub source. **For tutorials** on how to perform different analysis with sommer please look at the vignettes by typing in the terminal:

vignette("v1.sommer.quick.start")

vignette("v2.sommer.changes.and.faqs")

vignette("v3.sommer.qg")

vignette("v4.sommer.gxe")

or visit **https://covaruber.github.io**

Usage

```
GWAS(fixed, random, rcov, data, weights, W,
    nIters=20, tolParConvLL = 1e-03, tolParInv = 1e-06,
    init=NULL, constraints=NULL,method="NR",
    getPEV=TRUE,naMethodX="exclude",
    naMethodY="exclude",returnParam=FALSE,
    dateWarning=TRUE,date.warning=TRUE,verbose=FALSE,
    stepWeight=NULL, emWeight=NULL,
    M=NULL, gTerm=NULL, n.PC = 0, min.MAF = 0.05,
    P3D = TRUE)
```

Arguments

`fixed`	A formula specifying the **response variable(s) and fixed effects**, i.e: *response ~ covariate* for univariate models *cbind(response.i,response.j) ~ covariate* for multivariate models The fcm function can be used to constrain fixed effects in multi-response models.
`random`	a formula specifying the name of the **random effects**, i.e. *random= ~ genotype + year*. Useful functions can be used to fit heterogeneous variances and other special models (*see 'Special Functions' in the Details section for more information*): `vsr(...,Gu,Gt,Gtc)` is the main function to specify variance models and special structures for random effects. On the ... argument you provide the unknown variance-covariance structures (i.e. usr,dsr,at,csr) and the random effect where such covariance structure will be used (the random effect of interest). Gu is used to provide known covariance matrices among the levels of the random effect, Gt initial values and Gtc for constraints. Auxiliar functions for building the variance models are:

`rcov` a formula specifying the name of the **error term**, i.e. *rcov= ~ units*.
The functions that can be used to fit heterogeneous residual variances are the same used on the random term but the random effect is always "units", i.e. *rcov=~vsr(dsr(Location),units)*

`data` a data frame containing the variables specified in the formulas for response, fixed, and random effects.

`weights` name of the covariate for weights. To be used for the product R = Wsi*R*Wsi, where * is the matrix product, Wsi is the square root of the inverse of W and R is the residual matrix.

`W` Alternatively, instead of providing a vector of weights the user can specify an entire W matrix (e.g., when covariances exist). To be used first to produce Wis = solve(chol(W)), and then calculate R = Wsi*R*Wsi.t(), where * is the matrix product, and R is the residual matrix. Only one of the arguments weights or W should be used. If both are indicated W will be given the preference.

`nIters` Maximum number of iterations allowed. Default value is 15.

`tolParConvLL` Convergence criteria.

`tolParInv` tolerance parameter for matrix inverse used when singularities are encountered.

`init` initial values for the variance components. By default this is NULL and variance components are estimated by the method selected, but in case the user want to provide initial values for ALL var-cov components this argument is functional. It has to be provided as a list or an array, where each list element is one variance component and if multitrait model is pursued each element of the list is a matrix of variance covariance components among traits. Initial values can also be provided in the Gt argument of the vsr function.Is highly encouraged to use the Gt and Gtc arguments of the vsr function instead of this argument

`constraints` when initial values are provided these have to be accompanied by their constraints. See the vsr function for more details on the constraints. Is highly encouraged to use the Gt and Gtc arguments of the vsr function instead of this argument.

`method` this refers to the method or algorithm to be used for estimating variance components. Direct-inversion Newton-Raphson **NR** and Average Information **AI** (Tunnicliffe 1989; Gilmour et al. 1995; Lee et al. 2015).

`getPEV` a TRUE/FALSE value indicating if the program should return the predicted error variance and variance for random effects. This option is provided since this can take a long time for certain models where p > n by a big extent.

`naMethodX` one of the two possible values; "include" or "exclude". If "include" is selected then the function will impute the X matrices for fixed effects with the median value. If "exclude" is selected it will get rid of all rows with missing values for the X (fixed) covariates. The default is "exclude". The "include" option should be used carefully.

`naMethodY` one of the three possible values; "include", "include2" or "exclude". If "include" is selected then the function will impute the response variables with the median value. The difference between "include" and "include2" is only available in the multitrait models when the imputation can happen for the entire matrix of responses or only for complete cases ("include2"). If "exclude" is selected it will

get rid of rows in responses where missing values are present for the estimation of variance components. The default is "exclude".

`returnParam` a TRUE/FALSE value to indicate if the program should return the parameters used for modeling without fitting the model.

`dateWarning` a TRUE/FALSE value to indicate if the program should warn you when is time to update the sommer package.

`date.warning` a TRUE/FALSE value to indicate if the program should warn you when is time to update the sommer package.

`verbose` a TRUE/FALSE value to indicate if the program should return the progress of the iterative algorithm.

`stepWeight` A vector of values (of length equal to the number of iterations) indicating the weight used to multiply the update (delta) for variance components at each iteration. If NULL the 1st iteration will be multiplied by 0.5, the 2nd by 0.7, and the rest by 0.9. This argument can help to avoid that variance components go outside the parameter space in the initial iterations which doesn't happen very often with the NR method but it can be detected by looking at the behavior of the likelihood. In that case you may want to give a smaller weight to the initial 8-10 iterations.

`emWeight` A vector of values (of length equal to the number of iterations) indicating with values between 0 and 1 the weight assigned to the EM information matrix. And the values 1 - emWeight will be applied to the NR/AI information matrix to produce a joint information matrix. If NULL weights for EM information matrix are zero and 1 for the NR/AI information matrix.

`M` The marker matrix containing the marker scores for each level of the random effect selected in the gTerm argument, coded as numeric based on the number of reference alleles in the genotype call, e.g. (-1,0,1) = (aa,Aa,AA), levels in diploid individuals. Individuals in rows and markers in columns. No additional columns should be provided, is a purely numerical matrix. Similar logic applies to polyploid individuals, e.g. (-3,-2,-1,0,1,2,3) = (aaaa,aaaA,aaAA,Aaaa,AAaa,AAAa,AAAA).

`gTerm` a character vector indicating the random effect linked to the marker matrix M (i.e. the genetic term) in the model. The random effect selected should have the same number of levels than the number of rows of M. When fitting only a random effect without a special covariance structure (e.g., dsr, usr, etc.) you will need to add the call 'u:' to the name of the random effect given the behavior of the naming rules of the solver when having a simple random effect without covariance structure.

`n.PC` Number of principal components to include as fixed effects. Default is 0 (equals K model).

`min.MAF` Specifies the minimum minor allele frequency (MAF). If a marker has a MAF less than min.MAF, it is assigned a zero score.

`P3D` When P3D=TRUE, variance components are estimated by REML only once, without any markers in the model and then a for loop for hypothesis testing is performed. When P3D=FALSE, variance components are estimated by REML for each marker separately. The latter can be quite time consuming. As many models will be run as number of marker.

Details

Citation

Type *citation("sommer")* to know how to cite the sommer package in your publications.

Models Enabled

For details about the models enabled and more information about the covariance structures please check the help page of the package (sommer). In general the GWAS model implemented in sommer to obtain marker effect is a generalized linear model of the form:

b = (X'V-X)X'V-y

with X = ZMi

where: b is the marker effect (dimensions 1 x mt) y is the response variable (univariate or multivariate) (dimensions 1 x nt) V- is the inverse of the phenotypic variance matrix (dimensions nt x nt) Z is the incidence matrix for the random effect selected (gTerm argument) to perform the GWAS (dimensions nt x ut) Mi is the ith column of the marker matrix (M argument) (dimensions u x m)

for t traits, n observations, m markers and u levels of the random effect. Depending if P3D is TRUE or FALSE the V- matrix will be calculated once and used for all marker tests (P3D=TRUE) or estimated through REML for each marker (P3D=FALSE).

vignette('sommer.start')

Bug report and contact

If you have any technical questions or suggestions please post it in https://stackoverflow.com or https://stats.stackexchange.com.

If you have any bug report please go to https://github.com/covaruber/sommer or send me an email to address it asap.

Value

If all parameters are correctly indicated the program will return a list with the following information:

`Vi`	the inverse of the phenotypic variance matrix V^- = (ZGZ+R)^-1
`sigma`	a list with the values of the variance-covariance components with one list element for each random effect.
`sigma_scaled`	a list with the values of the scaled variance-covariance components with one list element for each random effect.
`sigmaSE`	Hessian matrix containing the variance-covariance for the variance components. SE's can be obtained taking the square root of the diagonal values of the Hessian.
`Beta`	a data frame for trait BLUEs (fixed effects).
`VarBeta`	a variance-covariance matrix for trait BLUEs
`U`	a list (one element for each random effect) with a data frame for trait BLUPs.
`VarU`	a list (one element for each random effect) with the variance-covariance matrix for trait BLUPs.
`PevU`	a list (one element for each random effect) with the predicted error variance matrix for trait BLUPs.
`fitted`	Fitted values y.hat=XB

residuals	Residual values e = Y - XB
AIC	Akaike information criterion
BIC	Bayesian information criterion
convergence	a TRUE/FALSE statement indicating if the model converged.
monitor	The values of log-likelihood and variance-covariance components across iterations during the REML estimation.
scores	marker scores (-log_(10)p) for the traits
method	The method for extimation of variance components specified by the user.
constraints	contraints used in the mixed models for the random effects.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G. Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 2016, 11(6): doi:10.1371/journal.pone.0156744

Covarrubias-Pazaran G. 2018. Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639

Bernardo Rex. 2010. Breeding for quantitative traits in plants. Second edition. Stemma Press. 390 pp.

Gilmour et al. 1995. Average Information REML: An efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51(4):1440-1450.

Kang et al. 2008. Efficient control of population structure in model organism association mapping. Genetics 178:1709-1723.

Lee, D.-J., Durban, M., and Eilers, P.H.C. (2013). Efficient two-dimensional smoothing with P-spline ANOVA mixed models and nested bases. Computational Statistics and Data Analysis, 61, 22 - 37.

Lee et al. 2015. MTG2: An efficient algorithm for multivariate linear mixed model analysis based on genomic information. Cold Spring Harbor. doi: https://doi.org/10.1101/027201.

Maier et al. 2015. Joint analysis of psychiatric disorders increases accuracy of risk prediction for schizophrenia, bipolar disorder, and major depressive disorder. Am J Hum Genet; 96(2):283-294.

Rodriguez-Alvarez, Maria Xose, et al. Correcting for spatial heterogeneity in plant breeding experiments with P-splines. Spatial Statistics 23 (2018): 52-71.

Searle. 1993. Applying the EM algorithm to calculating ML and REML estimates of variance components. Paper invited for the 1993 American Statistical Association Meeting, San Francisco.

Yu et al. 2006. A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. Genetics 38:203-208.

Tunnicliffe W. 1989. On the use of marginal likelihood in time series model estimation. JRSS 51(1):15-27.

Zhang et al. 2010. Mixed linear model approach adapted for genome-wide association studies. Nat. Genet. 42:355-360.

Examples

```
####=========================================####
#### For CRAN time limitations most lines in the
#### examples are silenced with one '#' mark,
#### remove them and run the examples using
#### command + shift + C |OR| control + shift + C
####=========================================####
#####=========================================####
##### potato example
#####=========================================####
#
# data(DT_polyploid, package="enhancer")
# DT <- DT_polyploid
# GT <- GT_polyploid
# MP <- MP_polyploid
# ####=========================================####
# ####### convert markers to numeric format
# ####=========================================####
# numo <- atcg1234(data=GT, ploidy=4);
# numo$M[1:5,1:5];
# numo$ref.allele[,1:5]
#
# ###=========================================####
# ###### plants with both genotypes and phenotypes
# ###=========================================####
# common <- intersect(DT$Name,rownames(numo$M))
#
# ###=========================================####
# ### get the markers and phenotypes for such inds
# ###=========================================####
# marks <- numo$M[common,]; marks[1:5,1:5]
# DT2 <- DT[match(common,DT$Name),];
# DT2 <- as.data.frame(DT2)
# DT2[1:5,]
#
# ###=========================================####
# ###### Additive relationship matrix, specify ploidy
# ###=========================================####
# A <- A.mat(marks)
# ###=========================================####
# ### run it as GWAS model
# ###=========================================####
# ans2 <- GWAS(tuber_shape~1,
#              random=~vsr(Name,Gu=A),
#              rcov=~units,
#              gTerm = "u:Name",
#              M=marks, data=DT2)
# plot(ans2$scores[,1])
```

`H.mat` *Combined relationship matrix H*

Description

Given a matrix A and a matrix G returns a H matrix with the C++ Armadillo library.

Usage

```
H.mat(A, G, tau = 1, omega = 1, tolparinv=1e-6)
```

Arguments

A	Finite symmetric additive relationship matrix based on pedigree. Row and column names must be identical, non-empty and unique.
G	Finite symmetric additive relationship matrix based on marker data. Row and column names must be identical, non-empty and unique; every individual must also occur in A.
tau	As described by Martini et al. (2018).
omega	As described by Martini et al. (2018).
tolparinv	Finite positive diagonal increment used for one retry when a required positive-definite factorization fails. An error is raised if the retry also fails.

Details

Let subscripts 1 and 2 denote non-genotyped and genotyped individuals, respectively. The genotyped block is $H_{22} = [\tau G_{22}^{-1} + (1-\omega)A_{22}^{-1}]^{-1}$. The remaining corrections are computed using Cholesky solves with A_{22}, rather than explicitly forming its inverse for the default weights. When `omega=1` and `tau>0`, $H_{22} = G_{22}/\tau$; redundant inversions are avoided while retaining the diagonal retry for singular inputs. The retry increment is applied separately to A_{22}, G_{22}, or the genotyped precision block when required.

The returned matrix is dense. Its row and column order is `rev(unique(c(rownames(G), rownames(A))))`, including when there is just one genotyped individual.

Value

H Matrix with the relationship between the individuals based on pedigree and corrected by molecular information

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Martini, J. W., Schrauf, M. F., Garcia-Baccino, C. A., Pimentel, E. C., Munilla, S., Rogberg-Munoz, A., ... & Simianer, H. (2018). The effect of the H-1 scaling factors tau and omega on the structure of H in the single-step procedure. Genetics Selection Evolution, 50(1), 16.

Examples

```
####============================================####
####random population of 200 lines with 1000 markers
####============================================####
M <- matrix(rep(0,200*1000),200,1000)
for (i in 1:200) {
  M[i,] <- sample(c(-1,0,0,1), size=1000, replace=TRUE)
}
rownames(M) <- 1:nrow(M)
v <- sample(1:nrow(M),100)
M2 <- M[v,]

A <- A.mat(M) # assume this is a pedigree-based matrix for the sake of example
G <- A.mat(M2)

H <- H.mat(A,G)
# colfunc <- colorRampPalette(c("steelblue4","springgreen","yellow"))
# hv <- heatmap(H[1:15,1:15], col = colfunc(100),Colv = "Rowv")
```

`ism` *Identity covariance structure*

Description

Identity covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
ism(x)
```

Arguments

`x`	Factor, character vector, numeric design, or matrix defining covariance levels.

Details

`ism` returns $K = I$. It has no covariance-shape parameters; the only unknown covariance scale is the σ^2 supplied by `vsm`. It is commonly used as the final term of `vsm` to indicate an independent main effect.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`vsm`, `mmes`.

Examples

```
## Not run:
vsm(ism(id))
vsm(dsm(environment), ism(genotype))

## End(Not run)
```

`loadings_mmes` *Extract factor-analytic loadings from a fitted mmes model*

Description

Reconstructs the normalized loadings matrix and specific variances of a `fam()` or `rrm()` covariance-shaping factor fitted inside `vsm()`, on the same scale as the fitted covariance matrix returned in `object$theta`.

Usage

```
loadings_mmes(object, term = NULL, varianceScale=TRUE, rotation=TRUE)
```

Arguments

`object`	a fitted model of class `"mmes"`.
`term`	character name of the random term (matching a name in `object$covStruct`) built with a single `fam()` or `rrm()` covariance-shaping factor. If `NULL`, the unique such term in the model is used automatically; an error is raised if none or more than one exist.
`varianceScale`	Whether to return loadings multiplied by `sqrt(sigma2)` and specific variances multiplied by `sigma2`. When `FALSE`, both describe the dimensionless covariance shape.
`rotation`	a logical value to indicate if loadings should be rotated by its singular vectors.

Details

`fam()`/`rrm()` remove their otherwise redundant internal scale so that `vsm()` owns the single overall variance; the loadings and specific variances reported by `loadings_mmes` are re-normalized so that

$$\Sigma = \sigma^2(\Lambda\Lambda^{\mathsf{T}} + \Psi)$$

reproduces `object$theta[[term]]` exactly when `varianceScale=FALSE`, where σ^2 is `object$covPar[[term]][1]`. With `varianceScale=TRUE`, the returned loadings crossproduct plus the diagonal of the returned specific variances directly reproduces that covariance.

Value

A list containing:

loadings a levels-by-factor matrix Λ.

specific a named vector of specific variances Ψ (identically zero for every level in a `rrm()` term).

sigma2 the fitted overall variance scale.

model `"fa"` or `"rr"`.

term the resolved term name.

See Also

`scores_mmes`, `fam`, `rrm`, `vsm`, `mmes`.

Examples

```
## Not run:
mix <- mmes(BLUEs ~ trial,
            random = ~ vsm(fam(trial, 2), ism(genotype)),
            rcov = ~ units, data = dt)
loadings_mmes(mix)

## End(Not run)
```

`mam` *Moving-average covariance structures of order one or two*

Description

Moving-average covariance structures of order one or two. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
mam(x, order = 1L, theta = NULL, fixed = NULL)
ma1m(x, theta = 0.15, fixed = FALSE)
ma2m(x, theta = c(0.15, 0.05), fixed = NULL)
```

Arguments

`x`	Ordered factor defining covariance levels.
`order`	Moving-average order; currently 1 or 2.
`theta`	Starting MA polynomial coefficients. Defaults to 0.15 for each order.
`fixed`	Logical vector with one value per MA coefficient.

Details

Using the conventional polynomial $e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2}$, the autocovariance at lag h is proportional to $\sum_{j=0}^{p-h} \theta_j \theta_{j+h}$ with $\theta_0 = 1$. The covariance is normalized to correlation scale and is zero beyond lag p. Invertibility of the MA polynomial is not required merely to define this covariance matrix. `ma1m` and `ma2m` are convenience wrappers.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`ar1m`, `toeplitzm`, `vsm`.

Examples

```
## Not run:
vsm(ma1m(time), ism(id))
vsm(ma2m(time), ism(id))

## End(Not run)
```

`maternm` *Matern spatial covariance structure*

Description

Matern spatial covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
maternm(x, range = NULL, nu = 0.5, fixed = c(FALSE, FALSE),
        distance = NULL, anisotropy = c("none", "geometric"),
        angle = pi/2, ratio = 1.5)
```

Arguments

`x`	Numeric coordinate vector or matrix/data frame whose rows give spatial coordinates.
`range`	Positive starting range parameter. If NULL, the median positive pairwise distance is used.
`nu`	Positive starting Matern smoothness parameter.
`fixed`	Logical of length one or two (range, nu), or four with geometric anisotropy (range, nu, angle, ratio); length one is recycled.

distance	Optional finite symmetric q by q distance matrix with zero diagonal, aligned to the unique spatial locations.
anisotropy	"geometric" rotates two-dimensional coordinates by angle and divides the second rotated axis by ratio before computing distances (ASReml anisotropic Matern).
angle	Starting anisotropy angle in (0, pi).
ratio	Starting positive anisotropy ratio.

Details

For distance d, the correlation is

$$K(d) = \frac{2^{1-\nu}}{\Gamma(\nu)} z^{\nu} K_{\nu}(z), \qquad z = \frac{\sqrt{2\nu}d}{\phi},$$

where ϕ is range. Both range and smoothness are positive and are optimized on log scales. The implementation uses the generic R evaluator and central factor-level numerical derivatives. Repeated coordinate rows are mapped to the same covariance level.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

metricm, sar, car, vsm.

Examples

```
## Not run:
vsm(maternm(cbind(xcoord,ycoord)), ism(genotype))
vsm(maternm(time), ism(id))

## End(Not run)
```

---

meffects_mmes *Marker effects recovered from a fitted relationship-based term*

Description

Back-transforms the BLUPs of a random term fitted with a genomic relationship matrix (GBLUP) into BLUPs of marker effects, optionally with standard errors and back-solved GWAS-type tests. This is the post-fit equivalent of ASReml's mef argument and does not require refitting the model.

Usage

```
meffects_mmes(object, term, M, method = c("A.mat", "custom"),
              min.MAF = 0, Z = NULL, scale = NULL, blend = 0, Gu = NULL,
              se = FALSE, chunk = 1000)
```

Arguments

object	A fitted mmes model.
term	Name (as in names(object$uList)) or position of the random term fitted with Gu.
M	Marker matrix (individuals by markers, coded -1/0/1 as for A.mat) with rownames matching the levels of the term. Use the same matrix (same rows) that was used to build the relationship matrix, because allele frequencies are computed from it. Rows that are not levels of the term are ignored.
method	"A.mat" reproduces exactly how A.mat builds G from M (imputation, min.MAF and monomorphic filters, centring by $2p$ and scaling by $2\sum p_j q_j$). "custom" uses Z and scale.
min.MAF	The min.MAF used in A.mat.
Z	For method="custom": transformed marker matrix with rownames such that $G = scale \cdot ZZ^\top$.
scale	For method="custom": the scalar s in $G = sZZ^\top$.
blend	λ if the fitted matrix was $(1-\lambda)G + \lambda I$. Small bending constants added only for invertibility can be left at 0.
Gu	Optional relationship matrix (or inverse, with attr(Gu,"inverse")=TRUE). By default the retained precision used in the fit is reused; it is rebuilt from the model inputs only when it is unavailable.
se	If TRUE, add standard errors, z statistics and p-values.
chunk	Positive integer number of markers processed per block when se=TRUE.

Details

If the fitted relationship matrix is $G = (1-\lambda)sZZ^\top + \lambda I$, the BLUP of the marker effects is

$$\hat{a} = (1-\lambda)s\, Z^\top G^{-1}\hat{u},$$

using the same G^{-1} as the fit, so $Z\hat{a} = \hat{u} - \lambda G^{-1}\hat{u}$ (exactly $\hat{u}$ when $\lambda = 0$). For structured terms such as vsm(dsm(env), ism(id), Gu=Gi) effects are returned per coordinate (environment, trait, ...).

With se=TRUE, following Gualdron-Duarte et al. (2014),

$$\mathrm{Var}(\hat{a}) = H^\top(\sigma_c^2 G - C^{uu})H, \qquad H = (1-\lambda)sG^{-1}Z,$$

where σ_c^2 is the variance of the coordinate and C^{uu} the prediction error covariance of the term; $z = \hat{a}/\mathrm{SE}$ gives back-solved GWAS tests. Only diagonal elements are formed: the C^{uu} part uses solves with the stored coefficient matrix in blocks of chunk markers, sharing one coefficient-matrix factorization across blocks and coordinates. The marker transform $G^{-1}Z$ is formed only for each block; effects-only calculations do not materialize it for all markers.

Rebuilding G^{-1}, if no retained precision is available, requires the data used in the original call; supply Gu otherwise. Markers removed by the filters are returned as NA.

Value

A data frame with columns `marker`, `coordinate` (structured terms only), `effect` and, if `se=TRUE`, `se`, `z` and `p.value`. Attributes `scale` and `markersKept`.

References

Gualdron-Duarte JL, Cantet RJC, Bates RO, Ernst CW, Raney NE, Steibel JP (2014). Rapid screening for phenotype-genotype associations by linear transformations of genomic evaluations. BMC Bioinformatics 15: 246.

See Also

`A.mat`, `mmes`, `vsm`

Examples

```
set.seed(7)
M <- matrix(sample(c(-1, 0, 1), 40 * 80, TRUE), 40)
rownames(M) <- paste0("g", 1:40); colnames(M) <- paste0("snp", 1:80)
G <- 0.95 * A.mat(M) + 0.05 * diag(40)
Gi <- solve(G); attr(Gi, "inverse") <- TRUE
d <- data.frame(id = factor(rep(rownames(M), each = 2), levels = rownames(M)))
d$y <- rnorm(80)
fit <- mmes(y ~ 1,
  random = ~ vsm(ism(id), Gu = Gi),
  data = d,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
head(meffects_mmes(fit,
  1,
  M,
  blend = 0.05,
  se = TRUE))
```

---

metricm *Distance-based (metric) spatial correlation structures*

Description

Exponential, power, gaussian, spherical and circular correlation structures computed from distances between locations, isotropic or anisotropic. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
metricm(x, model = c("exponential", "power", "gaussian", "spherical", "circular"),
        range = NULL, rho = NULL, anisotropy = c("none", "product", "geometric"),
         angle = pi/2, ratio = 1.5, metric = c("euclidean", "manhattan"),
         distance = NULL, fixed = NULL)
```

Arguments

x	Numeric coordinate vector (1-D) or matrix/data frame whose rows give coordinates. Repeated coordinates share one covariance level.
model	Correlation function, see Details.
range	Starting range ϕ (one per axis with product anisotropy). Defaults to the median positive distance.
rho	Starting correlation per unit distance for model="power", in (0,1). Defaults to e^{-1} at the median distance.
anisotropy	"none"; "product", one parameter per axis, $K = f_x(\|dx\|)f_y(\|dy\|)$; or "geometric", coordinates rotated by angle and the second rotated axis divided by ratio. Anisotropy requires 2-D coordinates.
angle	Starting anisotropy angle in (0, pi).
ratio	Starting positive anisotropy ratio.
metric	Distance metric for isotropic models; "manhattan" is only valid (positive definite) for the exponential and power models.
distance	Optional symmetric q by q distance matrix with zero diagonal, aligned to the unique locations (isotropic models only).
fixed	Logical vector, one value per parameter (in the order range/rho, then angle and ratio), recycled if of length one.

