Package ‘spaMM’

May 30, 2020

Type Package

Title Mixed-Effect Models, Particularly Spatial Models

Encoding UTF-8

Version 3.3.0

Date 2020-05-30

Maintainer François Rousset <francois.rousset@umontpellier.fr>

Imports methods, stats, graphics, Matrix, MASS, proxy, Rcpp (>= 0.12.10), nlme, nloptr, minqa, pbapply, crayon, gmp, ROI, boot

LinkingTo Rcpp, RcppEigen (>= 0.3.3.5.0)

Depends R (>= 3.2.0)

Suggests maps, testthat, lme4, rsae, rcdd, pedigreemm, foreach, multilevel, Infusion (>= 1.3.0), IsoriX (>= 0.8.1), blackbox (>= 1.1.25), RSpectra, ROI.plugin.glpk

Enhances multcomp

NeedsCompilation yes

Description Inference based on mixed-effect models, including generalized linear mixed models with spatial correlations and models with non-Gaussian random effects (e.g., Beta). Both classical geostatistical models, and Markov random field models on irregular grids, can be fitted. Variation in residual variance (heteroscedasticity) can itself be represented by a generalized linear mixed model. Various approximations of likelihood or restricted likelihood are implemented, in particular h-likelihood (Lee and Nelder 2001 <doi:10.1093/biomet/88.4.987>) and Laplace approximation.

License CeCILL-2

URL https://www.r-project.org,
http://kimura.univ-montp2.fr/~rousset/spaMM.htm

RoxygenNote 6.1.1

ByteCompile true

Author François Rousset [aut, cre, cph]
(<https://orcid.org/0000-0003-4670-0371>),
Jean-Baptiste Ferdy [aut, cph],

Alexandre Courtiol [aut] (<https://orcid.org/0000-0003-0637-2959>),
GSL authors [ctb] (src/gsl_bessel.*)

Repository CRAN

Date/Publication 2020-05-30 05:10:02 UTC

R topics documented:

`adjlg` *Simulated data set for testing sparse-precision code*

Description

This is used in `tests/test-adjacency-long.R`

Usage

```
data("adjlg")
```

Format

Includes an adjacency matrix `adjlgMat`. and a data frame `adjlg` with 5474 observations on the following 8 variables.

`ID` a factor with levels 1 to 1000

`months` a numeric vector

`GENDER` a character vector

`AGE` a numeric vector

`X1` a numeric vector

`X2` a numeric vector

`month` a numeric vector

`BUY` a numeric vector

Source

The simulation code show in Example was suggested by Jeroen van den Ochtend.

Examples

```
data(adjlg)
## See further usage in tests/test-adjacency-long.R
## Not run:
# as produced by:
  library(data.table) ## Included data produced using version 1.10.4.3
  library(igraph) ## Included data produced using version 1.2.1

  rsample <- function(N=100, ## size of implied adjacency matrix
                      month_max=10,seed) {
    if (is.integer(seed)) set.seed(seed)
    dt <- data.table(ID=factor(1:N))
    dt$months <- sample(1:month_max,N,replace=T) ## # of liens for each level of ID
    dt$GENDER <- sample(c("MALE","FEMALE"),N,replace=TRUE)
    dt$AGE <- sample(18:99,N,replace=T)
    dt$X1 <- sample(1000:9900,N,replace=T)
    dt$X2 <-  runif(N)

    dt <- dt[, c(.SD, month=data.table(seq(from=1, to=months, by = 1))), by = ID]
    dt[,BUY := 0]
    dt[month.V1==months,BUY := sample(c(0,1),1),by=ID]
    setnames(dt,"month.V1","month")

    #### create adjacency matrix
    Network <- data.table(OUT=sample(dt$ID,N*month_max*4/10))
    Network$IN <- sample(dt$ID,N*month_max*4/10)
    Network <- Network[IN != OUT]
    Network <- unique(Network)
    g <- graph.data.frame(Network,directed=F)
   g <- add_vertices(g,sum(!unique(dt$ID) %in% V(g)),name=unique(dt[!dt$ID %in% V(g),list(ID)]))
    Network <- as_adjacency_matrix(g,sparse = TRUE,type="both")
```

```
  return(list(data=dt,adjMatrix=Network))
}

set.seed(123)
adjlg_sam <- rsample(N=1000,seed=NULL)
adjlg <- as.data.frame(adjlg_sam$data)
adjlgMat <- adjlg_sam$adjMatrix

## End(Not run)
```

AIC *Extractors for information criteria such as AIC*

Description

get_any_IC computes model selection/information criteria such as AIC. See Details for more information about these criteria. The other extractors AIC and extractAIC are methods for HLfit objects of generic functions defined in other packages; AIC is equivalent to get_any_IC, and extractAIC additionally returns a number of degrees of freedom.