Details

With $h = d/\phi$:

model	$K(d)$	validity	ASReml
exponential	e^{-h}	any dimension	exp, iexp
power	ρ^d	any dimension	exp, iexp (same parameterization)
gaussian	e^{-h^2}	any dimension	gau, igau
spherical	$1 - \frac{3}{2}h + \frac{1}{2}h^3$ for $h < 1$, else 0	up to 3-D	isp
circular	$\frac{2}{\pi}(\arccos h - h\sqrt{1-h^2})$ for $h < 1$, else 0	up to 2-D	cir

Product anisotropy (exponential, power and gaussian) corresponds to ASReml aexp/agau; geometric anisotropy corresponds to ASReml's anisotropic Matern and is also available in maternm. The power model is the exponential model with $\phi = -1/\log\rho$, and the exponential model equals maternm(nu=0.5).

Ranges and ratios are optimized on log scales, correlations and angles through bounded logits, with analytic derivatives. The structure is dense over the q locations, so it is intended for up to a few thousand locations. On a complete row-by-column grid, product anisotropy is a Kronecker product of two 1-D structures; vsm(metricm(row, "power"), metricm(col, "power"), ism(units)) fits the same model while keeping the compact grid engine. The gaussian model is nearly singular for close locations; keep a residual (nugget) term. With geometric anisotropy the angle is not identifiable when the ratio is close to one, and the ratio can drift to extreme values when the data are isotropic.

Value

A list with the incidence matrix Z and a CovarianceFactor descriptor covFactor, for use inside vsm.

See Also

maternm, ar1m, sar, car, vsm.

Examples

```
set.seed(1)
co <- cbind(x = runif(30, 0, 10), y = runif(30, 0, 10))
d <- data.frame(x = rep(co[, 1], each = 2), y = rep(co[, 2], each = 2), one = factor(1))
d$yield <- rnorm(60)
fit <- mmes(yield ~ 1,
  random = ~ vsm(metricm(cbind(x, y), model = "power"), ism(one)),
  data = d,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
covparams_mmes(fit)
```

---

mmer **m***ixed* **m***odel* **e***quations for* **r** *records*

Description

THIS FUNCTION HAS BEEN DEPRECATD PLEASE EXPLORE mmes. The mmer function uses the Direct-Inversion Newton-Raphson or Average Information coded in C++ using the Armadillo library to optimize dense matrix operations common in genomic selection models. These algorithms are **intended to be used for problems of the type c > r (more coefficients to estimate than records in the dataset) and/or dense matrices**. For problems with sparse data, or problems of the type r > c (more records in the dataset than coefficients to estimate), the MME-based algorithm in the mmes function is faster and we recommend to shift to use that function.

Usage

```
mmer(fixed, random, rcov, data, weights, W, nIters=20, tolParConvLL = 1e-03,
     tolParInv = 1e-06, init=NULL, constraints=NULL,method="NR", getPEV=TRUE,
     naMethodX="exclude", naMethodY="exclude",returnParam=FALSE,
   dateWarning=TRUE,date.warning=TRUE,verbose=TRUE, reshapeOutput=TRUE, stepWeight=NULL,
     emWeight=NULL, contrasts=NULL)
```

Arguments

fixed	A formula specifying the **response variable(s) and fixed effects**, e.g.: *response ~ covariate* for univariate models *cbind(response.i,response.j) ~ covariate* for multivariate models The fcm function can be used to constrain fixed effects in multi-response models.

`random` A formula specifying the name of the **random effects**, e.g. *random*= ~ *genotype* + *year*.

Useful functions can be used to fit heterogeneous variances and other special models (*see 'Special Functions' in the Details section for more information*):

`vsr(...,Gu,Gti,Gtc)` is the main function to specify variance models and special structures for random effects. On the ... argument you provide the unknown variance-covariance structures (e.g., usr,dsr,atr,csr) and the random effect where such covariance structure will be used (the random effect of interest). Gu is used to provide known covariance matrices among the levels of the random effect, Gti initial values and Gtc for constraints. Auxiliar functions for building the variance models are:

** `dsr(x)`, `usr(x)`, `csr(x)` and `atr(x,levs)` can be used to specify unknown diagonal, unstructured and customized unstructured and diagonal covariance structures to be estimated by REML.

** `unsm(x)`, `fixm(x)` and `diag(x)` can be used to build easily matrices to specify constraints in the Gtc argument of the `vsr()` function.

** `overlay()`, `spl2Da()`, `spl2Db()`, and `leg()` functions can be used to specify overlayed of design matrices of random effects, two dimensional spline and random regression models within the `vsr()` function.

`gvsr(...,Gu,Guc,Gti,Gtc)` is an alternative function to specify general variance structures between different random effects. An special case in the indirect genetic effect models. Is similar to the vsr function but in the ... argument the different random effects are provided.

`rcov` A formula specifying the name of the **error term**, e.g., *rcov*= ~ *units*.

Special heterogeneous and special variance models and constraints for the residual part are the same used on the random term but the name of the random effect is always "units" which can be thought as a column with as many levels as rows in the data, e.g., *rcov=~vsr(dsr(covariate),units)*

`data` A data frame containing the variables specified in the formulas for response, fixed, and random effects.

`weights` Name of the covariate for weights. To be used for the product R = Wsi*R*Wsi, where * is the matrix product, Wsi is the square root of the inverse of W and R is the residual matrix.

`W` Alternatively, instead of providing a vector of weights the user can specify an entire W matrix (e.g., when covariances exist). To be used first to produce Wis = solve(chol(W)), and then calculate R = Wsi*R*Wsi.t(), where * is the matrix product, and R is the residual matrix. Only one of the arguments weights or W should be used. If both are indicated W will be given the preference.

`nIters` Maximum number of iterations allowed.

`tolParConvLL` Convergence criteria for the change in log-likelihood.

`tolParInv` Tolerance parameter for matrix inverse used when singularities are encountered in the estimation procedure.

`init` Initial values for the variance components. By default this is NULL and initial values for the variance components are provided by the algorithm, but in case the user want to provide initial values for ALL var-cov components this argument is

functional. It has to be provided as a list, where each list element corresponds to one random effect (1x1 matrix) and if multitrait model is pursued each element of the list is a matrix of variance covariance components among traits for such random effect. Initial values can also be provided in the Gti argument of the vsr function. Is highly encouraged to use the Gti and Gtc arguments of the vsr function instead of this argument, but these argument can be used to provide all initial values at once

`constraints` When initial values are provided these have to be accompanied by their constraints. See the vsr function for more details on the constraints. Is highly encouraged to use the Gti and Gtc arguments of the vsr function instead of this argument but these argument can be used to provide all constraints at once.

`method` This refers to the method or algorithm to be used for estimating variance components. Direct-inversion Newton-Raphson **NR** and Average Information **AI** (Tunnicliffe 1989; Gilmour et al. 1995; Lee et al. 2015).

`getPEV` A TRUE/FALSE value indicating if the program should return the predicted error variance and variance for random effects. This option is provided since this can take a long time for certain models where p is > n by a big extent.

`naMethodX` One of the two possible values; "include" or "exclude". If "include" is selected then the function will impute the X matrices for fixed effects with the median value. If "exclude" is selected it will get rid of all rows with missing values for the X (fixed) covariates. The default is "exclude". The "include" option should be used carefully.

`naMethodY` One of the three possible values; "include", "include2" or "exclude" (default) to treat the observations in response variable to be used in the estimation of variance components. The first option "include" will impute the response variables for all rows with the median value, whereas "include2" imputes the responses only for rows where there is observation(s) for at least one of the responses (only available in the multi-response models). If "exclude" is selected (default) it will get rid of rows in response(s) where missing values are present for at least one of the responses.

`returnParam` A TRUE/FALSE value to indicate if the program should return the parameters to be used for fitting the model instead of fitting the model.

`dateWarning` A TRUE/FALSE value to indicate if the program should warn you when is time to update the sommer package.

`date.warning` A TRUE/FALSE value to indicate if the program should warn you when is time to update the sommer package. This argument will be removed soon, just left for backcompatibility.

`verbose` A TRUE/FALSE value to indicate if the program should return the progress of the iterative algorithm.

`reshapeOutput` A TRUE/FALSE value to indicate if the output should be reshaped to be easier to interpret for the user, some information is missing from the multivariate models for an easy interpretation.

`stepWeight` A vector of values (of length equal to the number of iterations) indicating the weight used to multiply the update (delta) for variance components at each iteration. If NULL the 1st iteration will be multiplied by 0.5, the 2nd by 0.7, and

the rest by 0.9. This argument can help to avoid that variance components go outside the parameter space in the initial iterations which doesn't happen very often with the NR method but it can be detected by looking at the behavior of the likelihood. In that case you may want to give a smaller weight to the initial 8-10 iterations.

`emWeight` A vector of values (of length equal to the number of iterations) indicating with values between 0 and 1 the weight assigned to the EM information matrix. And the values 1 - emWeight will be applied to the NR/AI information matrix to produce a joint information matrix.

`contrasts` an optional list. See the contrasts.arg of model.matrix.default.

Details

The use of this function requires a good understanding of mixed models. Please review the 'sommer.quick.start' vignette and pay attention to details like format of your random and fixed variables (e.g. character and factor variables have different properties when returning BLUEs or BLUPs, please see the 'sommer.changes.and.faqs' vignette).

For tutorials on how to perform different analysis with sommer please look at the vignettes by typing in the terminal:

vignette("v1.sommer.quick.start")

vignette("v2.sommer.changes.and.faqs")

vignette("v3.sommer.qg")

vignette("v4.sommer.gxe")

Citation

Type *citation("sommer")* to know how to cite the sommer package in your publications.

Special variance structures

`vsr(atr(x,levels),y)`

can be used to specify heterogeneous variance for the "y" covariate at specific levels of the covariate "x", e.g., *random=~vsr(at(Location,c("A","B")),ID)* fits a variance component for ID at levels A and B of the covariate Location.

`vsr(dsr(x),y)`

can be used to specify a diagonal covariance structure for the "y" covariate for all levels of the covariate "x", e.g., *random=~vsr(dsr(Location),ID)* fits a variance component for ID at all levels of the covariate Location.

`vsr(usr(x),y)`

can be used to specify an unstructured covariance structure for the "y" covariate for all levels of the covariate "x", e.g., *random=~vsr(usr(Location),ID)* fits variance and covariance components for ID at all levels of the covariate Location.

`vsr(overlay(...,rlist=NULL,prefix=NULL))`

can be used to specify overlay of design matrices between consecutive random effects specified, e.g., *random=~vsr(overlay(male,female))* overlays (overlaps) the incidence matrices for the male and female random effects to obtain a single variance component for both effects. The 'rlist' argument

is a list with each element being a numeric value that multiplies the incidence matrix to be overlayed. See `overlay` for details.Can be combined with vsr().

`vsr(leg(x,n),y)`

can be used to fit a random regression model using a numerical variable `x` that marks the trayectory for the random effect `y`. The leg function can be combined with the special functions `dsr`, `usr` `at` and `csr`. For example *random=~vsr(leg(x,1),y)* or *random=~vsr(usr(leg(x,1)),y)*.

`vsr(x,Gtc=fcm(v))`

can be used to constrain fixed effects in the multi-response mixed models. This is a vector that specifies if the fixed effect is to be estimated for such trait. For example *fixed=cbind(response.i, response.j)~vsr(Rowf, Gtc=fcm(c(1,0)))* means that the fixed effect Rowf should only be estimated for the first response and the second should only have the intercept.

`gvsr(x,y)`

can be used to fit variance and covariance parameters between two or more random effects. For example, indirect genetic effect models.

`spl2Da(x.coord, y.coord, at.var, at.levels))`

can be used to fit a 2-dimensional spline (e.g., spatial modeling) using coordinates `x.coord` and `y.coord` (in numeric class) assuming a single variance component. The 2D spline can be fitted at specific levels using the `at.var` and `at.levels` arguments. For example *random=~spl2Da(x.coord=Row,y.coord=Range,at.*

`spl2Db(x.coord, y.coord, at.var, at.levels))`

can be used to fit a 2-dimensional spline (e.g., spatial modeling) using coordinates `x.coord` and `y.coord` (in numeric class) assuming multiple variance components. The 2D spline can be fitted at specific levels using the `at.var` and `at.levels` arguments. For example *random=~spl2Db(x.coord=Row,y.coord=Range,at.*

S3 methods

S3 methods are available for some parameter extraction such as `fitted.mmer`, `residuals.mmer`, `summary.mmer`, `randef`, `coef.mmer`, `anova.mmer`, `plot.mmer`, and `predict.mmer` to obtain adjusted means. In addition, the `vpredict` function (replacement of the pin function) can be used to estimate standard errors for linear combinations of variance components (e.g., ratios like h2).

Additional Functions

Additional functions for genetic analysis have been included such as relationship matrix building (`A.mat`, `D.mat`, `E.mat`, `H.mat`), build a genotypic hybrid marker matrix (`build.HMM`), plot of genetic maps (`map.plot`), and manhattan plots (`manhattan`). If you need to build a pedigree-based relationship matrix use the `getA` function from the pedigreemm package.

Bug report and contact

If you have any technical questions or suggestions please post it in https://stackoverflow.com or https://stats.stackexchange.com

If you have any bug report please go to https://github.com/covaruber/sommer or send me an email to address it asap, just make sure you have read the vignettes carefully before sending your question.

Example Datasets

The package has been equiped with several datasets to learn how to use the sommer package:

* `DT_halfdiallel`, `DT_fulldiallel` and `DT_mohring` datasets have examples to fit half and full diallel designs.

* `DT_h2` to calculate heritability

* DT_cornhybrids and DT_technow datasets to perform genomic prediction in hybrid single crosses

* DT_wheat dataset to do genomic prediction in single crosses in species displaying only additive effects.

* DT_cpdata dataset to fit genomic prediction models within a biparental population coming from 2 highly heterozygous parents including additive, dominance and epistatic effects.

* DT_polyploid to fit genomic prediction and GWAS analysis in polyploids.

* DT_gryphon data contains an example of an animal model including pedigree information.

* DT_btdata dataset contains an animal (birds) model.

* DT_legendre simulated dataset for random regression model.

* DT_sleepstudy dataset to know how to translate lme4 models to sommer models.

* DT_ige dataset to show how to fit indirect genetic effect models.

Models Enabled

For details about the models enabled and more information about the covariance structures please check the help page of the package (sommer).

Value

If all parameters are correctly indicated the program will return a list with the following information:

Vi	the inverse of the phenotypic variance matrix V^- = (ZGZ+R)^-1
P	the projection matrix Vi - [Vi*(X*Vi*X)^-*Vi]
sigma	a list with the values of the variance-covariance components with one list element for each random effect.
sigma_scaled	a list with the values of the scaled variance-covariance components with one list element for each random effect.
sigmaSE	Hessian matrix containing the variance-covariance for the variance components. SE's can be obtained taking the square root of the diagonal values of the Hessian.
Beta	a data frame for trait BLUEs (fixed effects).
VarBeta	a variance-covariance matrix for trait BLUEs
U	a list (one element for each random effect) with a data frame for trait BLUPs.
VarU	a list (one element for each random effect) with the variance-covariance matrix for trait BLUPs.
PevU	a list (one element for each random effect) with the predicted error variance matrix for trait BLUPs.
fitted	Fitted values y.hat=XB
residuals	Residual values e = Y - XB
AIC	Akaike information criterion
BIC	Bayesian information criterion
convergence	a TRUE/FALSE statement indicating if the model converged.
monitor	The values of log-likelihood and variance-covariance components across iterations during the REML estimation.

percChange	The percent change of variance components across iterations. There should be one column less than the number of iterations. Calculated as percChange = ((x_i/x_i-1) - 1) * 100 where i is the ith iteration.
dL	The vector of first derivatives of the likelihood with respect to the ith variance-covariance component.
dL2	The matrix of second derivatives of the likelihood with respect to the i.j th variance-covariance component.
method	The method for extimation of variance components specified by the user.
call	Formula for fixed, random and rcov used.
constraints	contraints used in the mixed models for the random effects.
constraintsF	contraints used in the mixed models for the fixed effects.
data	The dataset used in the model after removing missing records for the response variable.
dataOriginal	The original dataset used in the model.
terms	The name of terms for responses, fixed, random and residual effects in the model.
termsN	The number of effects associated to fixed, random and residual effects in the model.
sigmaVector	a vectorized version of the sigma element (variance-covariance components) to match easily the standard errors of the var-cov components stored in the element sigmaSE.
reshapeOutput	The value provided to the mmer function for the argument with the same name.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G. Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 2016, 11(6): doi:10.1371/journal.pone.0156744

Covarrubias-Pazaran G. 2018. Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639

Bernardo Rex. 2010. Breeding for quantitative traits in plants. Second edition. Stemma Press. 390 pp.

Gilmour et al. 1995. Average Information REML: An efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51(4):1440-1450.

Kang et al. 2008. Efficient control of population structure in model organism association mapping. Genetics 178:1709-1723.

Lee, D.-J., Durban, M., and Eilers, P.H.C. (2013). Efficient two-dimensional smoothing with P-spline ANOVA mixed models and nested bases. Computational Statistics and Data Analysis, 61, 22 - 37.

Lee et al. 2015. MTG2: An efficient algorithm for multivariate linear mixed model analysis based on genomic information. Cold Spring Harbor. doi: https://doi.org/10.1101/027201.

Maier et al. 2015. Joint analysis of psychiatric disorders increases accuracy of risk prediction for schizophrenia, bipolar disorder, and major depressive disorder. Am J Hum Genet; 96(2):283-294.

Rodriguez-Alvarez, Maria Xose, et al. Correcting for spatial heterogeneity in plant breeding experiments with P-splines. Spatial Statistics 23 (2018): 52-71.

Searle. 1993. Applying the EM algorithm to calculating ML and REML estimates of variance components. Paper invited for the 1993 American Statistical Association Meeting, San Francisco.

Yu et al. 2006. A unified mixed-model method for association mapping that accounts for multiple levels of relatedness. Genetics 38:203-208.

Tunnicliffe W. 1989. On the use of marginal likelihood in time series model estimation. JRSS 51(1):15-27.

Zhang et al. 2010. Mixed linear model approach adapted for genome-wide association studies. Nat. Genet. 42:355-360.

Examples

```
####============================================####
#### For CRAN time limitations most lines in the
#### examples are silenced with one '#' mark,
#### remove them and run the examples
####============================================####

####============================================####
#### EXAMPLES
#### Different models with sommer
####============================================####

data(DT_example, package="enhancer")
DT <- DT_example
head(DT)

####============================================####
#### Univariate homogeneous variance models  ####
####============================================####

## Compound simmetry (CS) model
ans1 <- mmer(Yield~Env,
             random= ~ Name + Env:Name,
             rcov= ~ units,
             data=DT)
summary(ans1)

####============================================####
#### Univariate heterogeneous variance models  ####
####============================================####

## Compound simmetry (CS) + Diagonal (DIAG) model
ans2 <- mmer(Yield~Env,
             random= ~Name + vsr(dsr(Env),Name),
             rcov= ~ vsr(dsr(Env),units),
             data=DT)
```

```
summary(ans2)




####============================================####
####  Univariate unstructured variance models  ####
####============================================####

ans3 <- mmer(Yield~Env,
             random=~ vsr(usr(Env),Name),
             rcov=~vsr(dsr(Env),units),
             data=DT)
summary(ans3)

####============================================####
#### Multivariate homogeneous variance models ####
####============================================####

## Multivariate Compound simmetry (CS) model
DT$EnvName <- paste(DT$Env,DT$Name)
ans4 <- mmer(cbind(Yield, Weight) ~ Env,
              random= ~ vsr(Name, Gtc = unsm(2)) + vsr(EnvName,Gtc = unsm(2)),
              rcov= ~ vsr(units, Gtc = unsm(2)),
              data=DT)
summary(ans4)

####================================================####
#### Multivariate heterogeneous variance models  ####
####================================================####

## Multivariate Compound simmetry (CS) + Diagonal (DIAG) model
ans5 <- mmer(cbind(Yield, Weight) ~ Env,
              random= ~ vsr(Name, Gtc = unsm(2)) + vsr(dsr(Env),Name, Gtc = unsm(2)),
              rcov= ~ vsr(dsr(Env),units, Gtc = unsm(2)),
              data=DT)
summary(ans5)

####============================================####
#### Multivariate unstructured variance models ####
####============================================####

ans6 <- mmer(cbind(Yield, Weight) ~ Env,
              random= ~ vsr(usr(Env),Name, Gtc = unsm(2)),
              rcov= ~ vsr(dsr(Env),units, Gtc = unsm(2)),
              data=DT)
summary(ans6)

####========================================####
####========================================####
#### EXAMPLE SET 2
#### 2 variance components
```

```
#### one random effect with variance covariance structure
####=========================================####
####=========================================####

data("DT_cpdata", package="enhancer")
DT <- DT_cpdata
GT <- GT_cpdata
MP <- MP_cpdata
head(DT)
GT[1:4,1:4]
#### create the variance-covariance matrix
A <- A.mat(GT)
#### look at the data and fit the model
mix1 <- mmer(Yield~1,
             random=~vsr(id, Gu=A) + Rowf,
             rcov=~units,
             data=DT)
summary(mix1)$varcomp

#### multi trait example
mix2 <- mmer(cbind(Yield,color)~1,
              random = ~ vsr(id, Gu=A, Gtc = unsm(2)) + # unstructured at trait level
                           vsr(Rowf, Gtc=diag(2)) + # diagonal structure at trait level
                            vsr(Colf, Gtc=diag(2)), # diagonal structure at trait level
              rcov = ~ vsr(units, Gtc = unsm(2)), # unstructured at trait level
              data=DT)
summary(mix2)
```

`mmes` **m***ixed* **m***odel* **e***quations* **s***olver*

Description

Fits a generalized linear mixed model using model formulas, covariance structures and observation data. Special covariance structures can be specified with `vsm()`. Numerical fitting choices are supplied through the strictly validated `control` object; see `mmes_control` and the `sommer.mmes.options` vignette.

Usage

```
mmes(fixed, random, rcov, data, W, weights=NULL,
     naMethodX="exclude", naMethodY="exclude",
     naMethodRandom="exclude", naMethodR="exclude",
     contrasts=NULL, family=stats::gaussian(), vcc=NULL,
     covPar=NULL, control=mmes_control(), .internal=NULL, ...)
```

Arguments

`fixed`

A formula specifying the **response variable(s) and fixed effects**, i.e:

response ~ covariate

By default, the fixed-effect design must have full column rank after observation filtering. Before fitting, `mmes()` reports all-zero columns (for example, unused levels or unobserved interaction combinations) or linearly dependent columns (for example, a parent factor together with unrestricted nested cell effects). Aliased coefficients are not automatically dropped or constrained to zero unless explicitly requested with `control=mmes_control(representation=list(fixedRank="drop"))`. This option selects a full-rank subset using shared native sparse QR for the Henderson and direct engines. The original coefficient layout is preserved, with omitted coefficients returned as zero identification constraints and flagged in `summary(fit)$betas`; these zeros are not estimates of independently identifiable effects. Details, mappings and supported inference workflows are documented in `mmes_control`.

Apply `droplevels()` to the retained data to remove unused factor levels. For a sparse interaction, an observed-cell factor created with `interaction(..., drop=TRUE)` and a formula such as `response ~ 0 + cell` can represent the same fitted cell means without redundant columns. Adding the parent factor again reintroduces aliasing. Dropping unused levels alone does not remove all unobserved interaction combinations. The diagnostic setup mode (`control=mmes_control(output=list(se`) still returns the design without requiring full column rank, allowing it to be inspected. With `fixedRank="drop"`, setup instead returns the reduced design and its `fixedRank` mapping.

`random`

A formula specifying the **random effects**, e.g., *random = ~ genotype + year*. A simple random term that is not wrapped in `vsm` is treated as an identity-covariance random effect.

The `vsm` function is the main interface for specifying structured covariance models for random effects. Its current covariance model is

$$G = \sigma^2(K_1 \otimes K_2 \otimes \cdots \otimes K_m) \otimes A,$$

where σ^2 is the single variance scale owned by `vsm()`, $K_1, \ldots, K_m$ are covariance-shaping factors supplied by covariance constructors, and A is the known covariance relationship for the final random effect. In the Henderson implementation `Gu` supplies the **inverse** of A; it must have row and column names matching the levels of the final random effect and `attr(Gu,"inverse")=TRUE`.

Within `vsm(...)`, the **last** constructor supplies the incidence matrix for the main random effect, while all preceding constructors define covariance-shaping factors. The number of covariance factors is not limited. For example,

`random = ~ vsm(dsm(Location), ism(Name))`

fits heterogeneous variances across `Location` for the random effect `Name`, while

`random = ~ vsm(dsm(Environment), ar1m(Row), ar1m(Column), ism(Name), Gu=Ainv)`

specifies an arbitrary Kronecker product of covariance structures before the relationship matrix for `Name`.

Covariance constructors currently available include:

ism for identity covariance;

dsm and atm for diagonal and selected-level diagonal covariance structures;

usm for an unstructured covariance matrix;

csm and corgm for correlation structures;

ar1m, ar2m, and ar3m for autoregressive covariance structures;

mam (including ma1m and ma2m) for moving-average covariance structures;

toeplitzm for a general positive-definite Toeplitz correlation structure;

fam for factor-analytic covariance;

rrm for a reduced-rank random covariance with specific variances fixed at zero; use fam(..., variance="homogeneous") for an estimated common specific variance;

antem for antedependence covariance;

maternm for Matern spatial covariance;

metricm for exponential, power, gaussian, spherical and circular distance-based correlations (isotropic or anisotropic);

sar and car for simultaneous and conditional autoregressive spatial covariance models; and

ownm for a fixed or user-defined covariance function.

Design-matrix utilities such as overlay, spl2Dc, leg, and redmm can still be combined with vsm() when appropriate.

See vsm and the individual covariance-constructor help pages for details on parameterizations and examples.

A single relationship term may request Lee–van der Werf rotation with vsm(..., Gu=Gu, rotation=TRUE). With control$formulation="henderson", random coefficients are represented in the eigenbasis and the diagonal relationship precision is used in the mixed model equations. With control$formulation="direct", the equivalent transformed observation-covariance parameterization is used. Public random effects, fitted values, and residuals are returned in the original basis. The current implementation requires a complete balanced Gaussian layout, one rotated relationship term, and control$output$inverse=0 during fitting. Residual structures are allowed when they are rotation invariant: residuals may be structured across rotation blocks (e.g., rcov=~vsm(usm(trait), ism(units)) or rcov=~vsm(dsm(Env), ism(units))), but must be independent and identically structured across the levels of the rotated relationship term. Residual structures that correlate observations of different levels (e.g., ar1m(), maternm(), sar(), or car() over plots), residual variances that differ among levels within a rotation block, and W matrices that are not diagonal with a constant weight within each rotation block are rejected, as are non-Gaussian PQL fits.

rcov

A formula specifying the **residual covariance structure**. The default is *rcov* = ~ *units*, which fits a homogeneous residual variance.

Structured residual covariance is specified with the same vsm and Covariance-Factor constructors used for random effects. The residual term should be expressed as a single vsm() term; arbitrary covariance complexity is represented

by placing multiple covariance factors inside that term rather than by summing multiple residual terms.

The final term is normally `ism(units)`, where `units` is the observation-level incidence generated internally by `mmes`. For example,

`rcov = ~ vsm(dsm(Environment), ism(units))`

fits heterogeneous residual variances across environments, while

`rcov = ~ vsm(ar1m(Row), ar1m(Column), ism(units))`

fits a separable row-by-column AR1 residual covariance, with one correlation per direction shared by every other Kronecker factor. For section-specific variances and correlations, use

`rcov = ~ dsumm(vsm(ar1m(Row), ar1m(Column), ism(units)), by = Trial)`

(see `dsumm`). More generally, structures such as `usm`, `csm`, `toeplitzm`, `maternm`, `sar`, `car`, and `ownm` can be used when their incidence layout is appropriate for the residual model.

Residual covariance factors must identify exactly one covariance-product coordinate for each observation. `mmes()` constructs the residual block and local-coordinate indexing internally, so the data do **not** need to be manually sorted by the variables defining the residual structure.

Residual pairing key (multi-trait models). For a long-format multi-trait model the residual must know which records belong to the same experimental unit. With `ism(units)` records are paired implicitly by their order of appearance within each trait; a warning is issued when repeated residual coordinates must be paired by row order. An explicit key is given by replacing `units` with a column identifying the unit, e.g.

`rcov = ~ vsm(usm(trait), ism(record))`

where `record` is the column created by `stackTraits`. Records sharing a key value form one residual block (missing traits are simply absent from it); a trait may appear only once per key value.

As with random effects, `vsm()` owns one residual variance scale σ^2; the preceding covariance constructors provide dimensionless covariance shapes that are combined through an arbitrary Kronecker product.

See `vsm` and the individual covariance-constructor help pages for details.

`data` Optional data frame containing variables used by the model. Model expressions are evaluated using normal R scoping rules: variables are first sought in `data` and may also be obtained from the environments associated with the model formulas or from the calling environment. Consequently, auxiliary objects such as relationship matrices, coordinate vectors, or grouping variables do not have to be copied into `data`. If `data` is omitted, variables are evaluated from the formula/calling environment.

Observation-level variables used by the response, fixed effects, random effects, and residual covariance specification must nevertheless have compatible lengths. Internally, `mmes()` constructs a common observation mask and applies it consistently to all observation-level model components.

`W` Weights matrix (e.g., when covariance among plots exist). Internally W is squared and inverted as Wsi = solve(chol(W)), then the residual matrix is calculated as R = Wsi*O*Wsi.t(), where * is the matrix product, and O is the original

residual matrix. Weights can be combined with any residual structure, including Kronecker (multi-trait) residuals.

`weights` Optional one-sided formula describing independent row blocks of the supplied `W` matrix, for example `weights = ~trial`. Rows with different formula groups must have zero cross-group entries in `W`. This is structural metadata only: it does not generate numeric weights. Declared blocks are processed independently by the Henderson weighting path; without this formula, `W` is handled without a known grouping.

`control` A `sommer_mmes_control` object created by `mmes_control`. It owns numerical task, equation formulation, representation, solver, determinant, score, curvature, covariance update, proposal, convergence, output, execution and diagnostic settings. Its default is `mmes_control()`. Removed numerical arguments are not accepted by `mmes()`; configure those choices in this object. See the control help and vignette for defaults, mathematical ownership, eligibility and migration.

`naMethodX` Missing-data policy for variables entering the fixed-effect design matrix. The default, `"exclude"`, marks observations with missing fixed-effect information for removal from the common observation set. `"include"` retains the historical sommer behavior for supported cases by imputing missing fixed-effect covariates. This option should be used carefully, because imputation changes the fitted design matrix.

`naMethodY` Missing-data policy for the response variable(s). The default, `"exclude"`, removes observations with missing response information from the common observation set. The historical `"include"` and `"include2"` behaviors are retained where supported: `"include"` imputes missing responses, whereas `"include2"` is intended for multivariate-response situations in which an observation may be retained when at least one response is observed.

`naMethodRandom` Missing-data policy for observation-level variables used to construct random-effect terms. The default is `"exclude"`. Under this policy, an observation for which a required random-effect classification or covariate cannot be evaluated is excluded from the common observation set. Random terms are evaluated before the final observation subset is applied, so the same inclusion mask is used consistently for the response, fixed-effect design, and all random-effect design matrices.

`naMethodR` Missing-data policy for observation-level variables used to define the residual covariance structure. The default is `"exclude"`. Observations lacking information required to assign their residual covariance-product coordinate are excluded from the common observation set. The residual block and local-coordinate layout are then constructed on the retained observations.

`contrasts` an optional list. See the contrasts.arg of model.matrix.default.

`vcc` Optional data frame of equality/scaling constraints between covariance parameters (columns `parameter`, `group` and optionally `scale`), identified through the `vcParams` table returned by `control$output$setup=TRUE` and stored in the fit. See `vcc`.

`family` A `stats::family()` object. The default Gaussian identity family uses the ordinary linear mixed-model implementation. Other families are fitted by penalized quasi-likelihood (PQL): each outer iteration forms an IRLS working

response and weights, then fits the resulting weighted Gaussian mixed model with the Henderson solver. The initial implementation supports one numeric response. Any family with a valid link can be used, e.g. `binomial()` (logit, probit, cloglog), `poisson()`, `Gamma()`, `inverse.gaussian()`, `gaussian(link="log")`, `quasipoisson()` and `MASS::negative.binomial(theta)`. Binomial counts out of a known number of trials are supplied as a single proportion response built with `binm`, whose trials are used as prior weights. Binomial, Poisson and negative binomial working residual dispersions are fixed to one. PQL uses outer-loop controls from `control$task$pql`; compatibility with update and solver requests is model-dependent. For other families the reported `dispersion` is the Pearson estimate. The negative binomial θ is estimated (conditional on the random effects) with `control$task$pql=list(estimateTheta=TRUE)` and returned in `theta.nb` and `theta.nb.se`. Because the working-model log-likelihood is not a GLMM likelihood, `llik`, `AIC` and `BIC` are `NA` for PQL fits and `anova()` likelihood ratio tests are refused; use `deviance` and Wald tests instead. Formula offsets are included on the link scale. A supplied symmetric positive definite `W` is combined with the IRLS precision after observation filtering, and may be sparse and non-diagonal.

Long-format multi-trait data with a different family per trait are fitted by supplying `family = familym("trait", Disease = binomial(), Yield = gaussian())`. See `familym` for the required residual structure; in that case `dispersion` is a named vector with one value per trait.

`covPar` — Known covariance parameters for the solve task selected by `control$task$mode="solve"`. Accepted forms are a compatible fitted `mmes` object, a named list of term parameters, or a parameter data frame. When `NULL`, covariance values must be fixed in the model.

`.internal` — Reserved for package-internal recursive fitting state. Users should not set this argument.

`...` — Deprecated legacy arguments. The supported arguments are `getPEV`, `tolParInv`, and `verbose`; they map respectively to `control$output$inverse`, `control$execution$inverseTol`, and `control$diagnostics$verbose`. These arguments override matching settings in `control` and issue a deprecation warning.

Details

Structurally unestimable REML components

For Gaussian identity-link REML fits, `mmes()` checks whether an independently scaled random component contributes anything outside the fixed-effect design space, using the observations retained by the common missing-data filter. A component whose entire random design lies in that space is invisible to the restricted likelihood. Such components are removed before optimization, with a warning, rather than assigned an arbitrary variance by a regularized information-matrix solve.

The conservative check handles a whole random term with a free variance scale and fixed covariance shape, or individual levels of a single `dsm()` or `atm()` shaping factor when its scale and all variance ratios are free. For example, with fixed locale means, a locale-specific independent block variance is omitted when that locale has only one observed block. A shared block variance is retained if other locales provide information about it. Correlated, multiply shaped, constrained or partly fixed

structures are not automatically reduced at individual levels. The check does not diagnose general aliasing between covariance terms or weak identification near a variance boundary.

Automatic omission is not applied to ML, PQL, known-covariance solve tasks, `vcc` constraints, observation rotation or factor-score augmentation. For supported fits, omitted components retain their original positions in random-effect and covariance outputs, with zero covariance and zero random effects. These zeros denote an omission convention, not variances estimated to be zero. Their native-scale standard errors and Z-ratios are `NA`; `summary()` and `covparams_mmes()` label them `"unestimable (omitted)"`. Only retained free parameters contribute to optimization and information criteria. Omission preserves the restricted likelihood information about retained parameters; it does not establish zero between-block variation in the population. Inference on fixed means and predictions is conditional on the omission convention.

Fixed factors with many levels

In Henderson fits the fixed-effect design `X` is stored as a sparse matrix. The coefficient matrix `C` of the mixed model equations is factorized with sparse methods at every REML iteration, whatever the `control$output$inverse` setting. The fill-reducing ordering of the sparse factorization eliminates the fixed-effect equations within the same factor as the random ones, which is equivalent to absorbing them. A fixed factor with thousands of levels, such as contemporary groups, does not create a dense fixed-effect block during estimation. Variance components, BLUEs and BLUPs are identical to those of an explicit absorption of the fixed effects.

`control$output$inverse` only controls work done after convergence:

`0` **(default)** No inverse of `C` is formed. Fixed-effect standard errors in `summary()` are `NA`.

`1` Elements of the inverse are computed only within the sparsity pattern of the factor (Takahashi selected inverse), giving random-effect PEVs. The fixed part stays sparse. Fixed-effect standard errors in `summary()` remain `NA`.

`2` The complete inverse of `C`, including the fixed block, is formed as a dense matrix. This needs about $8(p+q)^2$ bytes for p fixed and q random effects (around 1.4 GB for 13,000 effects) and is the mode that reports fixed-effect standard errors in `summary()`.

With many fixed levels, keep `inverse=0` or `1`. When standard errors of particular fixed-effect combinations are needed, use `predict.mmes`, which computes exact standard errors from the sparse `C` without forming `Ci`. `postPEV` can add PEVs or the full inverse after fitting.

When even the sparse REML fit is too slow, a practical alternative is to estimate the variance components with the large factor fitted as random, optionally on a well-connected subset of the data. Then refit with the factor as fixed using `control=mmes_control(task=list(mode="solve"))` and `covPar` to obtain BLUEs and BLUPs with a single factorization. Simply omitting the factor during REML is not equivalent: its variation is then attributed to the residual and to confounded random terms.

Numerical controls

The opt-in `control$representation$equations="conditional"` path fits Gaussian FA models using sparse latent equations with exact REML AI updates by default, or conditional-moment EM through `control$update$method="em"`. The earlier `control$update$method="conditional-fa"` selector retains its EM behavior. Both optimizers support incomplete and unbalanced observations. It accepts one FA relationship term and diagonal or FA residuals. Matching `fam(shared="group")` tags request common trait loadings across genetic and residual sources. The default AI path is unchanged. See `mmes_control` and the conditional FA section of `vignette("sommer.mmes.options")`

for eligibility and output restrictions. Conditional fits retain factorized covariance descriptors, not dense `theta` matrices, public `C`, coefficient uncertainty, or covariance-parameter uncertainty. Use `covmatrix_mmes(..., se=FALSE)` to explicitly reconstruct a bounded-size covariance.

`mmes()` keeps model specification separate from numerical policy. The `control` argument defaults to `mmes_control()`, whose modules have strict schemas: unknown field names are errors. The default configuration retains the former standard bundle: Henderson equations, REML, exact AI curvature, EM recovery, no extrapolation, 30 inner iterations, full covariance information, coefficient-matrix retention, and inherited solver/resource defaults. The `mmes_control` help gives every field, default, module owner and validation rule.

Ownership and eligibility

Task `control$task` selects fitting or known-covariance solving, REML or ML, and outer PQL controls.

Formulation `control$formulation` selects Henderson or direct observation-space equations.

Representation and solve `control$representation` describes equation/covariance representation; `control$solve` owns linear solver, preconditioning, reuse and elimination requests.

Evaluation `control$logdet` owns determinant methods; `control$score` owns score traces and covariance checks; `control$curvature` owns exact or hybrid AI and reduced-direction work.

Optimization `control$update` owns EM/PXEM recovery, blending and common-scale profiling. `control$proposal` owns step scaling and Aitken extrapolation. Aitken and hybrid AI are mutually exclusive.

Convergence and outputs `control$convergence` owns inner iteration, tolerances and stochastic accuracy; `control$output` owns coefficient/covariance uncertainty and setup output.

Execution and diagnostics `control$execution` owns memory, hardware, threading, precision and seeds. `control$diagnostics` owns verbosity, package-age warnings and profiling.

These modules do not form a Cartesian product of interchangeable algorithms. Model-dependent admission remains authoritative; a requested mode may be ineligible. Consult `mmes_control` for prerequisites, exclusions and current behavior. No new method combinations are implied by the modular API. The internal `.mmes_capabilities()` catalog is descriptive and does not resolve eligibility or predict execution.

Value

If all parameters are correctly indicated the program will return a list with the following information:

`data`	the dataset used in model fitting after application of the common observation mask.
`obsInfo`	an observation-level audit table describing the common missing-data filter. It includes the original row index, validity indicators for response, fixed, random, and residual model components, the final inclusion indicator, and the reason for exclusion when applicable.
`remlOmissions`	when present, a data frame identifying structurally unestimable random covariance components omitted from REML, with columns `term`, `parameter` and `reason`. It is also available in `engineDiagnostics$remlOmissions`.

`Dtable` the table to be used for the predict function to help the program recognize the factors available.

`llik` the vector of log-likelihoods across iterations

`b` the vector of fixed effects on the original response scale.

`u` the vector of random effects on the original response scale, consistent with `uList`.

`bu` the vector of fixed and random effects together on the original response scale, formed by concatenating `b` and `u`.

`Ci` the inverse of the coefficient matrix.

`Ci_11` the inverse of the coefficient matrix pertaining to the fixed effects.

`theta` a list of estimated variance covariance matrices. Each element of the list corresponds to the different random and residual components

`covPar` the fitted covariance parameters in their descriptor-defined reported coordinates.

`control` the validated modular numerical controls used for the fit. See `mmes_control`.

`pcgRecycleState`
present only when `pcg$exportRecycle=TRUE`. A versioned serializable Krylov basis with coefficient-coordinate metadata, for explicit import into a compatible PCG fit.

`covarianceInformation`
`"none"` when covariance information was explicitly omitted. The default retains full covariance information.

`retainCoefficientMatrix`
FALSE when an opt-in C-free saddle-point fit omitted the final compatibility matrix. In that mode `C` is NULL, and C-based post-fit methods require a refit retaining C. The default retains C.

`engineDiagnostics`
path-dependent numerical diagnostics. `compression` reports method, original observations, groups, residual classes, retained statistic bytes and setup time. `pcg` reports iterative work, reuse and stochastic accuracy diagnostics for REML PCG. `profile` contains detailed phase timings when explicitly enabled. Retained statistic bytes are not process peak memory.

`pcgMatrixFree` logical indicating whether the matrix-free Kronecker PCG path was used during optimization.

`residualCovarianceCompact`
logical indicating that a complete, unweighted, non-diagonal Kronecker residual covariance was evaluated directly from its factors without materializing the full covariance matrix. In this case the residual element of `theta` is a 0 $times0$ matrix; use `covparams_mmes()` for fitted residual covariance parameters.

`covParNative` a data frame containing all fixed and estimated covariance parameters in the native model-facing scale defined by each CovarianceFactor. For example, `dsm()` entries are level-specific variances rather than variance ratios. The table includes delta-method `StdError` and `Zratio` columns when covariance information is retained, matching `covparams_mmes_se()`. With `covarianceInformation="none"` these columns are NA; `covparams_mmes()` still returns point estimates.

theta_se	covariance-parameter uncertainty matrix; NULL when covarianceInformation="none".
InfMat	information matrix; NULL when covarianceInformation="none".
monitor	The values of the variance-covariance components across iterations during the REML estimation.
AIC	Akaike information criterion
BIC	Bayesian information criterion
convergence	a TRUE/FALSE statement indicating if the model converged.
partitions	a list where each element contains a matrix indicating where each random effect starts and ends.
partitionsX	a list where each element contains a matrix indicating where each fixed effect starts and ends.
percDelta	the matrix of percentage change in deltas (see control$convergence$parameterTol).
normMonitor	the matrix of the three norms calculated (see control$convergence$parameterTol).
toBoundary	the matrix of variance components that were forced to the boundary across iterations.
Cchol	the Cholesky decomposition of the coefficient matrix.
y	the response vector.
W	the column binded matrix W = [X Z]
uList	a list containing the BLUPs in data frame format where rows are levels of the random effects and column the different factors at which the random effect is fitted. This is specially useful for diagonal and unstructured models.
uPevList	a list containing the BLUPs in data frame format where rows are levels of the random effects and column the different factors at which the random effect is fitted. This is specially useful for diagonal and unstructured models.
args	the fixed, random and residual formulas from the mmes model.
constraints	The vector of constraints.

Author(s)

Coded by Giovanny Covarrubias-Pazaran with contributions of Christelle Fernandez Camacho, Johan Aparicio-Arce, and Claudio Flavio 5.1 jaja (laughing in Spanish).

References

Covarrubias-Pazaran G. Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 2016, 11(6): doi:10.1371/journal.pone.0156744

Jensen, J., Mantysaari, E. A., Madsen, P., and Thompson, R. (1997). Residual maximum likelihood estimation of (co) variance components in multivariate mixed linear models using average information. Journal of the Indian Society of Agricultural Statistics, 49, 215-236.

Sanderson, C., & Curtin, R. (2025). Armadillo: An Efficient Framework for Numerical Linear Algebra. arXiv preprint arXiv:2502.03000.

Gilmour et al. 1995. Average Information REML: An efficient algorithm for variance parameter estimation in linear mixed models. Biometrics 51(4):1440-1450.

Examples

```
####============================================####
#### For CRAN time limitations most lines in the
#### examples are silenced with one '#' mark,
#### remove them and run the examples
####============================================####

data(DT_example, package="enhancer")
DT <- DT_example
head(DT)

####============================================####
#### Univariate homogeneous variance models  ####
####============================================####

## Compound simmetry (CS) model
ans1 <- mmes(Yield~Env,
             random= ~ Name + Env:Name,
             rcov= ~ units,
             data=DT)
summary(ans1)

## Known variances: solve the MME only (e.g. genetic evaluation)
sol1 <- mmes(Yield~Env,
             random= ~ Name + Env:Name,
             rcov= ~ units,
             data=DT, control=mmes_control(task=list(mode="solve")),
             covPar=ans1)
head(sol1$uList[[1]])



####============================================####
#### Univariate heterogeneous variance models  ####
####============================================####
DT=DT[with(DT, order(Env)), ]
## Compound simmetry (CS) + Diagonal (DIAG) model
ans2 <- mmes(Yield~Env,
             random= ~Name + vsm(dsm(Env),ism(Name)),
             rcov= ~ dsumm(vsm(ism(units)), by=Env),
             data=DT)
summary(ans2)

####============================================####
####  Univariate unstructured variance models  ####
####============================================####

ans3 <- mmes(Yield~Env,
             random=~ vsm(usm(Env),ism(Name)),
             rcov= ~ dsumm(vsm(ism(units)), by=Env),
             data=DT)
summary(ans3)
```

```
## simulated MET
nInds=300
nMarks=1000
nEnvs=3

#random population of nInds lines with nMarks markers
M <- matrix(rep(0,nInds*nMarks),nInds,nMarks)
for (i in 1:nInds) {
  M[i,] <- ifelse(runif(nMarks)<0.5,-1,1)
}
# compute the relationship matrix
MMT <- tcrossprod(M)
m <- sum(diag(MMT))/nrow(MMT)
MMT <- MMT/m
MMT <- MMT + diag(1e-05, ncol(MMT), ncol(MMT))
# get phenotypes
rg <- enhancer::simGECorMat(nEnvs, 1, mu=0.2, v=0.2)
rg <- Matrix::nearPD(rg)$mat
rg # desired genetic correlation
u <- MASS::mvrnorm(n = nMarks, rep(0, nEnvs), Sigma=rg) # marker effects
g <- crossprod(t(M),u) # genetic values
h2 <- 0.2  #heritability
envEffect <- rnorm(nEnvs, mean = abs(rnorm(1))) # environment means
e <- MASS::mvrnorm(n = nInds, envEffect, Sigma=(1-h2)/h2*diag(diag(var(g)))) # errors
p = g + e # phenotypes
y <- as.vector(p) # phenotypes as vector
env <- sort(rep(paste0("e",1:nEnvs), nrow(g))) # environment labels
ind <- rep(paste0("i",1:nrow(g)), nEnvs ) # individuals labels
colnames(MMT) <- rownames(MMT) <- paste0("i",1:nrow(g)) # assign dimnames

MMTi <- solve(MMT + diag(1e-4,ncol(MMT),ncol(MMT)))
MMTi[lower.tri(MMTi)] <- t(MMTi)[lower.tri(MMTi)] # fill the lower triangular
MMTi <- as(as(as( MMTi,  "dMatrix"), "generalMatrix"), "CsparseMatrix")
attr(MMTi, 'inverse')=TRUE

DT <- data.frame(ind=ind, env=env, y=y)
# fit the models
mixm2 <- mmes(y ~ env,
              random=~ vsm(dsm(env), ism(ind), Gu=MMTi),
              data=DT
)
summary(mixm2)$varcomp
```

mmes_control *Modular Numerical Controls for Mixed Models*

Description

Creates a strictly validated sommer_mmes_control object for the numerical methods used by mmes. Model formulas remain separate. Every module accepts only its documented fields; unknown module or field names are errors. Constructing controls does not fit a model or establish that requested methods are eligible for a particular model.