Usage

```
get_any_IC(object, ..., verbose=interactive() ,also_cAIC=TRUE)
## S3 method for class 'HLfit'
AIC(object, ..., k, verbose=interactive() ,also_cAIC=TRUE)
## S3 method for class 'HLfit'
extractAIC(fit, scale, k, ..., verbose=FALSE)
```

Arguments

object, fit	A object of class HLfit, as returned by the fitting functions in spaMM.
scale, k	Currently ignored, but are required in the definitions for consistency with the generic.
verbose	Whether to print the model selection criteria or not.
also_cAIC	Whether to include the conditional AIC in the result (its computation may be slow).
...	Other arguments that may be needed by some method.

Details

get_any_IC computes, optionally prints, and returns invisibly the following quantities. The **conditional AIC** (Vaida and Blanchard 2005) is a relative measure of quality of prediction of new realizations of a mixed model, conditional on the realized values of the random effects. It involves the conditional likelihood, and degrees of freedom for (i) estimated residual error parameters and (ii) the overall linear predictor characterized by the **Effective degrees of freedom** already discussed by previous authors including Lee and Nelder (2001), which gave a general formula for it in HGLMs. Both a plug-in "asymptotic" estimate of the conditional AIC and of this effective df are returned by `get_any_IC`. Note that these may be biased estimates of conditional AIC and effective df, and that more refined formulas are discussed in the literature (e.g. Overholser and Xu 2014), some of which may be implemented in future versions of `get_any_IC`. Lee et al. (2006) and Ha et al (2007) defined a corrected AIC [i.e., AIC(D*) in their eq. 7] which is here interpreted as the conditional AIC. Also returned are the **marginal AIC** (Akaike's classical AIC), and a focussed AIC for dispersion parameters (**dispersion AIC**) discussed by Ha et al (2007; eq.10). This diversity of criteria should encourage users to think twice before applying model selection automatically, which is no better although more fashionable than misuses of simple null hypothesis testing. Also, alternative procedures for model choice can be considered (e.g. Cox and Donnelly, 2011, p. 130-131).

Value

For `AIC` and `get_any_IC`, a numeric vector whose elements are described in the Details.

For `extractAIC`, a numeric vector of length 2, with first and second elements giving

`edf`	the degree of freedom of the fixed-effect terms of the model for the fitted model `fit`.
`AIC`	the (marginal) Akaike Information Criterion for `fit`.

References

Cox, D. R. and Donnelly C. A. (2011) Principles of Applied Statistics. Cambridge Univ. Press.

Ha, I. D., Lee, Y. and MacKenzie, G. (2007) Model selection for multi-component frailty models. Statistics in Medicine 26: 4790-4807.

Overholser R., and Xu R. (2104) Effective degrees of freedom and its application to conditional AIC for linear mixed-effects models with correlated error structures. J. Multivariate Anal. 132: 160-170.

Vaida, F., and Blanchard, S. (2005) Conditional Akaike information for mixed-effects models. Biometrika 92, 351-370.

Examples

```
data("wafers")
m1 <- HLfit(y ~ X1+X2+(1|batch), resid.model = ~ 1, data=wafers, method="ML")
get_any_IC(m1)
extractAIC(m1)
```

`arabidopsis` *Arabidopsis genetic and climatic data*

Description

For 948 "accessions" from European Arabidopsis thaliana populations, this data set merges the genotypic information at four single nucleotide polmorphisms (SNP) putatively involved in adaptation to climate (Fournier-Level et al, 2011, Table 1), with 13 climatic variables from Hancock et al. (2011).

Usage

```
data("arabidopsis")
```

Format

The data frame includes 948 observations on the following variables:

pos1046738, pos5510910, pos6235221, pos8132698 Genotypes at four SNP loci

LAT latitude

LONG longitude

seasonal, tempWarmest, tempColdest, preciWettest, preciDriest, preciCV, PAR_SPRING,

growingL, conseqCold, conseqFrFree, RelHumidSp, dayLSp, aridity Thirteen climatic variables. See Hancock et al. (2011) for details about these variables.

Details

The response is binary so `method="PQL/L"` seems warranted (see Rousset and Ferdy, 2014).

Source

The data were retrieved from `http://bergelson.uchicago.edu/regmap-data/climate-genome-scan` on 22 February 2013 (they may no longer be available from there).

References

Fournier-Level A, Korte A., Cooper M. D., Nordborg M., Schmitt J., Wilczek AM (2011). A map of local adaptation in Arabidopsis thaliana. Science 334: 86-89.

Hancock, A. M., Brachi, B., Faure, N., Horton, M. W., Jarymowycz, L. B., Sperone, F. G., Toomajian, C., Roux, F., and Bergelson, J. 2011. Adaptation to climate across the Arabidopsis thaliana genome, Science 334: 83-86.