Usage

```
mmes_control(formulation = "henderson", task = list(),
             representation = list(), solve = list(), logdet = list(),
             score = list(), curvature = list(), update = list(),
             proposal = list(), convergence = list(), output = list(),
             execution = list(), diagnostics = list())
```

Arguments

formulation	Equation domain: "henderson" (default) or "direct". Henderson forms coefficient-space mixed-model equations; direct uses observation-space inversion.
task	Named list with mode, objective, and pql. Defaults: mode="fit", objective="reml", and pql=list(maxit=20L, tol=1e-5, estimateTheta=FALSE, warmStart=TRUE). mode is "fit" or "solve"; objective is "reml" or "ml". The nested pql list accepts only maxit, tol, estimateTheta, and warmStart, with the defaults shown. These fields own task selection and outer PQL iteration, not inner covariance iterations.
representation	Named list with compression="off", factorScores="none", fixedRank="error", equations="normal", sparseFactors="auto", and matrixFree="auto" by default, unless corresponding sommer.mme.* options override experimental modes. sparseFactors="auto" enables exact sparse innovation kernels for modified-Cholesky factors of dimension at least 64 and maximum innovation lag three. It also enables sparse SAR/CAR kernels when the factor dimension is at least 500 and at most one percent of adjacency weights are nonzero. Other factors retain their existing path. Use "off" to disable or "force" to enable eligible sparse factor kernels regardless of size. If sommer.mme.saddlePoint resolves to "force", the default equations becomes "saddle-point". compression is "off", "groups", or "classes"; factorScores is "none", "fixed-shape", or "profile"; equations is "normal", "saddle-point", or "conditional" (experimental latent FA equations with independently selected AI/EM updates). fixedRank="error" retains the default error for rank-deficient fixed designs. These errors show the opt-in syntax before potentially long term and column labels so it remains visible when error output is truncated. Opt in with fixedRank="drop" to select a full-column-rank subset after observation filtering, before either the Henderson or direct engine runs. A shared C++ sparse QR removes zero columns

and detects linear dependencies on unit-norm columns, using pivot threshold `max(nrow(X), ncol(X)) * .Machine$double.eps`. COLAMD ordering and numerical pivoting choose the basis; retained columns are then restored to their original order. This need not select the same constraints as ASReml. No ridge penalty or generalized inverse is substituted for the fitted model. Omitted coefficients are returned as zero identification constraints, with unavailable standard errors and tests in summaries. `fit$fixedRank` records retained/omitted indices, original column names, rank, tolerance, and a sparse relation satisfying `X_original = X_retained %*% relation` to numerical precision. Direct-engine `VarBeta` remains in retained-column coordinates to avoid allocating a dense covariance over all original columns. Supported workflows are Gaussian identity-link fits, with normal or saddle-point Henderson equations or direct inversion, without PQL, solve mode, conditional FA, factor scores, observation rotation, or compression. Henderson predictions accept only contrasts estimable under the original fixed model; non-estimable rows produce an explicit error. Wald term tests, likelihood-ratio comparisons, Satterthwaite/Kenward-Roger inference, and `emmeans` currently require an explicitly full-rank refit. Existing direct-engine prediction/VarU limitations remain unchanged. REML likelihoods from different choices of fixed-effect basis can differ by an additive constant; compare covariance estimates and fitted values, not raw REML likelihoods across different bases. For rank-reduced fits, character classify predictions stop before allocating a Cartesian grid. Use `predict(fit, D="locale:cultivar", observed=TRUE)` for observation-weighted fixed predictions at combinations present in the retained data, or supply explicit estimable contrasts. The prediction `maxMemoryMB` safeguard checks dense solve/covariance workspace before allocation; see `predict.mmes` for weighting and limitations. The remaining fields use experimental mode values `"auto"`, `"off"`, or `"force"` (logical TRUE/FALSE are accepted). This module owns equation, compression, covariance-factor, and factor-score representations.

`solve` Named list. `method="auto"` selects among `"auto"`, `"ldlt"`, `"cholmod"`, and `"pcg"`. PCG fields default to `tol=1e-8`, `maxIters=0L`, `preconditioner="diagonal"`, `nystromRank=32L`, `warmStart=FALSE`, `recycleRank=0L`, `recycleMethod="coarse"`, `recycleState=NULL`, `exportRecycle=FALSE`, `rhsMethod="independent"`, `blockSize=16L`, and `residualRefresh=0L`. Also accepts the experimental solve requests `blockSchur="auto"`, `blockBanded="auto"`, `factorSchurMultiFactor="auto"`, and `dFirstML="auto"`; global `sommer.mme.*` options may override these mode defaults. The method owns checked linear solves, preconditioning, reuse, and eligible elimination. PCG settings apply only when an eligible PCG path uses them.

`logdet` Named list with `scaling="none"` and `lanczosSteps=20L` by default. Scaling is `"none"` or `"diagonal"`. This module owns determinant approximation and its correction, not linear-solve accuracy.

`score` Named list with `traceProbes=8L`, `estimator="hutchinson"`, `rank=16L`, `covarianceChecks="auto"`, and `traceBatch="off"` by default. The estimator is `"hutchinson"` or `"control-variate"`. `covarianceChecks` and `traceBatch` accept experimental mode values. This module owns covariance derivatives, trace estimation, and covariance checks.

`curvature` Named list with `method="exact-ai"` and `reduced="off"` by default. The method is `"exact-ai"` or `"hybrid-ai"`; `reduced` accepts experimental mode values. This module owns the AI metric and reduced-direction assembly.

update — Named list with `method="ai"`, `recovery="em"`, `blendWeight=NULL`, and `commonScale="off"` by default. `recovery` is `"em"` or `"pxem"`; `commonScale` accepts experimental mode values. This module owns covariance-coordinate updates, EM/PXEM recovery, blending, and common-scale profiling. `method="ai"` remains the default optimizer. With `representation$equations="conditional"`, it uses exact REML scores, average-information curvature, safeguarded steps and EM recovery in latent FA coordinates. `method="em"` selects conditional-moment EM and requires that representation. The earlier `method="conditional-fa"` selector continues to select conditional EM, not AI. Conditional equations support one FA relationship term with diagonal or FA residuals, including missing and unbalanced observations, using sparse LDLT posterior solves and selected inverse panels. They require Gaussian identity Henderson REML, `solve$method="auto"` or `"ldlt"`, `output=list(inverse=0, covariance="none", coefficientMatrix=FALSE)`, and no weights, rotation, `vcc`, other representation requests, or acceleration. Each source must be entirely free or entirely fixed. Conditional AI bounds its dense derivative-RHS and curvature workspace by `execution$denseMemoryMB`; `output$panelSize` also bounds projection solve panels. A rank-revealing eigensolve removes numerically null curvature directions. Accepted AI steps satisfy an observed-REML ascent check; failed proposals use an audited EM recovery step, never a silent estimator change. AI convergence requires a small diagonally scaled score using `convergence$parameterTol`; at numerical bounds it uses the projected (KKT) score. Conditional AI bounds free physical specific variances below by `max(1e-8, execution$inverseTol) * var(y)`, and shared genetic allocations to `[max(1e-8, execution$inverseTol), 1-max(1e-8, execution$inverseTol)]`. This prevents nearly deterministic latent equations from producing unreliable likelihoods. Starting values must respect these bounds; `execution$inverseTol` must be less than 0.5 on this path. The floors and active coordinate indices are reported in `conditionalFA$ai`. EM retains its existing machine-precision floor and likelihood/parameter stabilization; it is not silently changed to this constrained AI optimizer. See the conditional FA section of `vignette("sommer.mmes.options")`.

proposal — Named list with `stepWeight=NULL` and `extrapolation="none"` by default. Extrapolation is `"none"` or `"aitken"`. This module owns proposal scaling and extrapolation. Aitken extrapolation cannot be combined with `curvature$method="hybrid-ai"`.

convergence — Named list with `maxIters=30L`, `likelihoodTol=1e-4`, `parameterTol=1e-4`, `accuracy="fixed"`, `maxTraceProbes=NULL`, `maxLanczosSteps=NULL`, `logDetTol=1e-3`, and `scoreTol=1e-3`. Accuracy is `"fixed"` or `"adaptive"`; NULL stochastic limits resolve to the initial `traceProbes` and `lanczosSteps`. This module owns inner iteration convergence and stochastic accuracy requests.

output — Named list with `inverse=0L`, `covariance="full"`, `coefficientMatrix=TRUE`, `setup=FALSE`, `selectedInverse="auto"`, and `panelSize=64L` by default. `inverse` is 0 (no final coefficient inverse or fit-time PEV computation), 1 (selected inverse and random-effect PEVs), or 2 (full inverse and random-effect PEVs). Modes 1 and 2 compute PEVs during fitting; mode 1 does not form the complete coefficient-matrix inverse. With mode 0, use postPEV after fitting to compute PEVs or the full inverse, provided `coefficientMatrix=TRUE`; `covariance` is `"full"` or `"none"`; `selectedInverse` accepts experimental mode values. This module owns coefficient uncertainty, covariance-parameter uncertainty, retained coefficient-matrix output, and setup output.

`execution` Named list with `denseMemoryMB=1000`, `memoryPolicy="speed"`, `gpu=FALSE`, `gpuMinSize=256L`, `blasThreads=NULL`, `inverseTol=1e-6`, and `seed=0L` by default. A non-NULL `sommer.mme.denseMemoryMB`, `sommer.mme.memoryPolicy`, `sommer.mme.gpu`, or `sommer.mme.gpuMinSize` option supplies its corresponding default. This module owns resource policy, hardware, threading, numerical inverse tolerance, and stochastic seed.

`diagnostics` Named list with `verbose=TRUE`, `dateWarning=TRUE`, and `profile="off"` by default. `profile` accepts experimental mode values. This module owns ordinary fit messages, package-age warnings, and phase profiling.

Details

Module ownership describes mathematical responsibility, not a promise that every cross-module combination is supported. For example, formulation and solver choices are model-dependent; PCG has REML and output restrictions; compression, saddle-point equations, factor-score augmentation, common-scale profiling, and reduced AI each have additional prerequisites. Consult `vignette("sommer.mmes.options", package="sommer")` and the internal `.mmes_capabilities()` catalog for the current conceptual decision graph and capability inventory. The catalog is descriptive; fitting validation remains authoritative. The modular API does not imply a kernel rewrite or arbitrary method composition.

`task$mode="solve"` selects a known-covariance path rather than ordinary variance-component fitting; it requires an eligible Gaussian identity model and has narrower PCG/output options. `task$pql$estimateTheta=TRUE` requires a negative-binomial family. `representation$compression` needs eligible matching designs and independent residuals. Saddle-point equations, factor-score representation, sparse factors, and matrix-free PCG each have separate model and backend gates. PCG fitting is a Henderson REML path; selected-inverse output mode 1 is not supported by PCG. The vignette and capability catalog describe these prerequisites, but the fit-time validator decides eligibility.

Within PCG, `solve$preconditioner="nystrom"` uses `solve$nystromRank`; `solve$recycleState` and `solve$exportRecycle` require `solve$recycleMethod="krylov"` and a positive `solve$recycleRank`. `logdet$scaling` and Lanczos depth apply to stochastic determinant evaluation. `score$rank` is used by the control-variate estimator. Adaptive trace and Lanczos budgets apply to stochastic PCG accuracy. `output$panelSize` applies to full inverse panels and conditional FA posterior-moment/projection panels, not ordinary selected-inverse mode.

Experimental mode fields accept `"auto"`, `"off"`, or `"force"`; logical TRUE and FALSE mean force and off. Package options may supply defaults for these requests. `"force"` requests a path but does not bypass mathematical, structural, or memory eligibility checks. Nested lists reject unknown names and duplicate or empty names.

Defaults retain the established numerical bundle. Aitken belongs to `proposal$extrapolation`; hybrid AI belongs to `curvature$method`; they are mutually exclusive.

Migration from removed numerical arguments

`mmes()` now takes numerical settings through `control`. The old formals are not compatibility aliases and must not be passed to `mmes()`. Map former settings to canonical module paths:

`henderson` `formulation="henderson"` or `formulation="direct"`.

`solver`, `engine` `solve$method`; pass the complete control object as `control=...`.

`nIters`, `tolParConvLL`, `tolParConvNorm`, `tolParInv` `convergence$maxIters`, `convergence$likelihoodTol`, `convergence$parameterTol`, and `execution$inverseTol`, respectively.

`REML`, `solveOnly`, `pqlControl` `task$objective` (`"reml"` or `"ml"`), `task$mode` (`"fit"` or `"solve"`), and `task$pql`.

`computeCi`, `getPEV`, `returnParam` `output$inverse` and `output$setup`. PEV output is controlled by `output$inverse`; the former `getPEV` flag has no separate replacement.

`blend`, `blendWeight`, `stepWeight` `update$recovery`, `update$blendWeight`, and `proposal$stepWeight`.

`acceleration="aitken"` `proposal$extrapolation="aitken"`.

`acceleration="hybrid-ai"` `curvature$method="hybrid-ai"`. Do not combine with Aitken.

`compression`, `factorScoreAugmentation`, `covarianceInformation`, `retainCoefficientMatrix` `representation$compression`, `representation$factorScores`, `output$covariance`, and `output$coefficientMatrix`.

Former `pcg` **settings** Map solve tolerances, iteration limits, preconditioning, and reuse to `solve`; determinant scaling and Lanczos depth to `logdet`; trace probes, estimator, rank, and covariance checks to `score`; stochastic accuracy budgets to `convergence`; seed to `execution`.

Former `experimental` **settings** Map structured representation requests to `representation`, elimination requests to `solve`, trace batching to `score`, reduced AI to `curvature`, common-scale profiling to `update`, selected/full inverse requests to `output`, and profiling to `diagnostics`.

`denseMemoryMB`, `memoryPolicy`, `gpu`, `gpuMinSize`, `blasThreads` `execution` fields with the same names.

`verbose`, `dateWarning` `diagnostics` fields with the same names.

`recordDiagnostics`, `verboseDiagnostics` Removed flags with no direct replacement. Use `diagnostics$verbose` for fit messages and `diagnostics$profile` for implemented phase profiling.

With `diagnostics$verbose=TRUE`, the existing Henderson `MME` `path` line includes the coefficient solver, active/inactive latent reduced-rank and Schur variants, and the selected residual engine (including inactive grid and repeated-Kronecker paths). An inactive automatic Schur path is reported as "not selected", not as proof that the model lacks exploitable structure; disabled controls and unsupported coefficient backends are distinguished.

Value

An object of class `"sommer_mmes_control"`.

Compatibility

Settings are checked against one compatibility table when the control is built and again in `mmes`. A supplied setting that conflicts with another, or that would have no effect on the selected path, stops the call with a message that names every offending setting and how to fix it. A setting counts as supplied only when it differs from its default, so restating defaults is always accepted. The main rules are:

- `formulation="direct"` accepts only `task`, `update$recovery`, `update$blendWeight`, `proposal$stepWeight`, convergence iterations and tolerances, `output$setup`, `output$inverse` 0 or 2, memory and threading `execution` settings, and `diagnostics`.
- `task$mode="solve"` accepts only `solve$tol`, `solve$maxIters`, `solve$warmStart`, `solve$residualRefresh`, a non-Nystrom `solve$preconditioner`, `output$setup`, CPU `execution` settings and `diagnostics`.

- PCG-only settings (solve tolerances, preconditioner, recycling, `logdet`, `score` estimators, adaptive accuracy, `representation$matrixFree`, `execution$seed`) require `solve$method="pcg"`; `"auto"` never selects PCG.
- `solve$method="pcg"` rejects EM/PXEM recovery, blending, common-scale profiling, reduced AI, Aitken, hybrid AI, saddle-point equations, factor scores and `task$objective="ml"`.
- Aitken extrapolation, hybrid AI, saddle-point equations, factor scores, blending and `output$coefficientMatrix=FALS` each reject the modules they cannot combine with.

Rules that depend on the data, family, `vcc` or rotation are checked by `mmes` at fit time; for example a Gaussian identity fit stops when `task$pql` is changed.

See Also

`mmes`; `vignette("sommer.mmes.options", package="sommer")`

Examples

```
ctrl <- mmes_control(
  formulation = "henderson",
  solve = list(method = "pcg", tol = 1e-8),
  score = list(traceProbes = 16L),
  convergence = list(accuracy = "adaptive")
)
print(ctrl)
```

---

mme_workload_grid *Generate a Factorial Grid of MME Workloads*

Description

Generate a Factorial Grid of MME Workloads

Usage

```
mme_workload_grid(
  structures = c("ar1", "us", "diag"),
  n_ind = c(40L, 100L),
  n_levels = c(4L, 10L),
  relationships = c("none", "dense"),
  reml = TRUE,
  computeCi = 0L
)
```

Arguments

structures	Vector of covariance structures to test.
n_ind	Vector of individual counts.
n_levels	Vector of factor level counts.
relationships	Vector of relationship types.
reml	Logical; whether to fit by REML.
computeCi	Vector of computeCi modes.

Value

A named list of mme_workload objects.

MNR *Multivariate Newton-Raphson algorithm*

Description

Multivariate Newton-Raphson algorithm used behind mmes when the henderson argument is set to FALSE. Algorithm made available for users that want to avoid the user-friendly interface and provide their matrices directly.

Usage

```
MNR(Y, # response
    X, Gx, # fixed effects
    Z, K, # random effects
    R,  # residual effects
    Ge, GeI, # initial values and constraints
    W, isInvW, # weights matrix
    iters, tolpar, tolparinv, # other params
    ai, pev,
    verbose, retscaled,
    stepweight, emweight
    )
```

Arguments

Y	A matrix with rows for records and columns for traits. Expected class is a 'matrix'.
X	Design matrices for fixed effects. Expected class is a 'list' with as many design matrices as fixed effects. Matrices within the list should be of class 'matrix'
Gx	Trait multiplier matrix for fixed effects. Expected class is a 'list' with as many matrices as fixed effects. Each matrix is a square matrix with as many rows and columns as number of traits. Each matrix is of class 'matrix'

Z	Design matrices for random effects. Expected class is a 'list' with as many design matrices as random effects. Each element in the list should be a matrix of class 'dgCMatrix'
K	Covariance matrices for design matrices of random effects. Expected class is a 'list' with as many covariance matrices as random effects specified in Z. Each element in the list should be a matrix of class 'dgCMatrix'
R	Residual matrices for residual effects. Expected class is a 'list' with as many residual matrices as residual effects. Each element in the list should be a square matrix of dimensions n x n, where n is the number of records. Each matrix should be of class 'matrix'
Ge	Initial values for variance components. Expected class is a 'list' with as many variance component matrices as random effects specified in Z. Each element in the list should be a matrix of dimensions t x t, where t is the number of traits and of class 'matrix'. Initial values are any real values.
GeI	Initial constraints for variance components. Expected class is a 'list' with as many variance component constraints matrices as random effects specified in Z. Each element in the list should be a matrix of dimensions t x t, where t is the number of traits and of class 'matrix'. Values expected are: 0: not to be estimated 1: estimated and constrained to be positive (i.e. variance component) 2: estimated and unconstrained (can be negative or positive, i.e. covariance component) 3: not to be estimated but fixed (value has to be provided in the Gti argument) Please notice that lower triangular values of these matrices have to be equal to zero since only the values in the upper triangular are estimated.
W	Design matrix for weights. A matrix of class 'matrix' for weighting the records.
isInvW	A value of class 'logical' to indicate if the W matrix provided is already an inverse or not. This aims to speed up the computations.
iters	Maximum number of iterations allowed in REML. Value is of class 'integer'.
tolpar	Convergence criteria for the change in log-likelihood. Value is of class 'numeric'.
tolparinv	Tolerance parameter for matrix inverse used when singularities are encountered in the estimation procedure. Value is of class 'numeric'.
ai	A value of class 'logical' to indicate if the Average Information algorithm should be used instead. That is faster but much less stable.
pev	A value of class 'logical' to indicate if the predicted error variance should be computed or not. If FALSE computations are speeded up for models with many effects to be estimated.
verbose	A value of class 'logical' to value to indicate if the program should return the progress of the iterative algorithm.
retscaled	A value of class 'logical' to indicate if we should avoid scaling the traits.
stepweight	Vector of class 'numeric' with length equal to niters to specify the relative weight given to the second derivative Newton update.
emweight	Vector of class 'numeric' with length equal to niters to specify the relative weight given to the EM update compared to the NR update.

Details

This is the Rcpp-coded Direct-Inversion LMM REML algorithm used behind the mmer function.

Value

If all parameters are correctly indicated the program will return a list with the following information:

`Vi`	the inverse of the phenotypic variance matrix V^- = (ZGZ+R)^-1
`P`	the projection matrix Vi - [Vi*(X*Vi*X)^-*Vi]
`sigma`	a list with the values of the variance-covariance components with one list element for each random effect.
`sigma_scaled`	a list with the values of the scaled variance-covariance components with one list element for each random effect.
`sigmaSE`	Hessian matrix containing the variance-covariance for the variance components. SE's can be obtained taking the square root of the diagonal values of the Hessian.
`Beta`	a data frame for trait BLUEs (fixed effects).
`VarBeta`	a variance-covariance matrix for trait BLUEs
`U`	a list (one element for each random effect) with a data frame for trait BLUPs.
`VarU`	a list (one element for each random effect) with the variance-covariance matrix for trait BLUPs.
`PevU`	a list (one element for each random effect) with the predicted error variance matrix for trait BLUPs.
`fitted`	Fitted values y.hat=XB
`residuals`	Residual values e = Y - XB
`AIC`	Akaike information criterion
`BIC`	Bayesian information criterion
`convergence`	a TRUE/FALSE statement indicating if the model converged.
`monitor`	The values of log-likelihood and variance-covariance components across iterations during the REML estimation.
`percChange`	The percent change of variance components across iterations. There should be one column less than the number of iterations. Calculated as percChange = ((x_i/x_i-1) - 1) * 100 where i is the ith iteration.
`dL`	The vector of first derivatives of the likelihood with respect to the ith variance-covariance component.
`dL2`	The matrix of second derivatives of the likelihood with respect to the i.j th variance-covariance component.

References

Mrode, R. A. (2014). Linear models for the prediction of animal breeding values. Cabi.