Rousset F., Ferdy, J.-B. (2014) Testing environmental and genetic effects in the presence of spatial autocorrelation. Ecography, 37: 781-790. http://dx.doi.org/10.1111/ecog.00566

Examples

```
data("arabidopsis")
if (spaMM.getOption("example_maxtime")>3) {
  fitme(cbind(pos1046738,1-pos1046738)~seasonal+Matern(1|LAT+LONG),
        fixed=list(rho=0.119278,nu=0.236990,lambda=8.599),
        family=binomial(),method="PQL/L",data=arabidopsis)
}
## The above 'fixed' values are deduced from the following fit:
if (spaMM.getOption("example_maxtime")>46) {
  SNPfit <- fitme(cbind(pos1046738,1-pos1046738)~seasonal+Matern(1|LAT+LONG),
              verbose=c(TRACE=TRUE),
              family=binomial(),method="PQL/L",data=arabidopsis)
  summary(SNPfit) # p_v=-125.0392
}
```

`autoregressive` *Fitting autoregressive models*

Description

Two autoregressive(AR) models are currently implemented: the `adjacency` model (a conditional AR, i.e., CAR), and the `AR1` model for time series.

An AR1 random effect is specified as `AR1(1|<grouping factor>)`. It describes correlations between realizations of the random effect for (typically) successive time-steps by a correlation ϕ, denoted `ARphi` in function calls. Nested AR1 effects can be specified by a nested grouping factor, as in `AR1(1|<time index> %in% <nesting factor>)`.

A CAR random effect is specified as `adjacency(1|<grouping factor>)`. The correlations among levels of the random effect form a matrix $(\mathbf{I}-\rho$ `adjMatrix`$)^{-1}$, in terms of an `adjMatrix` matrix which must be provided, and of the scalar ρ, denoted `rho` in function calls. The rows and columns of `adjMatrix` must have names matching those of levels of the random effect **or else** be ordered as increasing values of the levels of the geographic location index specifying the spatial random effect. For example, if the model formula is
`y ~ adjacency(1|geo.loc)` and `<data>$geo.loc` is 2,4,3,1,... the first row/column of the matrix refers to `geo.loc`=1, i.e. to the fourth row of the data.

Details

Efficient algorithms for CAR models have been widely discussed in particular in the econometric literature (e.g., LeSage and Pace 2009), but these models are not necessarily recommended for irregular lattices (see Wall, 2004 and Martellosio, 2012 for some insights on the implications of autoregressive models).

In **CAR** models, the covariance matrix of random effects **u** can be described as $\lambda(\mathbf{I}-\rho\,\mathbf{W})^{-1}$ where **W** is the (symmetric) adjacency matrix. `HLCor` uses the spectral decomposition of the adjacency matrix, written as boldW=VDV' where **D** is a diagonal matrix of eigenvalues d_i. The covariance of **V'u** is $\lambda(\mathbf{I}-\rho\,\mathbf{D})^{-1}$, which is a diagonal matrix with elements λ_i=λ/$(1-\rho d_i)$. Hence $1/\lambda_i$ is in the linear predictor form α+βd_i This can be used to fit λ and ρ efficiently. A call to `corrHLfit` with the additional argument `init.HLfit=list(rho=0)` should be equivalent in speed and result to the `HLCor` call.

This is fast for small datasets (as in the example below) but more generic maximization algorithms may be preferable for large ones. It is suggested to use `fitme` generally unless one has a large number of small data sets to analyze. A call to `fitme` or `corrHLfit` without that initial value does not use the spectral decomposition. It performs numerical maximization of the likelihood (or restricted likelihood) as function of the correlation parameter ρ. The choice of fitting function may slightly impact the results. The ML fits by `corrHLfit` and `HLCor` should be practically equivalent. The REML fits should slightly differ from each other, due to the fact that the REML approximation for GLMMs does not maximize a single likelihood function.

If `HLCor` is used, the results are reported as the coefficients α (`(Intercept)`) and β (`adjd`) of the predictor for $1/\lambda_i$, in addition to the resulting values of ρ and of the common λ factor.

Different fits may also differ in using or not algorithms that exploit the sparsity of the precision matrix of the autoregressive random effect. By default, spaMM tends to select sparse-precision algorithms for large datasets and large (i.e. many-level) random effects (details are complex). However, for **AR1** models, the dimension of the implied precision matrix is determined by the extreme values of grouping factor (typically interpreted as a time index), as all intermediate values must be considered. Then, the correlation-based algorithms may be more efficient if only a few levels are present in the data, as only a small correlation matrix is required in that case.

References

LeSage, J., Pace, R.K. (2009) Introduction to Spatial Econometrics. Chapman & Hall/CRC.

Martellosio, F. (2012) The correlation structure of spatial autoregressions, Econometric Theory 28, 1373-1391.

Wall M.M. (2004) A close look at the spatial structure implied by the CAR and SAR models: Journal of Statistical Planning and Inference 121: 311-324.

Examples

```
##### AR1 random effect:
ts <- data.frame(lh=lh,time=seq(48)) ## using 'lh' data from stats package
fitme(lh ~ 1 +AR1(1|time), data=ts, method="REML")
# With fixed parameters:
# HLCor(lh ~ 1 +AR1(1|time), data=ts, ranPars=list(ARphi=0.5,lambda=0.25,phi=0.001))

##### CAR random effect:
data("scotlip")
# CAR by Laplace with 'outer' estimation of rho
if (spaMM.getOption("example_maxtime")>0.8) {
  fitme(cases ~ I(prop.ag/10)+adjacency(1|gridcode)+offset(log(expec)),
          adjMatrix=Nmatrix, family=poisson(), data=scotlip)
}

# CAR by Laplace with 'inner' estimation of rho
HLCor(cases ~ I(prop.ag/10)+adjacency(1|gridcode)+offset(log(expec)),
          adjMatrix=Nmatrix, family=poisson(), data=scotlip, method="ML")
```

---

blackcap — *Genetic polymorphism in relation to migration in the blackcap*

Description

This data set is extracted from a study of genetic polymorphisms potentially associated to migration behaviour in the blackcap (Sylvia atricapilla). Across different populations in Europe and Africa, the average migration behaviour was found to correlate with average allele size (dependent on the number of repeats of a small DNA motif) at the locus ADCYAP1, encoding a neuropeptide. This data set is quite small and ill-suited for separating random-effect variance from residual variance. The likelihood surface for the Matérn model actually has local maxima.

Usage

```
data("blackcap")
```

Format

The data frame includes 14 observations on the following variables:

latitude latitude, indeed.

longitude longitude, indeed.

migStatus migration status as determined by Mueller et al, from 0 (resident populations) to 2.5 (long-distance migratory populations)

means Mean allele sizes in each population

pos Numerical index for the populations

Details

Migration status was coded as : pure resident populations as '0', resident populations with some migratory restlessness as '0.5', partial migratory populations as '1', completely migratory populations migrating short-distances as '1.5', intermediate-distance migratory populations as '2' and distinct long-distance migratory populations as '2.5'.

Source

Data from Mueller et al. (2011), including supplementary material now available from https://doi.org/10.1098/rspb.2010.2567.

References

Mueller, J. C., Pulido, F., and Kempenaers, B. 2011. Identification of a gene associated with avian migratory behaviour, Proc. Roy. Soc. (Lond.) B 278, 2848-2856.

Examples

```
## see 'corrHLfit' and 'fixedLRT' for examples involving these data
```

CauchyCorr *Cauchy correlation function and Cauchy formula term*

Description

The Cauchy family of correlation functions is useful to describe spatial processes with power-law decrease of correlation at long distance. It is valid for Euclidean distances in spaces of any dimension, and for great-circle distances on spheres of any dimension. It has a scale parameter (`rho`, as in the Matérn correlation function), a `shape` (or "smoothness", Gneiting 2013) parameter, and a `long`-memory dependence (or, more abstractly, "shape"; Gneiting 2013) parameter (Gneiting and Schlater 2004). The present implementation also accepts a `Nugget` parameter. The family can be invoked in two ways. First, the `CauchyCorr` function evaluates correlations, using distances as input. Second, a term of the form `Cauchy(1|<...>)` in a `formula` specifies a random effect with Cauchy correlation function, using coordinates found in a data frame as input. In the latter case, the correlations between realizations of the random effect for any two observations in the data will be the value of the Cauchy function at the scaled distance between coordinates specified in <...>, using "+" as separator (e.g., `Cauchy(1|latitude + longitude)`).

Usage

```
CauchyCorr(d, rho=1, shape, longdep, Nugget=NULL)
# Cauchy(1|...)
```

Arguments

`d`	Euclidean or great-circle distance
`rho`	The scaling factor for distance, a real >0.
`shape`	The shape (smoothness) parameter, a real 0<.<=2 for Euclidean distances and 0<.<=1 for great-circle distances. Smoothness increases, and fractal dimension decreases, with increasing shape (the fractal dimension of realizations in spaces of dimension d being d+1-shape/2).
`longdep`	The long-memory dependence parameter, a real >0. It gives the exponent of the asymptotic decrease of correlation with distance: the **smaller** `longdep` is, the longer the dependence.
`Nugget`	(Following the jargon of Kriging) a parameter describing a discontinuous decrease in correlation at zero distance. Correlation will always be 1 at $d = 0$, and from which it immediately drops to (1-Nugget). Defaults to zero.
`...`	Names of coordinates, using "+" as separator (e.g., `Matern(1|latitude + longitude)`

Details

The correlation at distance $d > 0$ is

$$(1 - \text{Nugget})(1 + (\rho d)^t extrmshape)^(- textrmlongdep/shape)$$

Value

Scalar/vector/matrix depending on input.

References

Gneiting, T. and Schlater M. (2004) Stochastic models that separate fractal dimension and the Hurst effect. SIAM Rev. 46: 269–282.

Gneiting T. (2013) Strictly and non-strictly positive definite functions on spheres. Bernoulli 19: 1327-1349.