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

See Also

mmes – the core function of the package

Examples

```
data(DT_cpdata, package="enhancer")
DT <- DT_cpdata


# response matrix
y <- cbind(imputev(DT$Yield), imputev(DT$Firmness))
# fixed effect incidence matrix
X <- model.matrix(~Rowf, data=DT)
# random effect incidence matrix
Z <- Matrix::sparse.model.matrix(~id - 1, data=DT)
colnames(Z) <- gsub("id","",colnames(Z))
# covariance for random effect
GT <- GT_cpdata
A <- A.mat(GT) # additive relationship matrix
A <- A[colnames(Z),colnames(Z)] # make sure of the order
A <- A + diag(1e-4,nrow(A), nrow(A))
# residual effect incidence matrix (dimensions equal nrow(y))
R1 <- Matrix::Diagonal(n=nrow(y))
# weights matrix (dimensions equal nrow(y))
W <- diag(nrow(y))
# some other parameters
maxIter=3
stepWeight <- rep(0.9, maxIter)
stepWeight[1:2] <- c(0.5, 0.7)
emWeights <- rep(0,maxIter)
# model fit
res <- MNR( Y=y, # multi-trait response
            X=list(X), Gx=list(diag(2)), # fixed effects
            Z=list(Z), K=list(A), # random effects
            R=list(R1), # residual effects
            Ge=list( (diag(2)*.3)+.15 , diag(2)*0.75 ), # inital vc
            GeI=list(unsm2(2),diag(2)), # vc constraints
            W=W, isInvW=TRUE, # weights for records
            iters=maxIter,
            tolpar=1e-4, tolparinv=1e-6, # tolerance
            ai=FALSE, pev=FALSE, # algorithm specifics
            verbose=TRUE,
            retscaled=FALSE,
            stepweight=stepWeight, # second derivatives weights
            emweight=emWeights # em update weights
)
res$sigma
res$Beta
res$U[[1]]
```

ownm *User-defined covariance structure*

Description

User-defined covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
ownm(x, K = NULL, fun = NULL, par = numeric(), fixed = NULL,
  dfun = NULL, par_names = NULL, native_report = NULL)
```

Arguments

x	Variable or design defining q covariance levels.
K	Optional known positive-definite q by q covariance matrix. If supplied, the structure is fixed.
fun	Optional function fun(par) returning a finite q by q covariance matrix.
par	Starting parameter vector for fun.
fixed	Logical vector of length par indicating fixed parameters.
dfun	Optional derivative function dfun(par, k) returning the derivative of the raw covariance matrix with respect to parameter k.
par_names	Optional names for the parameters.
native_report	Optional function with arguments scale, par, factor, and absorb_scale. It must return a named numeric vector giving the fitted parameters in the model's preferred native scale. This callback is used by covparams_mmes().

Details

There are two modes. With K, ownm creates a fixed covariance factor. With fun, the matrix is evaluated at every optimizer point. The returned matrix is symmetrized and normalized by its first diagonal element so that vsm retains the unique overall variance scale. If dfun is supplied, its derivative is normalized analytically using the quotient rule; otherwise a central numerical derivative of the normalized factor is used. This is the general extension mechanism for covariance structures that do not require a native C++ evaluator. When native_report is omitted, the overall variance and naturally transformed par values are reported.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

vsm, covparams_mmes, rrm, maternm, sar, car.

Examples

```
## Not run:
K <- matrix(c(1,.3,.3,1),2,2)
vsm(ownm(group, K=K), ism(id))

cfun <- function(p) {
  r <- tanh(p[1]); matrix(c(1,r,r,1),2,2)
}
native <- function(scale, par, factor, absorb_scale=TRUE) {
  c(variance=scale, correlation=tanh(par[1]))
}
vsm(ownm(group, fun=cfun, par=0, native_report=native), ism(id))

## End(Not run)
```

`plot.mmes` *plot form a LMM plot with mmes*

Description

`plot` method for class `"mmes"`.

Usage

```
## S3 method for class 'mmes'
plot(x,stnd=TRUE, ...)
```

Arguments

`x`	an object of class `"mmes"`
`stnd`	argument for ploting the residuals to know if they should be standarized.
`...`	Further arguments to be passed

Value

vector of plot

Author(s)

Giovanny Covarrubias <covarrubiasp@wisc.edu>

See Also

`plot`, `mmes`

Examples

```
data(DT_yatesoats, package="enhancer")
DT <- DT_yatesoats
head(DT)
m3 <- mmes(fixed=Y ~ V + N + V:N,
           random = ~ B + B:MP,
           rcov=~units,
           data = DT)
```

---

pmonitor *plot the change of VC across iterations*

Description

`plot` for monitoring.

Usage

```
pmonitor(object, ...)
```

Arguments

`object`	model object of class `"mmes"`
`...`	Further arguments to be passed to the plot function.

Value

vector of plot

Author(s)

Giovanny Covarrubias

See Also

`plot`, `mmes`

Examples

```
data(DT_yatesoats, package="enhancer")
DT <- DT_yatesoats
head(DT)
m3 <- mmes(fixed=Y ~ V + N + V:N,
           random = ~ B + B:MP,
           rcov=~units,
           data = DT)
pmonitor(m3)
```

postPEV *Post-fit prediction error variances and inverse coefficient matrix*

Description

Computes prediction error variances (PEVs) or the complete inverse of the mixed model coefficient matrix after a model has been fitted with `mmes`. For PEV-only calculations, the function uses the Takahashi sparse inverse subset approach, avoiding computation of the complete inverse of the coefficient matrix.

Usage

```
postPEV(object, mode = 1L)
```

Arguments

`object`	A fitted model object of class `"mmes"`. The object must contain the final sparse mixed model coefficient matrix and its associated scaling factor.
`mode`	Integer specifying the post-fit inverse calculation to perform. Use `0` to perform no inverse calculation and clear previously computed inverse-derived outputs, `1` to compute the prediction error variances using the Takahashi sparse inverse subset approach without forming the complete inverse, or `2` to compute the complete inverse of the coefficient matrix and the prediction error variances. The default is `1`.

Details

The function is intended for post-processing models fitted with `mmes`, allowing prediction error variances or the complete inverse of the mixed model coefficient matrix to be calculated only when they are needed.

When `mode = 1`, the sparse mixed model coefficient matrix stored in the fitted object is factorized and the Takahashi equations are used to obtain the selected elements of its inverse required for the diagonal prediction error variances. The complete inverse is not formed.

When `mode = 2`, the complete inverse of the mixed model coefficient matrix is computed. The prediction error variances are then obtained from the corresponding diagonal elements of this inverse.

When `mode = 0`, no matrix factorization or inverse calculation is performed and previously stored inverse-derived results are cleared.

`predict.mmes` does not require any particular `mode`: it computes exact standard errors for arbitrary linear combinations of fixed and random effects directly from the stored coefficient matrix `C`, without using `Ci` at all (see `predict.mmes` for details). `postPEV()` is only needed when the diagonal prediction error variances (`uPevList`) or the complete inverse (`Ci`) are wanted for their own sake.

For the sampling variance of the BLUP itself, use `postVarU`: $\mathrm{Var}(\hat{u}) = G - \mathrm{Var}(u - \hat{u})$. PEV and BLUP sampling variance are different quantities. Neither includes uncertainty from estimating covariance parameters. `postVarU()` leaves existing PEV outputs unchanged.

For this post-fit calculation to be available, fit the mmes model with control = mmes_control(output = list(coefficientMatrix = TRUE)) so the final coefficient matrix is retained. Models can use control$output$inverse = 0 during fitting, then calculate PEVs or the complete inverse afterwards only if required.

Value

An object of class "mmes" containing the original fitted model together with the requested post-fit inverse information.

With mode = 1, uPevList contains random-effect prediction error variances. Ci may contain the selected inverse entries used to obtain those PEVs, but it does not contain the complete inverse.

With mode = 2, Ci contains the complete inverse of the mixed model coefficient matrix and uPevList contains the corresponding prediction error variances for the random effects.

With mode = 0, inverse-derived outputs are cleared.

The returned object also contains CiComputed and CiMode. CiComputed is TRUE only when the complete inverse was computed; CiMode records the selected mode.

References

Takahashi, K., Fagan, J., and Chin, M. S. (1973). Formation of a sparse bus impedance matrix and its application to short circuit study. Proceedings of the 8th PICA Conference, Minneapolis, Minnesota.

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
####============================================####
#### Fit a mixed model without computing PEVs
#### or the complete inverse during estimation
####============================================####

# mod <- mmes(
#   fixed = Yield ~ 1,
#   random = ~ vsm(ism(Genotype)),
#   rcov = ~ units,
#   data = DT,
#   control = mmes_control(output = list(inverse = 0))
# )

####============================================####
#### Compute PEVs afterwards using the
#### Takahashi sparse inverse subset
####============================================####

# mod <- postPEV(mod, mode = 1)
# mod$uPevList

####============================================####
```

```
#### Compute the complete inverse afterwards
####==========================================####

# mod <- postPEV(mod, mode = 2)
# mod$Ci
# mod$uPevList
```

---

postVarU *Post-fit Sampling Covariance of BLUPs*

Description

Computes the plug-in sampling variance or covariance of fitted random-effect predictors, $\hat{u}$, using Henderson's mixed-model coefficient system. This is distinct from the prior variance of the random effects and their prediction error variance (PEV).

Usage

```
postVarU(object, mode=1L)
```

Arguments

`object`	A Gaussian `mmes` Henderson fit with stored `C` and `Cscale`. Direct-engine, PQL and matrix-free `solveOnly` fits are not supported.
`mode`	`1` (default) returns diagonal BLUP sampling variances in `uVarList`; `2` additionally returns the joint sampling covariance matrix `VarU`; `0` clears only VarU-derived outputs.

Details

For $y = X\beta + Zu + e$, with $\mathrm{Var}(u) = G$ and $\mathrm{Var}(e) = R$, the unscaled Henderson matrix is

$$C_0 = \begin{bmatrix} X'R^{-1}X & X'R^{-1}Z \\ Z'R^{-1}X & Z'R^{-1}Z + G^{-1} \end{bmatrix}.$$

Its random-effect inverse block is $Q = (C_0^{-1})_{uu} = \mathrm{Var}(u - \hat{u})$. Consequently,

$$\mathrm{Var}(\hat{u}) = G - Q.$$

For known covariance parameters this identity is exact. At estimated parameters the result is a plug-in covariance, omitting covariance-parameter estimation uncertainty. The same identity is used for ML and REML fits.

The implementation does not construct or invert the observation covariance $V = ZGZ' + R$. Error covariance is computed by solving the stored Henderson system and multiplying by `Cscale` to recover the original response scale. For each independent random term, $G_k = \Sigma_k \otimes A_k$, where Σ_k is the fitted covariance-coordinate matrix and A_k the relationship covariance. Products with A_k are evaluated by solving its retained sparse precision, not by explicitly forming its inverse or the full joint G.

mode=1 processes identity contrasts in batches of at most 128 rows, so neither a full coefficient inverse nor a full random-effect covariance matrix is stored. It requires solves for all requested effect diagonals and can be expensive for large models. mode=2 requires quadratic storage in the number of random effects. Although independent terms have zero prior cross-covariance, their BLUPs can have nonzero sampling cross-covariance; the joint VarU includes these cross-term entries.

Rotation and factor-score augmentation are mapped to the public uList coordinates before computing error covariance. New fits retain randomPrecision in original level order, so external Gu objects are not needed after fitting. Older ordinary fits attempt to rebuild relationship precisions from saved model inputs, requiring the original formula objects; dimension checks cannot detect all changes to those objects. Older rotated or augmented fits must be refitted. No clipping is performed: small negative variances from numerical cancellation remain visible.

Existing Ci and uPevList are left unchanged. The variance here is not the empirical variance across BLUP values and is not a posterior variance of u. It is the repeated-sampling covariance of $\hat{u}$.

Value

The original fitted object with uVarList, a named list matching the dimensions and labels of uList, and uVarMode. With mode=2, VarU is the joint covariance in term order, with each uList matrix stacked column by column. Its labels contain term, covariance-coordinate and main-effect labels. Modes 0 and 1 remove any previous full VarU; mode 0 also removes uVarList.

See Also

postPEV, predict.mmes, covmatrix_mmes

Examples

```
data(DT_example, package="enhancer")
model <- mmes(Yield~Env,
  random=~Name,
  data=DT_example,
  control = mmes_control(output = list(inverse = 0),
  diagnostics = list(verbose = FALSE)))
model <- postVarU(model)
head(model$uVarList[[1]])
full <- postVarU(model, mode=2)
head(diag(full$VarU))

prediction <- predict(model, D="Name", PEV=TRUE, VarU=TRUE)
head(prediction$pvals)
prediction$PEV[1:3, 1:3]
prediction$VarU[1:3, 1:3]
prediction$sampling.vcov[1:3, 1:3]
```

predict.mmes *Predict form of a LMM fitted with mmes*

Description

predict method for class "mmes".

Usage

```
## S3 method for class 'mmes'
predict(object, Dtable=NULL, D, levels=NULL, sed=FALSE,
        pairwise=FALSE, adjust="none", df=Inf, PEV=TRUE, VarU=TRUE,
        observed=FALSE, maxMemoryMB=NULL, ...)
```

Arguments

object	a mixed model of class "mmes"
Dtable	a table specifying the terms to be included or averaged. An "include" term uses the cells matching each prediction row. For categorical random effects these are mapped within the fitted term, including fitted cells without records. Numeric fixed terms retain their evaluated covariate values rather than being replaced by indicators. An "include and average" term means that the included cells of that effect are averaged: each prediction row is divided by the number of cells of the term present with it (for fixed terms this count includes the reference cells absorbed in the intercept). An "average" term alone averages over the selected factor levels. For categorical terms the weights are 1/#levels; numeric fixed terms are evaluated at their covariate settings before averaging. If a term is not considered "include" or "average" is then totally ignored in the BLUP and SE calculation. The default rule to invoke when the user doesn't provide the Dtable is to include and average all terms that match the argument D. Dtable contains only term-level rules, not level selections or numeric settings. Supply these through the levels argument; resolved settings are returned separately as levels. A levels column in a supplied Dtable is rejected; move its non-NULL entries to the named levels argument. With observed=TRUE, start from object$Dtable and change the random-term flags. Fixed flags must remain unchanged: all fixed terms use empirical retained-record averages. For random terms, include=TRUE alone uses the empirical within-group average of the fitted random design. With an indicator design, include=TRUE, average=TRUE instead gives equal weight to selected coefficient levels that co-occur with each prediction group. A group with no such levels is an error. include=FALSE, average=TRUE gives equal weight to all selected fitted coefficient levels, even if not observed within a prediction group. Both flags FALSE exclude the term. Random-level restrictions may be supplied

through the levels argument. These observed-path weights do not construct a Cartesian grid.

D — a character string specifying the variable used to extract levels for the rows of the D matrix and its construction. Alternatively, the D matrix (of class dgCMatrix) specifying the matrix to be used for the predictions directly.

levels — an optional named list whose names are values of Dtable$term. Selections and resolved numeric settings are returned separately as levels, and can be reused alongside the returned Dtable. NULL entries mean unrestricted factor levels or default numeric settings. Levels are written as data levels for fixed factors ("a:b" for interactions, e.g. "Victory:0.2"), as cell labels for categorical random terms (e.g. "CA.2011:genotype1" for an Env by Name term), and as numeric settings for covariate terms. A term containing one numeric variable accepts a numeric scalar, e.g. levels=list("Env:cov"=5). Numeric or mixed terms also accept a named per-variable list, e.g. levels=list("Env:cov"=list(Env=c("CA.2011", "CA.2012"), cov=5)). Each numeric setting must be one finite value. Use the exact term spelling in object$Dtable$term. For an included categorical term matching the classify variable(s) the levels select the prediction rows, in the given order, and may include levels without records (e.g. levels only present in Gu). For other included or averaged terms, only the listed levels enter the inclusion or the average. Restrictions are term-specific: selecting levels of V does not automatically restrict V:N.

sed — if TRUE, return the matrix of standard errors of differences sed and its summary avsed (mean, minimum and maximum).

pairwise — TRUE for all pairwise differences between predictions, or a single prediction level to compare every other prediction against it.

adjust — multiplicity adjustment passed to p.adjust.

df — degrees of freedom for pairwise tests: Inf (normal-based, default), a positive number, "residual" for n - rank(X), or "satterthwaite"/"kr" (see wald_mmes). The latter two apply to differences involving only fixed effects; differences involving random effects keep normal-based tests (df=Inf). With "kr" the SED of those differences uses the Kenward-Roger adjusted covariance.

PEV — Logical, default TRUE. Return the full target prediction-error covariance as PEV and its diagonal as pvals$pev. When FALSE, only these additional outputs are omitted; vcov and std.error remain available for compatibility.

VarU — Logical, default TRUE. Return the sampling covariance of the random contribution as VarU, its diagonal as pvals$var.u, and the sampling covariance of the entire predictor as sampling.vcov. Set FALSE to skip the additional relationship solves and outputs. These calculations require a Gaussian Henderson fit. For PQL predictions use VarU=FALSE; their existing working-model uncertainty remains an approximation.

observed — Logical, default FALSE. For rank-reduced Henderson fits, character classify predictions stop before expanding a Cartesian grid by default. Opt in with observed=TRUE to predict the observation-weighted fixed predictor within each observed categorical combination of D in the retained fitting data. All fixed terms contribute their empirical within-group averages; random contributions are excluded by default but can be explicitly included or averaged using random rows of Dtable.

This is not a balanced-grid marginal mean. Only categorical classify variables in the fixed formula are supported. `levels` may contain an entry named exactly as `D`, with unique observed labels to select and order rows, and entries named by fitted random terms to restrict their coefficients. Random labels may be coefficient names, main-effect levels, covariance coordinate levels, or structured cell labels such as `"north:b3"`. Unobserved requested combinations are errors, not silently removed. Explicit matrix contrasts must use `observed=FALSE` and are still checked for estimability. This option is not used for full-rank fits.

`maxMemoryMB` Positive finite workspace budget in decimal MB for rank-reduced predictions. Defaults to the fitted `control$execution$denseMemoryMB`, or 1000 MB for older objects. Before grouped design construction or covariance solves, an estimate includes dense engine-coordinate right-hand sides, solve copies, covariance outputs, and requested SED/pairwise workspace. It is an allocation safeguard, not a bound on total process memory or sparse factorization fill-in. Exceeding the budget raises an error suggesting fewer levels or smaller explicit contrast batches. `PEV=FALSE` does not eliminate the mandatory full `vcov`. Increase the budget only when memory is available. Existing full-rank prediction behavior is unchanged.

... Further arguments to be passed.

Details

This function allows to produce predictions specifying those variables that define the margins of the hypertable to be predicted (argument D). Predictions are obtained for each combination of values of the specified variables that is present in the data set used to fit the model. See vignettes for more details.

For a rank-reduced fit, `predict(fit, D="locale:cultivar", observed=TRUE)` constructs sparse averages directly from retained observed combinations, without generating the locale-by-cultivar Cartesian product. Returned rows are ordered by first appearance unless `levels` specifies their order. Missing-response rows excluded from the fit cannot create prediction rows. An observed marginal prediction, such as `D="locale"`, averages the fitted fixed predictor using the observed records in each locale; it is not an equally weighted or cultivar-adjusted comparison. Estimability is checked before covariance computation. Manual random contributions are added in the same public coefficient coordinates as explicit contrasts, so their prediction-error, sampling covariance and pairwise calculations follow the usual Henderson path. When `include=TRUE` alone has random-level restrictions, omitted coefficient contributions are zeroed without renormalizing the empirical group weights. Use `include=TRUE, average=TRUE` for equally weighted selected co-occurring indicator levels.

For categorical structured random terms, the fitted covariance-coordinate and main-effect levels define the genetic prediction weights, even when a cell has no observations. The `levels` argument can additionally request fitted levels present only in `Gu`. Unsupported random designs that cannot be mapped for all prediction rows produce a warning; an explicit `D` matrix can be used for these designs.

Numeric fixed terms, including factor-by-covariate interactions, are evaluated using the fitted fixed formula and contrasts. Unless overridden, each numeric variable is set to its mean in the retained fitting data. Factor levels are equally weighted when averaged. Thus an averaged `Env:cov` term with one slope per environment uses `mean(cov)/nlevels(Env)` for each slope, not `1/nlevels(Env)`.

Write $D = [L_b, L_u]$. The target is $t = L_b\beta + L_u u$ and its predictor is $\hat{t} = L_b\hat{\beta} + L_u\hat{u}$. The existing prediction `PEV` and `VarU` matrices are for the linear combinations represented by the final returned `D`, after applying `Dtable` and `levels`, not copies of model-level effect variances. Each covariance matrix has one row and column per requested prediction. The existing `vcov`, optional `PEV`, and squared `std.error` refer to $\mathrm{Var}(t - \hat{t})$, not generally to $\mathrm{Var}(\hat{t})$. They are computed from

$$\mathrm{Var}(t - \hat{t}) = DC_0^{-1}D',$$

where C_0 is the original-scale Henderson coefficient matrix defined in `postVarU`. The stored system `C` is solved against contrast transposes and the result multiplied by `Cscale`. Arbitrary contrasts work regardless of `computeCi`; no stored `Ci` is used.

With $Q = (C_0^{-1})_{uu}$ and $B = (C_0^{-1})_{bb}$, the new outputs are

$$\mathrm{VarU} = L_u(G - Q)L_u',$$

$$\mathrm{sampling.vcov} = L_bBL_b' + L_u(G - Q)L_u'.$$

The BLUE and BLUP have zero sampling cross-covariance, whereas the fixed/random blocks of the inverse Henderson matrix describe error covariance and generally are not zero. In particular, subtracting the full target PEV from L_uGL_u' is incorrect if fixed effects are included. A fixed-only prediction has zero `VarU`, but generally nonzero `sampling.vcov` and PEV. Random-only predictions satisfy $L_uGL_u' = \mathrm{PEV} + \mathrm{VarU}$.

The implementation batches full, random-only and fixed-only contrasts into one Henderson factorization. Prior products use the term structure $G_k = \Sigma_k \otimes A_k$ and sparse relationship-precision solves. Neither the observation covariance V nor its inverse is constructed. Additional matrices require quadratic storage in the number of predictions. Rotation and factor-score mappings operate in public effect coordinates. New fits retain relationship precisions; older ordinary fits attempt model input reconstruction as described in `postVarU`.

All covariance outputs use the fitted variance components as fixed, omitting their estimation uncertainty. They are exact for known components and plug-in approximations for estimated components, for both ML and REML. SEDs, pairwise tests and `std.error` retain their PEV-based meaning; enabling VarU does not change those calculations. Direct and matrix-free `solveOnly` fits lack the stored coefficient system required here.

Value

`pvals`	the table of predictions according to the specified arguments.
`vcov`	The full target prediction-error covariance, always returned.
`PEV`	When requested, identical to `vcov`; its diagonal is `pvals$pev`.
`VarU`	When requested, sampling covariance of the predicted random contribution only; its diagonal is `pvals$var.u`.
`sampling.vcov`	When `VarU=TRUE`, sampling covariance of the entire fixed-plus-random predictor. This is not the covariance used for standard errors.
`D`	the model matrix for predictions as defined in Welham et al.(2004).
`Dtable`	the compact table specifying the terms to include and terms to be averaged.
`levels`	the named list of level selections and resolved numeric settings used to construct D. Numeric defaults on the regular prediction path are retained-data means. With `observed=TRUE`, fixed numeric contributions remain empirical within-group averages; numeric overrides require explicit estimable contrasts.

observed	For observed=TRUE, a list containing the classify values, retained observation counts per prediction, and a description of the empirical weighting and selected random terms. The returned Dtable records the resolved random flags; levels stores level restrictions separately.
sed, avsed	when sed=TRUE: the standard errors of differences, sqrt(v_ii + v_jj - 2 v_ij) from vcov, and their mean, minimum and maximum.
pairwise	when requested: a data frame with level1, level2, difference (level1 - level2), SED, statistic, df, p.value and p.adjusted.

Author(s)

Giovanny Covarrubias-Pazaran

References

Welham, S., Cullis, B., Gogel, B., Gilmour, A., and Thompson, R. (2004). Prediction in linear mixed models. Australian and New Zealand Journal of Statistics, 46, 325 - 347.

See Also

predict, mmes

Examples

```
data(DT_yatesoats, package="enhancer")
DT <- DT_yatesoats
m3 <- mmes(fixed=Y ~ V + N + V:N ,
           random = ~ B + B:MP,
           rcov=~units,
           data = DT)

m3 <- postPEV(m3, mode = 2)

############################
## predict means for nitrogen
############################
Dt <- m3$Dtable; Dt
# first fixed effect just average
Dt[1,"average"] = TRUE
# second fixed effect include
Dt[2,"include"] = TRUE
# third fixed effect include and average
Dt[3,"include"] = TRUE
Dt[3,"average"] = TRUE
Dt

pp=predict(object=m3, Dtable=Dt, D="N")
pp$pvals

############################
## predict means for variety
```

```
##############################

Dt <- m3$Dtable; Dt
# first fixed effect include
Dt[1,"include"] = TRUE
# second fixed effect just average
Dt[2,"average"] = TRUE
# third fixed effect include and average
Dt[3,"include"] = TRUE
Dt[3,"average"] = TRUE
Dt

pp=predict(object=m3, Dtable=Dt, D="V")
pp$pvals

##############################
## predict means for nitrogen:variety
##############################
# prediction matrix D based on (equivalent to classify in asreml)
Dt <- m3$Dtable; Dt
# first fixed effect include and average
Dt[1,"include"] = TRUE
Dt[1,"average"] = TRUE
# second fixed effect include and average
Dt[2,"include"] = TRUE
Dt[2,"average"] = TRUE
# third fixed effect include and average
Dt[3,"include"] = TRUE
Dt[3,"average"] = TRUE
Dt

pp=predict(object=m3, Dtable=Dt, D="N:V")
pp$pvals

##############################
## select levels through the argument, keeping Dtable compact
##############################
Dt <- m3$Dtable
Dt$average[Dt$term %in% c("1", "V", "V:N")] <- TRUE
Dt$include[Dt$term %in% c("N", "V:N")] <- TRUE

p <- predict(m3, D="N", Dtable=Dt,
             levels=list(N=c("0.6", "0.2")))
p$pvals
p$levels[["N"]]

p2 <- predict(m3, D="N", Dtable=p$Dtable, levels=p$levels)
all.equal(p$D, p2$D)

## Covariances of the requested linear combinations
p <- predict(m3, D="N", Dtable=Dt, PEV=TRUE, VarU=TRUE)
p$PEV
p$VarU
```

```
p$sampling.vcov
randomColumns <- seq.int(nrow(m3$b)+1L, nrow(m3$bu))
randomD <- p$D[,randomColumns,drop=FALSE]
full <- postVarU(m3, mode=2)
all.equal(unname(p$VarU),
          unname(as.matrix(randomD %*% full$VarU %*% t(randomD))))
```

`print.sommer_mmes_control`

Print Modular Numerical Controls

Description

Prints the formulation and each named numerical-control module. Nested lists are summarized rather than expanded; long vectors are represented by their length. The method returns the control object invisibly.