Examples

```
data("blackcap")
fitme(migStatus ~ means+ Cauchy(1|latitude+longitude),data=blackcap,
      fixed=list(longdep=0.5,shape=0.5,rho=0.05))
## The Cauchy family can be used in Euclidean spaces of any dimension:
set.seed(123)
randpts <- matrix(rnorm(20),nrow=5)
distMatrix <- as.matrix(proxy::dist(randpts))
CauchyCorr(distMatrix,rho=0.1,shape=1,longdep=10)
```

COMPoisson *Conway-Maxwell-Poisson (COM-Poisson) GLM family*

Description

The COM-Poisson family is a generalization of the Poisson family which can describe over-dispersed as well as under-dispersed count data. It is indexed by a parameter nu that quantifies such dispersion. It includes the Poisson, geometric and Bernoulli as special (or limit) cases (see Details). The COM-Poisson family is here implemented as a family object, so that it can be fitted by glm, and further used to model conditional responses in mixed models fitted by this package's functions (see Examples). nu is distinct from the dispersion parameter $\nu = 1/\phi$ considered elsewhere in this package and in the GLM literature, as ν affects in a more specific way the log-likelihood. The "canonical link" $\theta(\mu)$ between the canonical GLM parameter θ and the expectation μ of the response does not have a known expression in terms of elementary functions. The link inverse is $\mu(\theta) = \sum_(i = 0)^\infty \lambda^i/(i!)^\nu$ for $\lambda = e^\theta$ (hence the link is here nicknamed "loglambda").

Usage

```
COMPoisson(nu =  stop("COMPoisson's 'nu' must be specified"),
           link = "loglambda")
```

Arguments

link	GLM link function. Cannot be modified.
nu	Under-dispersion parameter. The fitme and corrHLfit functions called with family=COMPoisson() (no given nu value) will estimate this parameter. In other usage of this family, nu must be specified. COMPoisson(nu=1) is the Poisson family.

Details

For nu>1, the distribution is under-dispersed. The limit as nu$\rightarrow\infty$ is the Bernoulli distribution with expectation $\lambda/(1+\lambda)$.

The link inverse function, as shown in Description, involves an infinite summation. In this summation and related computations for the COMPoisson model, the sum can be easily approximated by a finite sum for large nu but not when nu approaches zero. For this reason, the code may fail to fit distributions with nu approaching 0 (strong residual over-dispersion). The case nu=0 itself is the geometric distribution with parameter λ and is fitted by an ad hoc algorithm devoid of such problems. Otherwise, spaMM truncates the sum, and uses numerical integrals to approximate missing terms (which slows down the fitting operation). In addition, it applies an ad hoc continuity correction to ensure continuity of the result in nu=1 (Poisson case). These corrections affect numerical results for the case of residual overdispersion but are negligible for the case of residual underdispersion. Alternatively, spaMM uses Gaunt et al.'s approximations when the condition defined in spaMM.getOption("CMP_asympto_cond") is satisfied. All approximations reduces the accuracy of computations, in a way that can impede the extended Levenberg-Marquardt algorithm sometimes needed by spaMM.

The name `COMP_nu` should be used to set initial values or bounds on `nu` in control arguments of the fitting functions (e.g., `fitme(.,init=list(COMP_nu=1))`). Fixed values should be set by the family argument (`COMPoisson(nu=.)`).

Value

A family object.

References

Gaunt, Robert E. and Iyengar, Satish and Olde Daalhuis, Adri B. and Simsek, Burcin. An asymptotic expansion for the normalizing constant of the Conway–Maxwell–Poisson distribution. Ann Inst Stat Math (2017) doi: 10.1007/s1046301706296.

G. Shmueli, T. P. Minka, J. B. Kadane, S. Borle and P. Boatwright (2005) A useful distribution for fitting discrete data: revival of the Conway-Maxwell-Poisson distribution. Appl. Statist. 54: 127-142.

Sellers KF, Shmueli G (2010) A Flexible Regression Model for Count Data. Ann. Appl. Stat. 4: 943–961

Examples

```
if (spaMM.getOption("example_maxtime")>0.9) {
  # Fitting COMPoisson model with estimated nu parameter:
  #
  data("freight") ## example from Sellers & Shmueli, Ann. Appl. Stat. 4: 943961 (2010)
  fitme(broken ~ transfers, data=freight, family = COMPoisson())

  # glm(), HLCor() and HLfit() handle spaMM::COMPoisson() with fixed overdispersion:
  #
  glm(broken ~ transfers, data=freight, family = COMPoisson(nu=10))
  HLfit(broken ~ transfers+(1|id), data=freight, family = COMPoisson(nu=10),method="ML")

  # Equivalence of poisson() and COMPoisson(nu=1):
  #
  COMPglm <- glm(broken ~ transfers, data=freight, family = poisson())
  coef(COMPglm)
  logLik(COMPglm)
  COMPglm <- glm(broken ~ transfers, data=freight, family = COMPoisson(nu=1))
  coef(COMPglm)
  logLik(COMPglm)
  HLfit(broken ~ transfers, data=freight, family = COMPoisson(nu=1))

}
```

`confint.HLfit` *Confidence intervals*

Description

This computes confidence intervals for a given parameter, based either on parametric bootstrap or, for **fixed-effect** parameters, on the p_v-based approximation of the profile likelihood ratio for this parameter. The profiling is over all other fitted parameters: other fixed effects, as well as variances of random effects and spatial correlations if these were fitted. The bootstrap is performed if the parm argument is a function or a quoted expression or if the boot_args argument is a list. The profile confidence interval is computed if neither of these conditions is true; in that case parm must be the name of a **fixed-effect** coefficient.

Usage

```
## S3 method for class 'HLfit'
confint(object, parm, level=0.95, verbose=TRUE,
                           boot_args=NULL, ...)
```

Arguments

object	An object of class HLfit, as returned by the fitting functions in spaMM.
parm	character, or function, or a quoted expression. If **character**, The name of a parameter to be fitted, or its position in the the object's $fixef vector. Valid names are those of the object's $fixef. If a **function**, it must return the focal parameter estimate from a fit object. If a **quoted expression**, it must likewise extract a parameter estimate from a fit object; this expression must refer to the fitted object as 'hlfit' (see Examples).
level	The coverage of the interval.
verbose	whether to print the interval or not. As the function returns its more extensive results invisibly, this printing is the only visible output.
boot_args	NULL or a list of arguments passed to functions spaMM_boot and boot.ci. It must contain element nsim (for spaMM_boot). The type argument of boot.ci can only be given as element ci_type, to avoid conflict with the type argument of spaMM_boot.
...	Additional arguments (maybe not used, but conforming to the generic definition of confint).

Value

If a bootstrap was performed, the result of the boot.ci call is returned, with basic and percentile intervals, and modified call element. Otherwise, a list including the confidence interval for the target parameter, and the fits lowerfit and upperfit giving the profile fits at the confidence bounds. This is returned invisibly.

Examples

```
data("wafers")
wfit <- HLfit(y ~X1+(1|batch),family=Gamma(log),data=wafers,method="ML")
confint(wfit,"X1")  # profile CI
if (spaMM.getOption("example_maxtime")>30) {
```

```
    confint(wfit,"X1", boot_args=list(nsim=99)) # bootstrap CI induced by 'boot_args'
    # bootstrap CI induced by 'parm' being a function:
    confint(wfit,parm=function(v) fixef(v)[["X1"]],
            boot_args=list(nb_cores=10, nsim=199))
    # Same effect if 'parm' is a quoted expression in terms of 'hlfit':
    confint(wfit,parm=quote(fixef(hlfit)[["X1"]]),
            boot_args=list(nb_cores=10, nsim=199))
}
```

---

convergence *Assessing convergence for fitted models*

Description

spaMM fits may produce convergence warnings coming from `.check_conv_glm_reinit()`. These can generally be ignored (particularly when they show a small criterion, <1e-5).

spaMM fits may also produce messages pointing to slow convergence and drawing users here. These do not necessarily mean the fit is incorrect. Rather, they suggest that another fitting strategy could be tried. Keep in mind that several parameters (notably the dispersion parameters: the variance of random effects and the residual variance parameter, if any) can be estimated either by the iterative algorithms implemented in `HLfit`, or by generic optimization methods. In my experience, slow convergence happens in certain cases where a large random-effect variance is considered by the algorithm used. The development of the `fitme` function aims to provide full control of the selection of algorithms. For example, if two random meffects are fitted, then `init=list(lambda=c(NA,NaN))` enforces generic optimization for the first variance and iterative algorithms for the second. `init=list(lambda=c(0.1,NaN))` has the same effect and additionnally provides control of the initial value for optimization (whereas `init.HLfit=list(lambda=c(NA,0.1))` will provide control of the initial value for iterations).

If the iterative algorithm is being used, then it is worth trying to use the generic optimization methods. In particular, if you used `HLfit`, try using `fitme`; if you already use `fitme`, try to enforce optimization of the random-effect variance(s). Conversely, if generic optimization is being used, the maximum lambda value could be controlled (say, `upper=list(lambda=c(10,NA))`), or the iterative algorithm can be called.

How to know which algorithm has been selected for each parameter? `fitme(.,verbose=c(verbose=TRUE))` shows sucessive values of the variables estimated by optimization (See Examples; if no value appears, then all are estimated by iterative methods).