Usage

```
## S3 method for class 'sommer_mmes_control'
print(x, ...)
```

Arguments

`x`	An object of class `sommer_mmes_control`.
`...`	Ignored.

`print.sommer_mme_benchmark`

Print method for sommer_mme_benchmark

Description

Print method for sommer_mme_benchmark

Usage

```
## S3 method for class 'sommer_mme_benchmark'
print(x, ...)
```

Arguments

`x`	An object of class `sommer_mme_benchmark`.
`...`	Ignored.

print.sommer_mme_crossover

Print method for sommer_mme_crossover

Description

Print method for sommer_mme_crossover

Usage

```
## S3 method for class 'sommer_mme_crossover'
print(x, ...)
```

Arguments

x	An object of class sommer_mme_crossover.
...	Ignored.

print.sommer_mme_profile

Print method for sommer_mme_profile

Description

Print method for sommer_mme_profile

Usage

```
## S3 method for class 'sommer_mme_profile'
print(x, ...)
```

Arguments

x	An object of class sommer_mme_profile.
...	Ignored.

profile_mme_workload *Profile an MME Workload by Phase*

Description

Fits one workload with phase profiling enabled and tabulates where the time goes.

Usage

```
profile_mme_workload(
  workload,
  engine = mmes_control(),
  nIters = 5L,
  dominantShare = 0.3
)
```

Arguments

workload	An mme_workload from generate_mme_workload.
engine	A mmes_control object; profiling is forced on.
nIters	Maximum REML/ML iterations. Default is 5.
dominantShare	Share of fit time above which a phase is flagged dominant. Default is 0.3.

Value

A list of class "sommer_mme_profile" with a per-phase table, workload counters, per-iteration seconds, and the total elapsed time.

r2 *Reliability*

Description

Calculates the reliability of BLUPs in a sommer model.

Usage

```
r2(object, object2=NULL)
```

Arguments

object	Model fitted with the mmes function.
object2	An optional model identical to object in the first argument but fitted with the argument returnParam set to TRUE to access the relationship matrices from the fitted model.

Details

The reliability method calculated is the classical animal model: R2=(G-PEV)/G

Value

result a list with as many elements as random effects fitted containing reliabilities for individual BLUPs.

References

Mrode, R. A. (2014). Linear models for the prediction of animal breeding values. Cabi.

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

See Also

`mmes` – the core function of the package

Examples

```
####============================================####
#### Example population
####============================================####
data(DT_example, package="enhancer")
DT <- DT_example
head(DT)
ans1 <- mmes(Yield~Env,
             random= ~ Name + Env:Name,
             rcov= ~ units,
             data=DT)
ans1 <- postPEV(ans1, mode = 1)
rel=r2(ans1)
```

---

randef *extracting random effects*

Description

This function is extracts the random effects from a mixed model fitted by mmer.

Usage

```
randef(object)
```

Arguments

`object`	an mmer object

Value

$randef a list structure with the random effects or BLUPs.

Examples

```
# randef(model)
```

`residuals.mmes` *Residuals form a GLMM fitted with mmes*

Description

`residuals` method for class `"mmes"`.

Usage

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

Arguments

`object`	an object of class `"mmes"`
`type`	For a non-Gaussian PQL fit, return response residuals (default), signed deviance residuals, Pearson residuals, or final IRLS working residuals. Deviance and Pearson residuals use the prior weights (binomial trials, see `binm`).
`...`	Further arguments to be passed

Value

For Gaussian fits, residuals of the form e = y - Xb - Zu. For non-Gaussian PQL fits, residuals on the requested scale.

Author(s)

Giovanny Covarrubias

See Also

`residuals`, `mmes`

rotate_back_mmes *Back-transform rotated mmes random effects*

Description

Back-transforms the selected relationship random effect from the solver eigenbasis to the original relationship-level basis.

Usage

```
rotate_back_mmes(object)
```

Arguments

object	An mmes model fitted with vsm(..., rotation=TRUE).

Details

If $Gu = U\Lambda U'$ and a denotes the solver-basis effect, the reported effect is $g = Ua$. Fitted mmes objects already expose this original-basis effect in uList; this function provides the explicit transformation from the retained engine output.

Value

A matrix of original-basis random effects, with relationship levels in rows and covariance-product coordinates in columns.

See Also

mmes, vsm

rrm *Reduced-rank covariance structure*

Description

Reduced-rank covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
rrm(x, k = 1L, loadings = NULL, fixed = NULL)
```

Arguments

x	Variable defining q covariance levels.
k	Reduced rank, satisfying 1 <= k < q.
loadings	Optional finite q by k starting loading matrix. The leading k by k block is made lower triangular and must have positive diagonal entries. Loadings are divided by their first diagonal entry to remove the internal scale.
fixed	Logical vector of length $qk - k(k-1)/2$, controlling the triangular loading entries in column-major order. The first entry is always fixed: the reference loading is one, and vsm() estimates the overall variance scale. Other fixed entries constrain relative loadings.

Details

The reduced-rank structure uses

$$K = \Lambda\Lambda^{\mathsf{T}}, \qquad \Lambda_{11} = 1,$$

and vsm() supplies the overall variance scale:

$$G = \sigma^2 K.$$

All specific variances are fixed at exactly zero; no diagonal remainder or numerical nugget is added. This is the same covariance family as ASReml's default rr(x, k) model. Individual loadings can differ by sign or rotation, so compare reconstructed covariance matrices when comparing fits.

The leading $k \times k$ loading block is lower triangular for rotational identification and its diagonal entries are positive. There are $qk - k(k-1)/2 - 1$ free shape parameters plus the overall scale. Because $k < q$, the random-effect covariance is positive semidefinite and singular by design.

The Henderson engine fits this structure through latent factors rather than inverting the singular environment covariance. The direct-inversion formulation (control$formulation="direct") also supports it, provided the total observation covariance is positive definite. rrm() is intended for random effects, not residual covariance structures.

Native latent-factor Henderson fits support control$solve$method="ldlt" and control$solve$method="cholmod" (including automatic selection). They use loading-aware likelihood scores and average-information updates in the original loading/variance coordinates. The loading derivatives account for the parameter-dependent latent design; the singular environment covariance is never inverted. control$proposal$stepWeight damps covariance-parameter proposals, while control$update$blendWeight blends EM/GEM or scalar PX-GEM with average-information update directions, selected through control$update$recovery, as in ordinary mmes() fitting. Parameter standard errors use the unblended average-information matrix at the final parameter values. Prediction errors are mapped back from the latent coefficient equations to the original random effects.

The likelihood trace terms are computed exactly by sharing random-design and residual-block contractions across covariance parameters. Bounded panels of Henderson solves avoid constructing a dense record-by-record covariance or projection matrix. Random trace solves use the loading span (rank k), rather than every environment direction. Reducing the coefficient dimension does not guarantee a faster fit for every dataset.

Section-specific spatial residuals supplied by dsumm() are evaluated on their observed residual blocks, including grids with missing cells and unlisted sections with independent residuals. Residual covariance derivatives and trace contractions never expand the full section-by-grid covariance.

`engineDiagnostics$residualEvaluation` reports `"observed-blocks"` and `engineDiagnostics$largestResidualBlo` records the largest observed block dimension used by the native RR AI evaluator.

Likelihood and AI calculations share the selected residual precision operator and the live coefficient factorization, including grid residuals with missing-cell absorption and eligible block-Schur solves. No separate coefficient factorization or dense observed-block precision is built for AI. Random trace panels use the latent Henderson identity to cancel relationship covariance actions. Grid residual traces use block-local projection panels rather than repeated full-observation precision applications. `engineDiagnostics$residualEngine` identifies the shared residual engine, and `sharedConditionalOperators` confirms its use. `aiElapsedSeconds` gives elapsed timings for the final AI evaluation's setup, random traces, residual traces, and information matrix.

The positive-specific-variance Woodbury Schur formulation used for `fam()` is unnecessary for exact RR: the specific-effect blocks are absent, not inverted at zero. The smaller latent coefficient system still uses the ordinary LDLT/CHOLMOD and eligible block-Schur dispatch. With `control$diagnostics$verbose=TRUE`, the native fit reports its RR path, selected coefficient solver, and then the progress table. Each accepted iteration reports likelihood, wall-clock time, cumulative CPU seconds, the scheduled `EMweight`, accepted-step `trustAlpha` and `lineAlpha`, and score norm. The step scales describe the update that reached the current row (both are one at initialization); `lineAlpha` includes `control$proposal$stepWeight` and any backtracking. Boundary-restraint and `notPD` columns are omitted because the latent optimizer does not use those ordinary covariance-update diagnostics. Numeric weights, step scales and CPU times are retained in `engineDiagnostics$iterationProgress`.

For native Henderson fits with weights `W`, let $U = \mathrm{chol}(W)$ in observation order. The effective residual precision is $UR_0^{-1}U^{\mathsf{T}}$, where R_0 is the unweighted residual covariance. The score and AI use derivatives of the corresponding covariance $U^{-\mathsf{T}}R_0U^{-1}$.

The native path does not currently support PCG, `vcc` constraints, covariance-update acceleration, `control$task$mode="solve"`, or `control$representation$factorScores` augmentation. These combinations produce explicit errors. The stored coefficient matrix `C` describes the latent equations; `coefficientMap` maps latent coefficients to the original effect space. Use `predict()`, `postPEV()`, and the fitted `uPevList` for prediction uncertainty rather than directly indexing `C` by original effect indices.

Earlier versions used $\Lambda\Lambda^{\mathsf{T}} + I_q$, estimating a nonzero common specific variance on the fitted scale. To fit that former covariance family, use `fam(x, k, variance="homogeneous")`.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`fam`, `ownm`, `vsm`.

Examples

```
## Not run:
vsm(rrm(environment, k=2), ism(genotype))
# Former rrm() model: an estimated common specific variance
vsm(fam(environment, k=2, variance="homogeneous"), ism(genotype))
```

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

`run_mme_benchmark` *Run Experimental MME Benchmark Suite*

Description

Executes a battery of workloads across candidate solver engines, measuring wall-clock time, iterations, peak memory (if available), active numerical paths, and asserting numerical parity.

Usage

```
run_mme_benchmark(
  workloads,
  engines = NULL,
  replications = 3L,
  verifyParity = TRUE,
  tolerance = 1e-05,
  nIters = 5L,
  verbose = TRUE
)
```

Arguments

`workloads`	A list of `mme_workload` objects (or a single workload).
`engines`	A named list of `mmes_control` objects. Defaults to comparing `baseline` (standard auto solver) against candidate experimental toggles.
`replications`	Number of timed repetitions (warmup run excluded). Default is 3.
`verifyParity`	Logical; whether to assert that experimental candidates match the reference engine in final likelihood, variance components, and BLUPs. Default is TRUE.
`tolerance`	Numerical tolerance for parity verification. Default is 1e-5.
`nIters`	Maximum REML/ML iterations for benchmark fits. Default is 5.
`verbose`	Logical; whether to print progress. Default is TRUE.

Value

An object of class `"sommer_mme_benchmark"`. Warmup and timed-run failures retain the same result columns with an `error` message. The `converged` column records the fitted model's `convergence` flag. `parity_ok` is `NA` when parity is not checked or the reference engine failed; numerical comparisons use final likelihoods and estimates, not the lengths of their optimization histories.

sar *Simultaneous autoregressive spatial covariance structure*

Description

Simultaneous autoregressive spatial covariance structure. The returned dimensionless covariance shape is intended for use inside vsm, which supplies the single overall variance scale.

Usage

```
sar(x, W, rho = 0.10, fixed = FALSE)
```

Arguments

x	Factor or design defining q spatial levels.
W	Square spatial weights matrix aligned to the levels of x. Row/column names are used when available.
rho	Starting SAR dependence parameter.
fixed	Logical indicating whether rho is fixed.

Details

The simultaneous autoregressive covariance is constructed from

$$B = I - \rho W, \qquad M = B^{-1}B^{-\mathsf{T}}, \qquad K = M/M_{11}.$$

For a general real weights matrix, rho is restricted to the conservative interval $(-1/r(W), 1/r(W))$, where $r(W)$ is the spectral radius. A bounded-logit working coordinate enforces the interval. The first covariance derivative is supplied analytically to the generic CovarianceFactor engine.

Value

A list containing the incidence/design matrix in Z and a compiled CovarianceFactor v2 descriptor in covFactor, for use inside vsm.

See Also

car, maternm, vsm.

Examples

```
## Not run:
vsm(sar(location, W), ism(genotype))

## End(Not run)
```

`scores_mmes` *Predict factor-analytic scores from a fitted mmes model*

Description

Predicts latent factor scores for each level of the main random-effect term crossed with a `fam()` or `rrm()` covariance-shaping factor, combining the fitted loadings (see `loadings_mmes`), the fitted covariance matrix, and the BLUPs already stored in `object$uList`.

Usage

```
scores_mmes(object, term = NULL, method = c("regression", "bartlett"),
            varianceScale=TRUE, rotation=TRUE)
```

Arguments

`object`	a fitted model of class `"mmes"`.
`term`	character name of the random term built with a single `fam()` or `rrm()` covariance-shaping factor. If `NULL`, the unique such term in the model is used automatically.
`method`	`"regression"` (default) uses the full fitted covariance Σ (Thomson's method); `"bartlett"` uses only the specific (residual) variances Ψ (Bartlett's classic unbiased estimator).
`varianceScale`	a logical argument to indicate if loadings should be returned in variance scale (multiplied by sqrt(sigma2)).
`rotation`	a logical value to indicate if loadings should be rotated by its singular vectors.

Details

Let U be the levels-by-levels BLUP matrix in `object$uList[[term]]` and Λ the loadings from `loadings_mmes`. The regression (Thomson) scores are

$$F = U\Sigma^{-1}\Lambda,$$

and the Bartlett scores are

$$F = U\Psi^{-1}\Lambda(\Lambda^{\mathsf{T}}\Psi^{-1}\Lambda)^{-1}.$$

Because U already contains shrunken (BLUP) effects, the resulting scores are similarly regularized. For positive-specific FA covariances, regression uses a Woodbury contraction and a rank-by-rank solve rather than a dense trait-by-trait solve. It also works when the conditional FA engine retains only factorized covariances.

For `rrm()`, the covariance is singular and the specific variances are zero. Regression scores instead recover the latent BLUP coordinates using

$$F = U\Lambda(\Lambda^{\mathsf{T}}\Lambda)^{-1},$$

so that $F\Lambda^{\mathsf{T}} = U$. This uses the returned loading scale and rotation. Bartlett scores are undefined for this zero-specific-variance model and produce an informative error.

Value

A matrix with one row per level of the main random-effect term and one column per latent factor.

See Also

loadings_mmes, fam, rrm, vsm, mmes.

Examples

```
## Not run:
mix <- mmes(BLUEs ~ trial,
            random = ~ vsm(fam(trial, 2), ism(genotype)),
            rcov = ~ units, data = dt)
scores_mmes(mix)

## End(Not run)
```

---

spl2Dc *Two-dimensional penalised tensor-product of marginal B-Spline basis.*

Description

Auxiliary function used for modelling the spatial or environmental effect as a two-dimensional penalised tensor-product (isotropic approach) based on Lee et al. (2013) and Rodriguez-Alvarez et al. (2018). This is a modified wrapper of some portions of the SpATS package to build a single incidence matrix containing all the columns from tensor products of the x and y coordinates and it fits such matrix as a single random effect. Then the heterogeneous covariances structure capabilities of sommer can be used to enhance the model fit. You may be interested in reading and citing not only sommec but also Wageningen publications if using this 2D spline methodology.

Usage

```
spl2Dc(x.coord,y.coord,at.var=NULL,at.levels=NULL, type="PSANOVA",
      nsegments = c(10,10), penaltyord = c(2,2), degree = c(3,3),
      nestorder = c(1,1), thetaC=NULL, theta=NULL, sp=FALSE)
```

Arguments

x.coord	vector of coordinates on the x-axis direction (i.e. row) to use in the 2 dimensional spline.
y.coord	vector of coordinates on the y-axis direction (i.e. range or column) to use in the 2 dimensional spline.
at.var	vector of indication variable where heterogeneous variance is required (e.g., a different spl2D for each field).
at.levels	character vector with the names of the leves for the at term that should be used, if missing all levels are used.

type	one of the two methods "PSANOVA" or "SAP". See details below.
nsegments	numerical vector of length 2 containing the number of segments for each marginal (strictly nsegments - 1 is the number of internal knots in the domain of the covariate). Atomic values are also valid, being recycled. Default set to 10.
penaltyord	numerical vector of length 2 containing the penalty order for each marginal. Atomic values are also valid, being recycled. Default set to 2 (second order). Currently, only second order penalties are allowed.
degree	numerical vector of length 2 containing the order of the polynomial of the B-spline basis for each marginal. Atomic values are also valid, being recycled. Default set to 3 (cubic B-splines).
nestorder	numerical vector of length 2 containing the divisor of the number of segments (nsegments) to be used for the construction of the nested B-spline basis for the smooth-by-smooth interaction component. In this case, the nested B-spline basis will be constructed assuming a total of nsegments/nestorder segments. Default set to 1, which implies that nested basis are not used. See SAP for more details.
thetaC	an optional matrix for constraints in the variance components.
theta	an optional matrix for initial values of the variance components.
sp	a TRUE/FALSE statement to indicate if the VC from this structure should be multiplied by the scale parameter added in the mmes function through the addScaleParam argument in the mmes function .

Details

The following documentation is taken from the SpATS package. Please refer to this package and associated publications if you are interested in going deeper on this technique:

Within the P-spline framework, anisotropic low-rank tensor-product smoothers have become the general approach for modelling multidimensional surfaces (Eilers and Marx 2003; Wood 2006). In the original SpATS package, was proposed to model the spatial or environmental effect by means of the tensor-product of B-splines basis functions. In other words, was proposed to model the spatial trend as a smooth bivariate surface jointly defined over the the spatial coordinates. Accordingly, the current function has been designed to allow the user to specify the spatial coordinates that the spatial trend is a function of. There is no restriction about how the spatial coordinates shall be specified: these can be the longitude and latitude of the position of the plot on the field or the column and row numbers. The only restriction is that the variables defining the spatial coordinates should be numeric (in contrast to factors).

As far as estimation is concerned, we have used in this package the equivalence between P-splines and linear mixed models (Currie and Durban, 2002). Under this approach, the smoothing parameters are expressed as the ratio between variance components. Moreover, the smooth components are decomposed in two parts: one which is not penalised (and treated as fixed) and one with is penalised (and treated as random). For the two-dimensional case, the mixed model representation leads also to a very interesting decomposition of the penalised part of the bivariate surface in three different components (Lee and Durban, 2011): (a) a component that contains the smooth main effect (smooth trend) along one of the covariates that the surface is a function of (as, e.g, the x-spatial coordinate or column position of the plot in the field), (b) a component that contains the smooth main effect (smooth trend) along the other covariate (i.e., the y-spatial coordinate or row position);

and (c) a smooth interaction component (sum of the linear-by-smooth interaction components and the smooth-by-smooth interaction component).

The original implementation of SpATS assumes two different smoothing parameters, i.e., one for each covariate in the smooth component. Accordingly, the same smoothing parameters are used for both, the main effects and the smooth interaction. However, this approach can be extended to deal with the ANOVA-type decomposition presented in Lee and Durban (2011). In their approach, four different smoothing parameters are considered for the smooth surface, that are in concordance with the aforementioned decomposition: (a) two smoothing parameter, one for each of the main effects; and (b) two smoothing parameter for the smooth interaction component.

It should be noted that, the computational burden associated with the estimation of the two-dimensional tensor-product smoother might be prohibitive if the dimension of the marginal bases is large. In these cases, Lee et al. (2013) propose to reduce the computational cost by using nested bases. The idea is to reduce the dimension of the marginal bases (and therefore the associated number of parameters to be estimated), but only for the smooth-by-smooth interaction component. As pointed out by the authors, this simplification can be justified by the fact that the main effects would in fact explain most of the structure (or spatial trend) presented in the data, and so a less rich representation of the smooth-by-smooth interaction component could be needed. In order to ensure that the reduced bivariate surface is in fact nested to the model including only the main effects, Lee et al. (2013) show that the number of segments used for the nested basis should be a divisor of the number of segments used in the original basis (`nsegments` argument). In the present function, the divisor of the number of segments is specified through the argument `nestorder`. For a more detailed review on this topic, see Lee (2010) and Lee et al. (2013). The "PSANOVA" approach represents an alternative method. In this case, the smooth bivariate surface (or spatial trend) is decomposed in five different components each of them depending on a single smoothing parameter (see Lee et al., 2013).

As mentioned at the beginning, the piece of documentation stated above was taken completely from the SpATS package in order to provide a deeper explanation. In practice, sommec uses some pieces of code from SpATS to build the design matrix containing all the columns from tensor products of the x and y coordinates and it fits such matrix as a single random effect. As a result the same variance component is assumed for the linear, linear by linear, linear by spline, and spline by spline interactions. This results in a less flexible approach than the one proposed by Rodriguez-Alvarez et al. (2018) but still makes a pretty good job to model the spatial variation. Use under your own risk.

References

Rodriguez-Alvarez, M.X, Boer, M.P., van Eeuwijk, F.A., and Eilers, P.H.C. (2018). SpATS: Spatial Analysis of Field Trials with Splines. R package version 1.0-9. https://CRAN.R-project.org/package=SpATS.

Rodriguez-Alvarez, M.X., et al. (2015) Fast smoothng parameter separaton n multdmensonal generalzed P-splnes: the SAP algorthm. Statistics and Computing 25.5: 941-957.

Lee, D.-J., Durban, M., and Eilers, P.H.C. (2013). Efficient two-dimensional smoothing with P-spline ANOVA mixed models and nested bases. Computational Statistics and Data Analysis, 61, 22 - 37.

Gilmour, A.R., Cullis, B.R., and Verbyla, A.P. (1997). Accounting for Natural and Extraneous Variation in the Analysis of Field Experiments. Journal of Agricultural, Biological, and Environmental Statistics, 2, 269 - 293.

See Also

mmes

Examples

```
## ============================ ##
## example to use spl2Dc()
## ============================ ##
data(DT_cpdata, package="enhancer")
# DT <- DT_cpdata
# GT <- GT_cpdata
# MP <- MP_cpdata
# A <- A.mat(GT)
## ============================ ##
## mimic 2 fields
## ============================ ##
# aa <- DT; bb <- DT
# aa$FIELD <- "A";bb$FIELD <- "B"
# set.seed(1234)
# aa$Yield <- aa$Yield + rnorm(length(aa$Yield),0,4)
# DT2 <- rbind(aa,bb)
# head(DT2)
# mix <- mmes(Yield~1, control = mmes_control(formulation = "direct"),
#             random=~
#               vsm(dsm(FIELD),ism(Rowf)) +
#               vsm(dsm(FIELD),ism(Colf)) +
#                 spl2Dc(Row,Col,at.var=FIELD),
#             rcov=~vsm(dsm(FIELD),ism(units)),
#             data=DT2)
#
# # extract spatial effects
# blup <- mix$uList$`spl2Dc(Row, Col, at.var = FIELD`
# head(blup) # 2 fields
# # recreate the incidence matrices
# xx=with(DT2, spl2Dc(Row,Col,at.var=FIELD))
# # get fitted values Zu for spatial effects and add them to the dataset
# field1 <- xx$Z$`A:all` %*% blup[,1]
# field2 <- xx$Z$`B:all` %*% blup[,2]
# DT2$spat <- field1+field2
# # plots the spatial effects
# lattice::levelplot(spat~Row*Col|FIELD, data=DT2)
```

---

spl2Dmats *Get Tensor Product Spline Mixed Model Incidence Matrices*

Description

spl2Dmats gets Tensor-Product P-Spline Mixed Model Incidence Matrices for use with sommer and its main function mmes. We thank Sue Welham for making the TPSbits package available to the

community. If you're using this function for your research please cite her TPSbits package :) this is mostly a wrapper of her tpsmmb function to enable the use in sommer.

Usage