Examples

```
## Not run:
air <- data.frame(passengers = as.numeric(AirPassengers),
                  year_z = scale(rep(1949:1960, each = 12)),
                  month = factor(rep(1:12, 12)))
air$time <- 1:nrow(air)
# Use verbose to find that lambda is estimated by optimization
fitme(passengers ~ month * year_z + AR1(1|time), data = air,
      verbose=c(TRACE=TRUE))
```

```
# Use init to enforce iterative algorithm for lambda estimation:
fitme(passengers ~ month * year_z + AR1(1|time), data = air,
      verbose=c(TRACE=TRUE), init=list(lambda=NaN))
# Use init to enforce generic optimization for lambda estimation,
#   and control initial value:
fitme(passengers ~ month * year_z + AR1(1|time), data = air,
      verbose=c(TRACE=TRUE), init=list(lambda=0.1))

## End(Not run)
```

---

corMatern *Matern Correlation Structure as a corSpatial object*

Description

This implements the Matérn correlation structure (see Matern) for use with lme or glmmPQL. Usage is as for others corSpatial objects such as corGaus or corExp, except that the Matérn family has an additional parameter. This function was defined for comparing results obtained with corrHLfit to those produced by lme and glmmmPQL. There are problems in fitting (G)LMMs in the latter way, so it is not a recommended practice.

Usage

```
corMatern(value = c(1, 0.5), form = ~1, nugget = FALSE, nuScaled = FALSE,
          metric = c("euclidean", "maximum", "manhattan"), fixed = FALSE)
```

Arguments

value
: An optional vector of parameter values, with serves as initial values or as fixed values depending on the fixed argument. It has either two or three elements, depending on the nugget argument.

  If nugget is FALSE, value should have two elements, corresponding to the "range" and the "smoothness" ν of the Matérn correlation structure. If value has zero length, the default is a range of 90% of the minimum distance and a smoothness of 0.5 (exponential correlation). **Warning**: the range parameter used in corSpatial objects is the inverse of the scale parameter used in MaternCorr and thus they have opposite meaning despite both being denoted ρ elsewhere in this package or in nlme literature.

  If nugget is TRUE, meaning that a nugget effect is present, value can contain two or three elements, the first two as above, the third being the "nugget effect" (one minus the correlation between two observations taken arbitrarily close together). If value has length zero or two, the nugget defaults to 0.1. The range and smoothness must be greater than zero and the nugget must be between zero and one.

form
: (Pasted from corSpatial) a one sided formula of the form ~ S1+...+Sp, or ~ S1+...+Sp | g, specifying spatial covariates S1 through Sp and, optionally, a grouping factor g. When a grouping factor is present in form, the correlation

structure is assumed to apply only to observations within the same grouping level; observations with different grouping levels are assumed to be uncorrelated. Defaults to ~ 1, which corresponds to using the order of the observations in the data as a covariate, and no groups.

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

`nuScaled` If `nuScaled` is set to `TRUE` the "range" parameter ρ is divided by $2\sqrt{\nu}$. With this option and for large values of ν, `corMatern` reproduces the calculation of `corGaus`. Defaults to `FALSE`, in which case the function compares to `corGaus` with range parameter $2(\sqrt{\nu})\rho$ when ν is large.

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

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

Details

This function is a constructor for the `corMatern` class, representing a Matérn spatial correlation structure. See `MaternCorr` for details on the Matérn family.

Value

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

Note

The R and C code for the methods for `corMatern` objects builds on code for `corSpatial` objects, by D.M. Bates, J.C. Pinheiro and S. DebRoy, in a circa-2012 version of nlme.

References

Mixed-Effects Models in S and S-PLUS, José C. Pinheiro and Douglas M. Bates, Statistics and Computing Series, Springer-Verlag, New York, NY, 2000.

See Also

`glmmPQL`, `lme`

Examples

```
## LMM
data("blackcap")
blackcapD <- cbind(blackcap,dummy=1) ## obscure, isn't it?
```

```
## With method= 'ML' in lme, The correlated random effect is described
##  as a correlated residual error and no extra residual variance is fitted:
nlme::lme(fixed = migStatus ~ means, data = blackcapD, random = ~ 1 | dummy,
    correlation = corMatern(form = ~ longitude+latitude  | dummy),
    method = "ML", control=nlme::lmeControl(sing.tol=1e-20))

## Binomial GLMM
if (spaMM.getOption("example_maxtime")>32) {
 data("Loaloa")
 LoaloaD <- cbind(Loaloa,dummy=1)
 MASS::glmmPQL(fixed =cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI,
        data = LoaloaD, random = ~ 1 | dummy,family=binomial,
        correlation = corMatern(form = ~ longitude+latitude | dummy))
}
```

---

corrHLfit *Fits a mixed model, typically a spatial GLMM.*

Description

This was the first function for fitting all spatial models in spaMM, and is still fully functional, but it is recommended to use fitme which has different defaults and generally selects more efficient fitting methods, and will handle all classes of models that spaMM can fit, including non-spatial ones. corrHLfit performs the joint estimation of correlation parameters, fixed effect and dispersion parameters.