```
spl2Dmats(
  x.coord.name,
  y.coord.name,
  data,
  at.name,
  at.levels,
  nsegments=NULL,
  minbound=NULL,
  maxbound=NULL,
  degree = c(3, 3),
  penaltyord = c(2,2),
  nestorder = c(1,1),
  method = "Lee"
)
```

Arguments

x.coord.name	A string. Gives the name of data element holding column locations.
y.coord.name	A string. Gives the name of data element holding row locations.
data	A dataframe. Holds the dataset to be used for fitting.
at.name	name of a variable defining if the 2D spline matrices should be created at different units (e.g., at different environments).
at.levels	a vector of names indicating which levels of the at.name variable should be used for fitting the 2D spline function.
nsegments	A list of length 2. Number of segments to split column and row ranges into, respectively (= number of internal knots + 1). If only one number is specified, that value is used in both dimensions. If not specified, (number of unique values - 1) is used in each dimension; for a grid layout (equal spacing) this gives a knot at each data value.
minbound	A list of length 2. The lower bound to be used for column and row dimensions respectively; default calculated as the minimum value for each dimension.
maxbound	A list of length 2. The upper bound to be used for column and row dimensions respectively; default calculated as the maximum value for each dimension.
degree	A list of length 2. The degree of polynomial spline to be used for column and row dimensions respectively; default=3.
penaltyord	A list of length 2. The order of differencing for column and row dimensions, respectively; default=2.
nestorder	A list of length 2. The order of nesting for column and row dimensions, respectively; default=1 (no nesting). A value of 2 generates a spline with half the number of segments in that dimension, etc. The number of segments in each direction must be a multiple of the order of nesting.

`method` A string. Method for forming the penalty; default="Lee" ie the penalty from Lee, Durban & Eilers (2013, CSDA 61, 22-37). The alternative method is "Wood" ie. the method from Wood et al (2012, Stat Comp 23, 341-360). This option is a research tool and requires further investigation.

Value

List of length 7 elements:

1. `data` = the input data frame augmented with structures required to fit tensor product splines in `asreml-R`. This data frame can be used to fit the TPS model.

   Added columns:

   - `TP.col`, `TP.row` = column and row coordinates
   - `TP.CxR` = combined index for use with smooth x smooth term
   - `TP.C.n` for n=1:(diff.c) = X parts of column spline for use in random model (where diff.c is the order of column differencing)
   - `TP.R.n` for n=1:(diff.r) = X parts of row spline for use in random model (where diff.r is the order of row differencing)
   - `TP.CR.n` for n=1:((diff.c*diff.r)) = interaction between the two X parts for use in fixed model. The first variate is a constant term which should be omitted from the model when the constant (1) is present. If all elements are included in the model then the constant term should be omitted, eg. `y ~ -1 + TP.CR.1 + TP.CR.2 + TP.CR.3 + TP.CR.4 + other terms...`
   - when `asreml="grp"` or "sepgrp", the spline basis functions are also added into the data frame. Column numbers for each term are given in the `grp` list structure.
2. `fR` = Xr1:Zc
3. `fC` = Xr2:Zc
4. `fR.C` = Zr:Xc1
5. `R.fC` = Zr:Xc2
6. `fR.fC` = Zc:Zr
7. `all` = Xr1:Zc | Xr2:Zc | Zr:Xc1 | Zr:Xc2 | Zc:Zr

Examples

```
data("DT_cpdata", package="enhancer")
DT <- DT_cpdata
GT <- GT_cpdata
MP <- MP_cpdata
#### create the variance-covariance matrix
A <- A.mat(GT) # additive relationship matrix

M <- spl2Dmats(x.coord.name = "Col", y.coord.name = "Row", data=DT, nseg =c(14,21))
head(M$data)
# m1g <- mmes(Yield~1+TP.CR.2+TP.CR.3+TP.CR.4,
#             random=~Rowf+Colf+vsm(ism(M$fC))+vsm(ism(M$fR))+
#               vsm(ism(M$fC.R))+vsm(ism(M$C.fR))+vsm(ism(M$fC.fR))+
#               vsm(ism(id),Gu=A),
```

```
#             data=M$data, tolpar = 1e-6,
#             iters=30)
#
# summary(m1g)$varcomp
```

`stackTraits` *Stack several trait columns into the long format used by multi-trait models*

Description

Converts a wide data frame (one column per trait) into the long format used by multi-trait `mmes` models: one row per record and trait, a factor identifying the trait, a single response column and a record key that pairs the traits measured on the same experimental unit.

Usage

```
stackTraits(data, traits, keep = NULL, trait = "trait", value = "value",
            record = "record")
```

Arguments

`data`	A data frame in wide format.
`traits`	Character vector with the names of the trait columns to stack. Their order defines the levels of the trait factor.
`keep`	Columns of `data` copied to every stacked row. By default, all columns that are not in `traits`.
`trait`	Name of the new trait factor column.
`value`	Name of the new response column.
`record`	Name of the new record-key factor column.

Details

Rows are stacked trait by trait, so row i of trait t is row $(t-1)n+i$ of the output. The record key is taken from the row names of `data` when these are unique and not the default `1..n`; otherwise it is `"r1"`, `"r2"`, ... Missing trait values are kept as `NA` (they are handled by the `naMethodY` rules of `mmes`, so incomplete records are allowed).

Using the record key in the residual, `rcov = ~ vsm(usm(trait), ism(record))`, pairs the residuals of the traits of each unit explicitly. With `ism(units)` the pairing is implied by the order of the rows within each trait instead.

Value

A data frame with the `keep` columns and the `record`, `trait` and `value` columns, with `length(traits) * nrow(data)` rows.

See Also

mmes, covmatrix_mmes, familym

Examples

```
set.seed(17)
n <- 60
DT <- data.frame(id = factor(rep(seq_len(n), each = 3)))
u <- matrix(rnorm(n * 2), n)
DT$trait_a <- u[DT$id, 1] + rnorm(nrow(DT), sd = 0.7)
DT$trait_b <- u[DT$id, 2] + rnorm(nrow(DT), sd = 1)
DTL <- stackTraits(DT, traits = c("trait_a", "trait_b"), keep = "id")
head(DTL)
table(DTL$trait)

# bivariate model with the residual paired through the record key
fit <- mmes(value ~ trait,
  random = ~ vsm(usm(trait), ism(id)),
  rcov = ~ vsm(usm(trait), ism(record)),
  data = DTL,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
covmatrix_mmes(fit, 1)$correlation   # genetic correlation
```

---

strm *Covariance structure spanning several random terms*

Description

Combines two or more random terms built with vsm into one random structure with an estimated covariance between the terms. Typical uses are direct-maternal animal models, correlated permanent-environment effects and indirect genetic effects.

Usage

```
strm(..., cov = usm, Gu = NULL, labels = NULL, sigma2 = NULL,
     fixedSigma2 = FALSE)
```

Arguments

...	Two or more random vsm() terms, optionally named (names become the term labels), e.g. dir = vsm(ism(id)), mat = vsm(ism(dam)).
cov	A covariance constructor applied to the term index, e.g. usm (default), dsm, csm, corgm, or function(x) fam(x, k = 1).
Gu	Precision matrix shared by all terms (attr(Gu, "inverse") = TRUE). If NULL, a Gu given inside the terms is used (all terms must then use the same one) or an identity over the union of the terms' levels.
labels	Term labels; default the argument names or t1, t2, ...
sigma2	Starting overall variance scale; NULL lets mmes choose.
fixedSigma2	Fix the overall variance scale.

Details

For terms $u_1, \ldots, u_T$ with incidence matrices $Z_1, \ldots, Z_T$ the structure is

$$\mathrm{Var}(u_1, \ldots, u_T) = \sigma^2 \, K_{terms} \otimes K_{inner} \otimes A,$$

where K_{terms} is given by `cov`, K_{inner} is the Kronecker product of the covariance factors used inside the terms (e.g. `dsm(env)`), and A is the relationship matrix. The requirements are those of a direct product: all terms share the coefficient levels (their levels are aligned on the union, or on the levels of `Gu`), the same `Gu`, and the same inner covariance constructors with the same levels; the parameters of the inner factors are shared. The `sigma2` of individual terms is ignored. BLUPs are returned in one `uList` element with a column per term (and inner level). `covm(ran1, ran2)` is the two-term special case with `cov = usm`.

The terms do not need to be adjacent in the model formula and their order is checked by level names rather than assumed.

Value

A random structure for the `random` formula of `mmes`.

See Also

`covm`, `vsm`, `usm`, `mmes`

Examples

```
set.seed(3)
ids <- paste0("i", 1:30)
A <- diag(30) + diag(0.2, 30); dimnames(A) <- list(ids, ids)
Ai <- solve(A); attr(Ai, "inverse") <- TRUE
d <- data.frame(id = factor(sample(ids, 200, TRUE), levels = ids),
                dam = factor(sample(ids[1:15], 200, TRUE), levels = ids))
d$y <- rnorm(200)
fit <- mmes(y ~ 1,
  random = ~ strm(dir = vsm(ism(id)), mat = vsm(ism(dam)), Gu = Ai),
  data = d,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
covparams_mmes(fit, 1)
```

---

summary.mmer *summary form a GLMM fitted with mmer*

Description

`summary` method for class `"mmer"`.

Usage

```
## S3 method for class 'mmer'
summary(object, ...)
```

Arguments

object	an object of class "mmer"
...	Further arguments to be passed

Value

vector of summary

Author(s)

Giovanny Covarrubias-Pazaran

See Also

summary, mmer

---

summary.mmes *summary form a GLMM fitted with mmes*

Description

summary method for class "mmes".

Usage

```
## S3 method for class 'mmes'
summary(object, ...)
```

Arguments

object	an object of class "mmes"
...	Further arguments to be passed

Value

vector of summary

Author(s)

Giovanny Covarrubias-Pazaran

See Also

summary, mmes

`toeplitzm` *General positive-definite Toeplitz correlation structure*

Description

General positive-definite Toeplitz correlation structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
toeplitzm(x, pacf = NULL, fixed = NULL)
```

Arguments

`x`	Ordered factor or design defining q covariance levels.
`pacf`	Optional q-1 starting reflection coefficients / partial autocorrelations. Defaults to 0.10.
`fixed`	Logical vector of length q-1.

Details

A full q by q Toeplitz correlation matrix is parameterized by q-1 reflection coefficients. Each PACF is mapped from an unconstrained working coordinate with `tanh`; the reflection coefficients are converted to a stable autoregressive representation and then to the implied Toeplitz correlation sequence. This parameterization preserves positive definiteness while allowing every lag correlation to vary indirectly.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`ar1m`, `ar2m`, `ar3m`, `mam`, `vsm`.

Examples

```
## Not run:
vsm(toeplitzm(time), ism(id))

## End(Not run)
```

`tpsmmbwrapper` *Get Tensor Product Spline Mixed Model Incidence Matrices*

Description

`tpsmmbwrapper` is a wrapper of tpsmmb function from the TPSbits package to avoid version dependencies but if you're using this function for your research please cite the TPSbits package. This function is internally used by the `spl2Dmatrices` function to get Tensor-Product P-Spline Mixed Model Bits (design matrices) for use with `sommer`.

Usage

```
tpsmmbwrapper(
  columncoordinates,
  rowcoordinates,
  data,
  nsegments=NULL,
  minbound=NULL,
  maxbound=NULL,
  degree = c(3, 3),
  penaltyord = c(2, 2),
  nestorder = c(1, 1),
  asreml = "mbf",
  eigenvalues = "include",
  method = "Lee",
  stub = NULL
)
```

Arguments

`columncoordinates`	A string. Gives the name of `data` element holding column locations.
`rowcoordinates`	A string. Gives the name of `data` element holding row locations.
`data`	A dataframe. Holds the dataset to be used for fitting.
`nsegments`	A list of length 2. Number of segments to split column and row ranges into, respectively (= number of internal knots + 1). If only one number is specified, that value is used in both dimensions. If not specified, (number of unique values - 1) is used in each dimension; for a grid layout (equal spacing) this gives a knot at each data value.
`minbound`	A list of length 2. The lower bound to be used for column and row dimensions respectively; default calculated as the minimum value for each dimension.
`maxbound`	A list of length 2. The upper bound to be used for column and row dimensions respectively; default calculated as the maximum value for each dimension.
`degree`	A list of length 2. The degree of polynomial spline to be used for column and row dimensions respectively; default=3.

`penaltyord` A list of length 2. The order of differencing for column and row dimensions, respectively; default=2.

`nestorder` A list of length 2. The order of nesting for column and row dimensions, respectively; default=1 (no nesting). A value of 2 generates a spline with half the number of segments in that dimension, etc. The number of segments in each direction must be a multiple of the order of nesting.

`asreml` A string. Indicates the types of structures to be generated for use in asreml models; default `"mbf"`. The appropriate eigenvalue scaling is included within the Z matrices unless setting `scaling="none"` is used, and then the scaling factors are supplied separately in the returned object.

- `asreml="mbf"` indicates the function should put the spline design matrices into structures for use with `"mbf"`;
- `asreml="grp"` indicates the function should add the composite spline design matrices (eg. for second-order differencing, matrices Xr1:Zc, Xr2:Zc, Zr:Xc1, Zr:Xc2 and Zc:Zr) into the data frame and provide a group list structure for each term;
- `asreml="sepgrp"` indicates the function should generate the individual X and Z spline design matrices separately (ie. Xc, Xr, Zc and Zr), plus the smooth x smooth interaction term as a whole (ie. Zc:Zr), and provide a group list structure for each term.
- `asreml="own"` indicates the function should generate the composite matrix (Xr:Zc | Zr:Xc | Zc:Zr) as a single set of columns.

`eigenvalues` A string. Indicates whether eigenvalues should be included within the Z design matrices `eigenvalues="include"`, or whether this scaling should be omitted (`eigenvalues="omit"`); default `eigenvalues="include"`. If the eigenvalue scaling is omitted from the Z design matrices, then it should instead be included in the model as a variance structure to obtain the correct TPspline model.

`method` A string. Method for forming the penalty; default=`"Lee"` ie the penalty from Lee, Durban & Eilers (2013, CSDA 61, 22-37). The alternative method is `"Wood"` ie. the method from Wood et al (2012, Stat Comp 23, 341-360). This option is a research tool and requires further investigation.

`stub` A string. Stub to be attached to names in the `mbf` list to avoid over-writing structures and general confusion.

Value

List of length 7, 8 or 9 (according to the `asreml` and `eigenvalues` parameter settings).

1. `data` = the input data frame augmented with structures required to fit tensor product splines in `asreml-R`. This data frame can be used to fit the TPS model.

   Added columns:

   - `TP.col`, `TP.row` = column and row coordinates
   - `TP.CxR` = combined index for use with smooth x smooth term
   - `TP.C.n` for n=1:(diff.c) = X parts of column spline for use in random model (where diff.c is the order of column differencing)

- TP.R.n for n=1:(diff.r) = X parts of row spline for use in random model (where diff.r is the order of row differencing)
- TP.CR.n for n=1:((diff.c*diff.r)) = interaction between the two X parts for use in fixed model. The first variate is a constant term which should be omitted from the model when the constant (1) is present. If all elements are included in the model then the constant term should be omitted, eg. y ~ -1 + TP.CR.1 + TP.CR.2 + TP.CR.3 + TP.CR.4 + other terms...
- when asreml="grp" or "sepgrp", the spline basis functions are also added into the data frame. Column numbers for each term are given in the grp list structure.

2. mbflist = list that can be used in call to asreml (so long as Z matrix data frames extracted with right names, eg BcZ<stub>.df)
3. BcZ.df = mbf data frame mapping onto smooth part of column spline, last column (labelled TP.col) gives column index
4. BrZ.df = mbf data frame mapping onto smooth part of row spline, last column (labelled TP.row) gives row index
5. BcrZ.df = mbf data frame mapping onto smooth x smooth term, last column (labelled TP.CxR) maps onto col x row combined index
6. dim = list structure, holding dimension values relating to the model:
   (a) "diff.c" = order of differencing used in column dimension
   (b) "nbc" = number of random basis functions in column dimension
   (c) "nbcn" = number of nested random basis functions in column dimension used in smooth x smooth term
   (d) "diff.r" = order of differencing used in column dimension
   (e) "nbr" = number of random basis functions in column dimension
   (f) "nbrn" = number of nested random basis functions in column dimension used in smooth x smooth term
7. trace = list of trace values for ZGZ' for the random TPspline terms, where Z is the design matrix and G is the known diagonal variance matrix derived from eigenvalues. This can be used to rescale the spline design matrix (or equivalently variance components).
8. grp = list structure, only added for settings asreml="grp", asreml="sepgrp" or asreml="own". For asreml="grp", provides column indexes for each of the 5 random components of the 2D splines. For asreml="sepgrp", provides column indexes for each of the X and Z component matrices for the 1D splines, plus the composite smooth x smooth interaction term. For asreml="own", provides column indexes for the composite random model. Dimensions of the components can be derived from the values in the dim item. The Z terms are scaled by the associated eigenvalues when eigenvalues="include", but not when eigenvalues="omit".
9. eigen = list structure, only added for option setting eigenvalues="omit". Holds the diagonal elements of the inverse variance matrix for the terms Xc:Zr (called diagr), Zc:Xr (called diagc) and Zc:Zr (called diagcr).

unsm *unstructured indication matrix*

Description

unsm creates a square matrix with ones in the diagonals and 2's in the off-diagonals to quickly specify an unstructured constraint in the Gtc argument of the vsm function.

Usage

```
unsm(x, reps=NULL)
```

Arguments

x	integer specifying the number of traits to be fitted for a given random effect.
reps	integer specifying the number of times the matrix should be repeated in a list format to provide easily the constraints in complex models that use the ds(), us() or cs() structures.

Value

$res a matrix or a list of matrices with the constraints to be provided in the Gtc argument of the vsm function.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Examples

```
unsm(3)
unsm(3,2)
```

usm *Unstructured positive-definite covariance structure*

Description

Unstructured positive-definite covariance structure. The returned dimensionless covariance shape is intended for use inside `vsm`, which supplies the single overall variance scale.

Usage

```
usm(x, theta = NULL, fixed = NULL)
```

Arguments

`x`	Variable or design defining q covariance levels.
`theta`	Optional positive-definite q by q starting covariance matrix.
`fixed`	Logical vector of length q(q+1)/2 - 1 controlling the estimable normalized-Cholesky parameters.

Details

The covariance shape is parameterized as

$$K = LL^{\mathsf{T}},$$

with lower-triangular L and $L_{11} = 1$. Diagonal elements after the first are positive and represented on log scales; lower off-diagonal elements are unrestricted. This guarantees positive definiteness without repeatedly repairing an unconstrained covariance matrix.

Value

A list containing the incidence/design matrix in `Z` and a compiled CovarianceFactor v2 descriptor in `covFactor`, for use inside `vsm`.

See Also

`vsm`, `mmes`.

Examples

```
## Not run:
vsm(usm(environment), ism(genotype))

## End(Not run)
```

usr *Unstructured covariance structure for* mmer *and* vsr

Description

Creates an unstructured variance-covariance component pattern for use by the mmer covariance-model interface, typically inside vsr.

Usage

```
usr(x)
```

Arguments

x	A factor, character vector, numeric vector, or design/incidence matrix defining the covariance dimension. Factor and character inputs are expanded to incidence columns. A matrix is used directly.

Details

usr() is part of the covariance-structure interface used by mmer/vsr; it is distinct from usm, which belongs to the newer mmes/vsm CovarianceFactor interface.

After constructing the design matrix Z, the function calls unsm(q), where q is the number of columns of Z, to create the thetaC pattern for an unstructured covariance matrix. Thus the associated covariance model permits a separate variance for every level and a separate covariance for every pair of levels:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1q} \\ \sigma_{12} & \sigma_2^2 & \cdots & \sigma_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{1q} & \sigma_{2q} & \cdots & \sigma_q^2 \end{bmatrix}.$$

The exact component coding is stored in the matrix returned by unsm(q) and passed to the vsr machinery through thetaC.

For a factor or character vector, missing observations are retained through the construction of the incidence matrix. For a numeric non-factor input, x is treated as a single design column.

Value

A list with components:

- Z: the design/incidence matrix associated with the covariance dimension.
- thetaC: the unstructured covariance-component pattern produced by unsm().

See Also

vsr, dsr, csr, atr, usm, mmer

Examples

```
# Unstructured covariance pattern across three levels
U <- usr(factor(c("A","B","A","C")))
U$Z
U$thetaC
```

---

vcc *Equality and scaling constraints between variance parameters*

Description

The vcc argument of mmes constrains groups of covariance parameters to be equal or to keep fixed ratios. Parameters are identified through the vcParams table.

Details

Parameters are estimated on transformed scales that guarantee valid covariance matrices (log for variances, atanh or bounded logits for correlations). A constraint is enforced exactly when it is affine on that scale, which covers equalities of any parameter and ratios of positive parameters (a log-scale offset). Relations such as $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$ are not affine on the log scale and are not supported.

Internally the free working parameters are written as $w = o + T\phi$. Each average-information update is projected onto this set in the information metric, which equals the constrained Newton step, and then repaired exactly; both the Henderson and the direct engines support constraints. The covariance of the estimates is $T(T^\top \mathcal{I} T)^{-1} T^\top$, so constrained parameters share their standard errors.

The vcParams table

`mmes(..., control = mmes_control(output = list(setup = TRUE)))$vcParams` (before fitting) and `fit$vcParams` (after fitting) list every covariance parameter with columns `index`, `term`, `parameter`, `kind`, `start` (natural scale), `free`, `structure` and `position` (and `group` when constraints were used). `kind` is `"scale"` for the variance scale (`sigma2`) owned by each `vsm()` term, or the transform used to estimate the parameter: `"exp"` (positive quantities such as ranges and variance ratios), `"identity"` (e.g. Cholesky or factor-analytic loadings), `"tanh"` or `"bounded_logit"` (correlation-type parameters).

Specifying constraints

vcc is a data frame with columns

parameter the `index` of the parameter, or its name `"term:parameter"` (e.g. `"vsm(ism(B)):sigma2"`); the `parameter` name alone is accepted when unique.

group parameters with the same group value are constrained together.

scale optional, default 1: each member equals `scale` times the first member of its group (natural scale). Scaling is allowed for `"scale"`, `"exp"` and `"identity"` parameters; correlation-type parameters can only be equated.

All members of a group must have the same `kind`: variance scales can be constrained with other variance scales, and other parameters with parameters estimated through the same transform. If any member of a group is fixed, the whole group is fixed at the value implied by that member. Starting values of the members are made consistent with the first (or fixed) member.

See Also

`mmes`, `vsm`, `covparams_mmes`

Examples

```
data(DT_yatesoats, package = "enhancer")
DT <- DT_yatesoats
p <- mmes(Y ~ V * N,
  random = ~ B + B:MP,
  rcov = ~ units,
  data = DT,
  control = mmes_control(output = list(setup = TRUE)))
p$vcParams
# block and whole-plot variances equal
m1 <- mmes(Y ~ V * N,
  random = ~ B + B:MP,
  rcov = ~ units,
  data = DT,
  vcc = data.frame(parameter = c(1, 2), group = 1),
  control = mmes_control(diagnostics = list(verbose = FALSE)))
# block variance twice the whole-plot variance
m2 <- mmes(Y ~ V * N,
  random = ~ B + B:MP,
  rcov = ~ units,
  data = DT,
  vcc = data.frame(parameter = c("vsm(ism(B:MP)):sigma2", "vsm(ism(B)):sigma2"),
                           group = 1, scale = c(1, 2)),
  control = mmes_control(diagnostics = list(verbose = FALSE)))
unlist(m2$covPar)
```

`vpredict` *vpredict form of a LMM fitted with mmes*

Description

`vpredict` method for class `"mmes"`.

Post-analysis procedure to calculate linear combinations of variance components. Its intended use is when the variance components are either simple variances or are variances and covariances in an unstructured matrix. The functions covered are linear combinations of the variance components (for example, phenotypic variance), a ratio of two components (for example, heritabilities) and the correlation based on three components (for example, genetic correlation).

The calculations are based on the estimated variance parameters and their variance matrix as represented by the inverse of the Fisher or Average information matrix. Note that this matrix has zero values for fixed variance parameters including those near the parameter space boundary.

The transform is specified with a formula. On the left side of the formula is a name for the transformation. On the right side of the formula is a transformation specified with shortcut names like 'V1', 'V2', etc. The easiest way to identify these shortcut names is to use 'summary(object)$varcomp'. The rows of this object can referred to with shortcuts 'V1', 'V2', etc. See the example below.

Usage

```
vpredict(object, transform)
## S3 method for class 'mmes'
vpredict(object, transform)
```

Arguments

`object`	a model fitted with the mmes function.
`transform`	a formula to calculate the function.

Details

The delta method (e.g., Lynch and Walsh 1998, Appendix 1; Ver Hoef 2012) uses a Taylor series expansion to approximate the moments of a function of parameters. A first-order delta-method approximation is used for the standard error of a function of (co)variance parameters. Partial first derivatives of the function are calculated by algorithmic differentiation with `deriv`. Estimates use the stored native-scale covariance parameter table, and uncertainty uses its full covariance matrix, including covariance between different fitted terms. Parameters with zero derivatives do not contribute uncertainty to the requested transformation.

Though `vpredict` can calculate standard errors for non-linear functions of (co)variance parameters from a fitted `mmes` model, it is limited to non-linear functions constructed by mathematical operations such as the arithmetic operators `+`, `-`, `*`, `/` and `^`, and single-variable functions such as `exp` and `log`. See `deriv` for more information.

Value

`dd`	the parameter and its standard error.

Author(s)

Giovanny Covarrubias

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Lynch, M. and B. Walsh 1998. Genetics and Analysis of Quantitative Traits. Sinauer Associates, Inc., Sunderland, MA, USA.

Ver Hoef, J.M. 2012. Who invented the delta method? The American Statistician 66:124-127. DOI: 10.1080/00031305.2012.687494

See Also

vpredict, mmes

Examples

```
####============================================####
####============================================####
#### EXAMPLE 1
#### simple example with univariate models
####============================================####
####============================================####
# data(DT_cpdata, package="enhancer")
# DT <- DT_cpdata
# GT <- GT_cpdata
# MP <- MP_cpdata
# #### create the variance-covariance matrix
# A <- A.mat(GT)
# #### look at the data and fit the model
# head(DT)
# mix1 <- mmes(Yield~1,
#              random=~vsm(ism(id),Gu=A),
#              data=DT)
# summary(mix1)$varcomp
# #### run the vpredict function
# vpredict(mix1, h2 ~ V1 / ( V1 + V2 ) )
```

---

vs *variance structure specification*

Description

vs DEPRECATED NOW. was the main function to build the variance-covariance structure for the random effects to be fitted in the mmer solver.

Usage

```
vs(..., Gu=NULL, Gti=NULL, Gtc=NULL, reorderGu=TRUE, buildGu=TRUE)
```

Arguments

...	variance structure to be specified following the logic desired in the internal kronecker product. For example, if user wants to define a diagonal variance structure for the random effect 'genotypes'(g) with respect to a random effect 'environments'(e), this is:

`var(g) = G.e @ I.g`

being `G.e` a matrix containing the variance covariance components for g (genotypes) in each level of e (environments), `I.g` is the covariance among levels of g (genotypes; i.e. relationship matrix), and @ is the kronecker product. This would be specified in the mmer solver as:

`random=~vs(dsr(e),g)`

One strength of sommer is the ability to specify very complex structures with as many kronecker products as desired. For example:

`var(g) = G.e @ G.f @ G.h @ I.g`

is equivalent to

`random=~vs(e,f,h,g)`

where different covariance structures can be applied to the levels of `e,f,h` or a combination of these). For more examples please see the vignettes 'sommer.start' available in the package.

`Gu` matrix with the known variance-covariance values for the levels of the u.th random effect (i.e. relationship matrix among individuals or any other known covariance matrix). If `NULL`, then an identity matrix is assumed. The Gu matrix can have more levels than the ones present in the random effect linked to it but not the other way around. Otherwise, an error message of missing level in Gu will be returned.

`Gti` matrix with dimensions t x t (t equal to number of traits) with initial values of the variance-covariance components for the random effect specified in the .... argument. If `NULL` the program will provide the initial values. The values need to be scaled, see Details section.

`Gtc` matrix with dimensions t x t (t equal to number of traits) of constraints for the variance-covariance components for the random effect specified in the ... argument according to the following rules:

`0`: not to be estimated

`1`: estimated and constrained to be positive (i.e. variance component)

`2`: estimated and unconstrained (can be negative or positive, i.e. covariance component)

`3`: not to be estimated but fixed (value has to be provided in the `Gti` argument)

In the multi-response scenario if the user doesn't specify this argument the default is to build an unstructured matrix (using the `unsm`() function). This argument needs to be used wisely since some covariance among responses may not make sense. Useful functions to specify constraints are; `diag`(), `unsm`(), `fixm`().

`reorderGu` a `TRUE/FALSE` statement if the Gu matrix should be reordered based on the names of the design matrix of the random effect or passed with the custom order of the user. This may be important when fitting covariance components in a customized fashion. Only for advanced users.

`buildGu` a `TRUE/FALSE` statement to indicate if the Gu matrix should be built in R when the value for the argument `Gu=NULL`. Repeat, only when when the value for the argument `Gu` is equal to `NULL`. In some cases when the incidence matrix is wide (e.g. rrBLUP models) the covariance structure is a huge p x p matrix that can be avoided when performing matrix operations. By setting this argument to `FALSE` it allows to skip forming this covariance matrix.

Details

When providing initial values in the `Gti` argument the user has to provide scaled variance component values. The user can provide values from a previous model by accessing the `sigma_scaled` output from an `mmer` model or if an specific value is desired the user can obtain the scaled value as:

`m = x/var(y)`

where `x` is the desired initial value and `y` is the response variable. You can find an example in the `DT_cpdata` dataset.

Value

$res a list with all neccesary elements (incidence matrices, known var-cov structures, unknown covariance structures to be estimated and constraints) to be used in the mmer solver.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Covarrubias-Pazaran G (2018) Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639

See Also

The core function of the package: `mmer`

Examples

```
data(DT_example, package="enhancer")
DT <- DT_example
A <- A_example

## ============================ ##
## example to without structure
## ============================ ##

mix <- mmer(Yield~Env,
            random= ~ vs(Name),
            rcov=~ vs(units),
            data=DT)
```

vsm *Variance-structure model with arbitrary Kronecker covariance factors*

Description

vsm constructs random-effect or residual covariance structures for mmes. Covariance-shaping terms are represented by CovarianceFactor v2 descriptors and may be combined in an arbitrary-depth Kronecker product. A single product-level variance, σ^2, is owned by vsm, which avoids scale confounding among the individual covariance factors.

Usage

```
vsm(..., Gu = NULL, sigma2 = NULL, fixedSigma2 = FALSE,
  rotation = FALSE, isFixed = FALSE, verbose = TRUE)
```

Arguments

...	One or more covariance-constructor terms. The last term supplies the main-effect incidence matrix. All preceding terms are covariance-shaping factors. Examples include dsm(environment), ar1m(row), and ism(genotype).
Gu	Optional known precision matrix for the levels of the final main-effect term. For mmes, Gu must have row and column names matching the main-effect levels and attr(Gu, "inverse") must be TRUE. If omitted, an identity precision matrix is used. Gu can also be supplied in 3-column (row, column, value) format as a numeric matrix or data frame, e.g. the output of ASReml-R ainverse() or the listAinv element of nadiv::makeAinv(). The first two columns are 1-based level indices and the third the precision values. The level names must be given in attr(Gu, "rowNames"). The lower triangle, the upper triangle, or both may be given (entries present in both must agree), and every level needs a diagonal entry. This format is detected automatically, is always taken to be a precision matrix, and does not require the inverse attribute. Symmetric precision matrices are stored with one triangle only.
sigma2	Positive starting value for the single overall covariance scale of this vsm term. If NULL (the default), mmes replaces it with a data-driven starting value computed from the response and fixed effects; supplying a value here always takes precedence and is never overridden.
fixedSigma2	Logical indicating whether the product-level σ^2 is fixed at its starting value.
rotation	Logical indicating whether the supplied Gu precision matrix should be eigen-decomposed for Lee–van der Werf rotation in mmes. Rotation is opt-in and currently requires a balanced Gaussian random-effect term and a rotation-invariant residual structure (see mmes).
isFixed	Logical retained for compatibility. If TRUE, return the combined design matrix instead of the structured vsm object.
verbose	Logical controlling messages, including notification when levels present in Gu are appended to the model matrix.

Details

Let $K_1, \ldots, K_m$ be the dimensionless covariance shapes supplied by the covariance constructors. The covariance represented by one `vsm` term is

$$\Sigma = \sigma^2(K_1 \otimes K_2 \otimes \cdots \otimes K_m).$$

For a random effect with known relationship precision `Gu`, the factor product determines the covariance among the crossed covariance coordinates and `Gu` determines the relationship structure among the levels of the final main effect.

The final argument in `...` is the main-effect incidence term. Every term before it is a covariance-shaping factor. Thus `vsm(dsm(location), ar1m(row), ism(genotype), Gu=Ainv)` represents a location-by-row covariance shape crossed with genotype. Kronecker ordering is left to right: earlier factors are the slow index and later factors are the fast index.

All optimizer coordinates are stored on unconstrained working scales. Each CovarianceFactor descriptor also carries its evaluator, derivative strategy, reporting transformation, and trust-region caps. Built-in high-use structures may use native C++ covariance primitives, while newer or user-defined structures can use generic R callbacks without requiring changes to `ai_mme_sp2`.

For residual covariance models, the covariance-shaping factors must define exactly one product coordinate for each observation. `mmes` combines that local coordinate with an independently constructed residual block index, so a large observation-level identity design is not materialized.

When `rotation=TRUE`, `Gu` is decomposed as

$$Gu = U\Lambda U',$$

where Λ is diagonal. The aligned original precision, diagonal precision, and eigenvectors are retained in the returned descriptor. The `mmes` front end validates the observation layout and applies the engine-specific rotation after observation filtering. Residual covariance must satisfy $R_{(i,a),(l,b)} = c_{ab}\delta_{il}$, where i, l index relationship levels and a, b index rotation blocks, so that $U'RU = R$; otherwise `mmes` stops with an error.

Value

A list containing the random-effect design blocks in `Z`, the precision matrix in `Gu`, the flattened Kronecker descriptor in `covStruct`, the product design, and, for residual models, the local covariance-coordinate index. When rotation is requested, `GuRot` contains the diagonal precision and `rotation` contains its eigen metadata. The `covStruct` object contains descriptor version 2 CovarianceFactor objects.

See Also

`mmes`, `ism`, `dsm`, `atm`, `usm`, `csm`, `ar1m`, `ar2m`, `ar3m`, `mam`, `corgm`, `fam`, `antem`, `rrm`, `maternm`, `toeplitzm`, `sar`, `car`, and `ownm`. See `dsumm` for section-specific (direct-sum) residual structures.

Examples

```
####==========================================####
#### For CRAN time limitations most lines in the
#### examples are silenced with one '#' mark,
```

```
#### remove them and run the examples
####=========================================####

data(DT_example, package="enhancer")
DT <- DT_example
head(DT)


str(with(DT, vsm(dsm(Env),ism(Name))))
str(with(DT, vsm(csm(Env),ism(Name))))
str(with(DT, vsm(toeplitzm(Env),ism(Name))))

## Gu in 3-column (row, column, value) format with level names in rowNames
Ainv <- data.frame(Row=c(1,2,2,3), Column=c(1,1,2,3),
                   Ainverse=c(2,-1,2,1))
attr(Ainv, "rowNames") <- c("a","b","c")
id <- factor(c("a","b","c","a"))
str(vsm(ism(id), Gu=Ainv)$Gu)
```

`vsr` *variance structure specification*

Description

`vsr` DEPRECATED NOW. was the main function to build the variance-covariance structure for the random effects to be fitted in the `mmer` solver.

Usage

```
vsr(..., Gu=NULL, Gti=NULL, Gtc=NULL, reorderGu=TRUE, buildGu=TRUE)
```

Arguments

`...`	variance structure to be specified following the logic desired in the internal kronecker product. For example, if user wants to define a diagonal variance structure for the random effect 'genotypes'(g) with respect to a random effect 'environments'(e), this is:
	`var(g) = G.e @ I.g`
	being `G.e` a matrix containing the variance covariance components for g (genotypes) in each level of e (environments), `I.g` is the covariance among levels of g (genotypes; i.e. relationship matrix), and @ is the kronecker product. This would be specified in the mmer solver as:
	`random=~vsr(dsr(e),g)`
	One strength of sommer is the ability to specify very complex structures with as many kronecker products as desired. For example:
	`var(g) = G.e @ G.f @ G.h @ I.g`
	is equivalent to

random=~vsr(e,f,h,g)

where different covariance structures can be applied to the levels of e,f,h. For more examples please see the vignettes 'sommer.start' available in the package.

Gu — matrix with the known variance-covariance values for the levels of the u.th random effect (i.e. relationship matrix among individuals or any other known covariance matrix). If NULL, then an identity matrix is assumed. The Gu matrix can have more levels than the ones present in the random effect linked to it but not the other way around. Otherwise, an error message of missing level in Gu will be returned.

Gti — matrix with dimensions t x t (t equal to number of traits) with initial values of the variance-covariance components for the random effect specified in the .... argument. If NULL the program will provide the initial values. The values need to be scaled, see Details section.

Gtc — matrix with dimensions t x t (t equal to number of traits) of constraints for the variance-covariance components for the random effect specified in the ... argument according to the following rules:

0: not to be estimated

1: estimated and constrained to be positive (i.e. variance component)

2: estimated and unconstrained (can be negative or positive, i.e. covariance component)

3: not to be estimated but fixed (value has to be provided in the Gti argument)

In the multi-response scenario if the user doesn't specify this argument the default is to build an unstructured matrix (using the unsm() function). This argument needs to be used wisely since some covariance among responses may not make sense. Useful functions to specify constraints are; diag(), unsm(), fixm().

reorderGu — a TRUE/FALSE statement if the Gu matrix should be reordered based on the names of the design matrix of the random effect or passed with the custom order of the user. This may be important when fitting covariance components in a customized fashion. Only for advanced users.

buildGu — a TRUE/FALSE statement to indicate if the Gu matrix should be built in R when the value for the argument Gu=NULL. Repeat, only when when the value for the argument Gu is equal to NULL. In some cases when the incidence matrix is wide (e.g. rrBLUP models) the covariance structure is a huge p x p matrix that can be avoided when performing matrix operations. By setting this argument to FALSE it allows to skip forming this covariance matrix.

Details

When providing initial values in the Gti argument the user has to provide scaled variance component values. The user can provide values from a previous model by accessing the sigma_scaled output from an mmer model or if an specific value is desired the user can obtain the scaled value as:

m = x/var(y)

where x is the desired initial value and y is the response variable. You can find an example in the DT_cpdata dataset.

Value

$res a list with all neccesary elements (incidence matrices, known var-cov structures, unknown covariance structures to be estimated and constraints) to be used in the mmer solver.

Author(s)

Giovanny Covarrubias-Pazaran

References

Covarrubias-Pazaran G (2016) Genome assisted prediction of quantitative traits using the R package sommer. PLoS ONE 11(6): doi:10.1371/journal.pone.0156744

Covarrubias-Pazaran G (2018) Software update: Moving the R package sommer to multivariate mixed models for genome-assisted prediction. doi: https://doi.org/10.1101/354639

Examples

```
data(DT_example, package="enhancer")
DT <- DT_example
A <- A_example

## ============================ ##
## example to without structure
## ============================ ##

mix <- mmer(Yield~Env,
            random= ~ vsr(Name),
            rcov=~ vsr(units),
            data=DT)
```

`wald_mmes` *Wald tests for fixed effects with optional small-sample denominator df*

Description

Pseudo analysis of variance for the fixed terms of an `mmes` fit, based on incremental or conditional Wald statistics, with chi-square, residual, Satterthwaite or Kenward-Roger reference distributions.

Usage

```
wald_mmes(object, ssType = c("incremental", "conditional"),
          denDF = c("none", "residual", "satterthwaite", "kr"), terms = NULL)
```

Arguments

object	a model of class "mmes".
ssType	"incremental" tests each term adjusted for the terms preceding it in the fixed formula. "conditional" tests each term adjusted for every other term that does not contain it, so marginality is respected: a term is never adjusted for a term that contains it structurally (A:B contains A) or implicitly (its columns, together with the intercept, span the tested term, e.g. locations coded separately within regions).
denDF	"none" gives chi-square tests of the Wald statistic. "residual" uses F tests with n - rank(X) denominator df. "satterthwaite" and "kr" (Kenward-Roger) compute approximate denominator df; "kr" also uses the Kenward-Roger adjusted covariance of the fixed effects and scaled F statistic.
terms	optional subset of fixed terms (as named in object$Dtable).

Details

Let Φ be the covariance of the fixed effects and $M = \Phi^{-1}$. For a given order of the terms, the upper Cholesky factor U of M defines $z = Ub$; the incremental Wald statistic of a term is the sum of the squared elements of z belonging to it, i.e. a test of $Lb = 0$ with L the corresponding rows of U. Conditional tests use the order [terms not containing the tested term, tested term, terms containing it].

For Satterthwaite and Kenward-Roger df the model inputs are rebuilt from the stored call and $X^\top V^{-1} X$ is re-evaluated at perturbed covariance parameters (on the reported scale of covPar); its numerical first and second derivatives give the quantities P_i and the Kenward-Roger adjustment. The covariance of the covariance parameters is theta_se, the inverse average information. The second-derivative term of V enters through the Hessian of $X^\top V^{-1} X$; it vanishes for variance-component models, where the method is the exact Kenward-Roger approximation. Because the average (rather than expected) information is used, df can differ slightly from software using the expected information. The re-evaluation requires the data and objects used in the original call to be available, REML estimation, and a Gaussian fit.

anova(object) with a single model returns wald_mmes(object, ...).

Value

A data frame of class "wald.mmes" with columns Df, denDF, Wald, F.value and p.value, one row per term.

References

Kenward MG and Roger JH (1997). Small sample inference for fixed effects from restricted maximum likelihood. Biometrics 53: 983-997.

See Also

mmes, predict.mmes, anova.mmes

Examples

```
data(DT_yatesoats, package = "enhancer")
m <- mmes(Y ~ V * N,
  random = ~ B + B:MP,
  rcov = ~ units,
  data = DT_yatesoats,
  control = mmes_control(diagnostics = list(verbose = FALSE)))
wald_mmes(m)
wald_mmes(m, ssType = "conditional", denDF = "kr")
```

Index

∗ **R package**
  sommer-package, 4
∗ **manip**
  stackTraits, 131
∗ **models**
  anova.mmes, 15
  binm, 24
  coef.mmes, 26
  covmatrix_mmes, 33
  familym, 50
  fitted.mmes, 51
  meffects_mmes, 66
  metricm, 68
  plot.mmes, 102
  pmonitor, 103
  predict.mmes, 108
  residuals.mmes, 118
  strm, 132
  summary.mmer, 133
  summary.mmes, 134
  vcc, 142
  vpredict, 143
  wald_mmes, 152

A.mat, *4*, 12, *18*, *48*, *67, 68*, *74*
analyze_mme_crossovers, 14
anova, *15*
anova.mmer, *74*
anova.mmer *(*anova.mmes*)*, 15
anova.mmes, *5*, 15, *153*
antem, 15, *81*, *149*
APY, 16
ar1m, *16*, 19, *21, 22*, *46*, *65*, *70*, *81*, *135*, *149*
ar2m, *19*, 20, *22*, *81*, *135*, *149*
ar3m, *19*, *21*, 21, *81*, *135*, *149*
atcg1234, *4*
atm, 22, *23, 24*, *81*, *149*
atr, 23, *39*, *45*, *71*, *73*, *141*

binm, 24, *84*, *118*
build.HMM, *4*, *74*

car, 25, *66*, *70*, *81, 82*, *101*, *123*, *149*
coef, *26*
coef.mmer, *74*
coef.mmer *(*coef.mmes*)*, 26
coef.mmes, *5*, 26
corgm, 27, *38*, *50, 51*, *81*, *149*
corImputation, 28
covm, 30, *133*
covmatrix_mmes, *28*, 33, *51*, *107*, *132*
covparams_mmes, *27*, *34*, 35, *37*, *46*, *101*, *143*
covparams_mmes_se, *34*, *36*, 36
csm, *28*, 37, *81, 82*, *149*
csr, *24*, 39, *45*, *71*, *141*

D.mat, *4*, 40, *48*, *74*
deriv, *144*
dfToMatrix, 42
diag, *71*, *146*, *151*
dsm, *23*, *38*, 43, *44–46*, *81*, *149*
dsr, *24*, *39*, 44, *71*, *73*, *141*
dsumm, *19*, *34, 35*, 45, *82*, *149*
DT_btdata, *6*, *75*
DT_cornhybrids, *6*, *75*
DT_cpdata, *6*, *75*, *147*, *151*
DT_fulldiallel, *5*, *74*
DT_gryphon, *6*, *75*
DT_h2, *5*, *74*
DT_halfdiallel, *5*, *74*
DT_ige, *75*
DT_legendre, *6*, *75*
DT_mohring, *5*, *74*
DT_polyploid, *6*, *75*
DT_sleepstudy, *6*, *75*
DT_technow, *6*, *75*
DT_wheat, *6*, *75*

E.mat, *4*, 47, *74*

fam, 48, *64*, *81*, *121*, *125*, *149*

familym, *27, 28*, 50, *84*, *132*
fitted, *52*
fitted.mmer, *74*
fitted.mmer *(*fitted.mmes*)*, 51
fitted.mmes, *5*, 51
fixm, 52, *71*, *146*, *151*

generate_mme_workload, 53, *116*
GWAS, 54

H.mat, *4*, *18*, 60, *74*

ism, *6*, 62, *81*, *149*

leg, *71*, *74*, *81*
loadings_mmes, 63, *124, 125*

ma1m *(*mam*)*, 64
ma2m *(*mam*)*, 64
mam, 64, *81*, *135*, *149*
manhattan, *4*, *74*
map.plot, *4*, *74*
maternm, *26*, 65, *69, 70*, *81, 82*, *101*, *123*, *149*
meffects_mmes, 66
metricm, *66*, 68, *81*
mme_workload_grid, 96
mmer, *4*, *8*, *10*, *24*, *39*, *45*, *54*, 70, *134*, *141*, *145*, *147*, *150*
mmes, *4*, *8*, *10*, *15*, *18*, *23–26*, *30–34*, *36, 37*, *44*, *46*, *50–52*, *54*, *62*, *64*, *67, 68*, *70*, 79, *91*, *95, 96*, *100*, *102, 103*, *112*, *117–119*, *125*, *128*, *131–134*, *140*, *142, 143*, *145*, *148, 149*, *152, 153*
mmes_control, *54*, *79, 80*, *83*, *85–87*, 91, *116*, *122*
MNR, 97

overlay, *71*, *73, 74*, *81*
ownm, *36*, *81, 82*, 101, *121*, *149*

p.adjust, *109*
plot, *102, 103*
plot.mmer, *74*
plot.mmer *(*plot.mmes*)*, 102
plot.mmes, *5*, 102
pmonitor, 103
postPEV, *85*, *93*, 104, *107*
postVarU, *104*, 106, *111*
predict, *112*
predict.mmer, *74*
predict.mmer *(*predict.mmes*)*, 108
predict.mmes, *5*, *85*, *92*, *104*, *107*, 108, *153*
print.mmer *(*summary.mmer*)*, 133
print.mmes *(*summary.mmes*)*, 134
print.sommer_mme_benchmark, 114
print.sommer_mme_crossover, 115
print.sommer_mme_profile, 115
print.sommer_mmes_control, 114
print.summary.mmer *(*summary.mmer*)*, 133
print.summary.mmes *(*summary.mmes*)*, 134
print.wald.mmes *(*wald_mmes*)*, 152
profile_mme_workload, 116

r2, *5*, 116
randef, *74*, 117
redmm, *81*
residuals, *118*
residuals.mmer, *74*
residuals.mmer *(*residuals.mmes*)*, 118
residuals.mmes, *5*, 118
rotate_back_mmes, 119
rrm, *49, 50*, *64*, *81*, *101*, 119, *125*, *149*
run_mme_benchmark, 122

sar, *26*, *66*, *70*, *81, 82*, *101*, 123, *149*
scores_mmes, *64*, 124
sommer, *58*, *75*
sommer *(*sommer-package*)*, 4
sommer-package, 4
spl2Da, *71*, *74*
spl2Da *(*spl2Dc*)*, 125
spl2Db, *71*, *74*
spl2Db *(*spl2Dc*)*, 125
spl2Dc, *5*, *81*, 125
spl2Dmats, 128
stackTraits, *34*, *50, 51*, *82*, 131
strm, 132
summary, *134*
summary.mmer, *74*, 133
summary.mmes, *5*, 134

toeplitzm, *16*, *19*, *21, 22*, *65*, *81, 82*, 135, *149*
tpsmmbwrapper, 136

unsm, *71*, 139, *146*, *151*
unsm2 *(*unsm*)*, 139
usm, *27, 28*, *50*, *81, 82*, *133*, 140, *141*, *149*
usr, *24*, *39*, *45*, *71*, *73*, 141

vcc, *83*, 142

vcParams (vcc), 142
vpredict, *5*, *34*, *74*, 143, *145*
vs, 145
vsm, *4*, *6, 7*, *10*, *16*, *19*, *21–23*, *26*, *28*, *30–32*, *36–38*, *44–46*, *50*, *52, 53*, *62*, *64–66*, *68*, *70*, *79–82*, *101*, *119*, *121*, *123*, *125*, *132, 133*, *135*, *139, 140*, *143*, 148
vsr, *23, 24*, *39*, *44, 45*, *55, 56*, *71–74*, *141*, 150

wald_mmes, *15*, *109*, 152