Usage

```
corrHLfit(formula, data, init.corrHLfit = list(),
          init.HLfit = list(), ranFix = list(), lower = list(),
          upper = list(),
          objective = NULL, resid.model = ~1,
          control.dist = list(), control.corrHLfit = list(),
          processed = NULL, family = gaussian(), method="REML",
          nb_cores = NULL, ...)
```

Arguments

formula
: Either a linear model formula (as handled by various fitting functions) or a predictor, i.e. a formula with attributes (see Predictor and examples below). See Details in spaMM for allowed terms in the formula.

data
: A data frame containing the variables in the response and the model formula.

init.corrHLfit
: An optional list of initial values for correlation and/or dispersion parameters, e.g. list(rho=1,nu=1,lambda=1,phi=1) where rho and nu are parameters of the Matérn family (see Matern), and lambda and phi are dispersion parameters (see Details in spaMM for the meaning of these parameters). All are optional, but giving values for a dispersion parameter changes the ways it is estimated (see Details). rho may be a vector (see make_scaled_dist) and, in that case, it is

	possible that some or all of its elements are NA, for which corrHLfit substitutes automatically determined values.
init.HLfit	See identically named HLfit argument.
ranFix	A list similar to init.corrHLfit, but specifying fixed values of the parameters not estimated. See ranFix for further information.
lower	An optional (sub)list of values of the parameters specified through init.corrHLfit, in the same format as init.corrHLfit, used as lower values in calls to optim. See Details for default values.
upper	Same as lower, but for upper values.
objective	For development purpose, not documented (this had a distinct use in the first version of spaMM, but has been deprecated as such).
resid.model	See identically named HLfit argument.
control.dist	See control.dist in HLCor
control.corrHLfit	This may be used control the optimizer. See spaMM.optionsfor default values.
processed	For programming purposes, not documented.
family	Either a family or a multi value.
method	Character: the fitting method to be used, such as "ML", "REML" or "PQL/L". "REML" is the default. Other possible values of HLfit's method argument are handled.
nb_cores	**Not yet operative**, only for development purposes. Number of cores to use for parallel computations.
...	Optional arguments passed to HLCor, HLfit or mat_sqrt, for example the distMatrix argument of HLCor. Arguments that do not fit within these functions are detected and a warning is issued.

Details

For approximations of likelihood, see method. For the possible structures of random effects, see random-effects,

By default corrHLfit will estimate correlation parameters by maximizing the objective value returned by HLCor calls wherein the dispersion parameters are estimated jointly with fixed effects for given correlation parameters. If dispersion parameters are specified in init.corrHLfit, they will also be estimated by maximizing the objective value, and HLCor calls will not estimate them jointly with fixed effects. This means that in general the fixed effect estimates may vary depending on init.corrHLfit when any form of REML correction is applied.

Correctly using corrHLfit for likelihood ratio tests of fixed effects may then be tricky. It is safe to perform full ML fits of all parameters (using method="ML") for such tests (see Examples). The higher level function fixedLRT is a safe interface for likelihood ratio tests using some form of REML estimation in corrHLfit.

attr(<fitted object>,"optimInfo")$lower and ...$upper gives the lower and upper bounds for optimization of correlation parameters. These are the default values if the user did not provide explicit values. For the adjacency model, the default values are the inverse of the maximum and minimum eigenvalues of the adjMatrix. For the Matérn model, the default values are not so easily summarized: they are intended to cover the range of values for which there is statistical information to distinguish among them.

Value

The return value of an HLCor call, with additional attributes. The HLCor call is evaluated at the estimated correlation parameter values. These values are included in the return object as its $corrPars member. The attributes added by corrHLfit include the original call of the function (which can be retrived by getCall(<fitted object>), and information about the optimization call within corrHLfit.

See Also

See more examples on data set Loaloa, to compare fit times by corrHLfit and fitme. See fixedLRT for likelihood ratio tests.

Examples

```
# Example with an adjacency matrix (autoregressive model):
if (spaMM.getOption("example_maxtime")>0.7) {
  corrHLfit(cases~I(prop.ag/10) +adjacency(1|gridcode)+offset(log(expec)),
          adjMatrix=Nmatrix,family=poisson(),data=scotlip,method="ML")
}

#### Examples with Matern correlations
## A likelihood ratio test based on the ML fits of a full and of a null model.
if (spaMM.getOption("example_maxtime")>1.4) {
 data("blackcap")
 (fullfit <- corrHLfit(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
                   method="ML") )
 (nullfit <- corrHLfit(migStatus ~ 1 + Matern(1|latitude+longitude),data=blackcap,
                   method="ML",init.corrHLfit=list(phi=1e-6)))
 ## p-value:
 1-pchisq(2*(logLik(fullfit)-logLik(nullfit)),df=1)
}
```

---

corrMatrix *Using a corrMatrix argument*

Description

corrMatrix is an argument of HLCor, of calls dist or matrix, with is used if the model formula contains a term of the form corrMatrix(1|<...>). It describes a correlation matrix, possibly as a dist object. A covariance matrix can actually be passed through this argument, but then it must be a full matrix, not a dist object. The way the rows and columns of the matrix are matched to the rows of the data depends on the nature of the grouping term <...>.

Details

The simplest case is illustrated in the first two examples below: the grouping term is identical to a single variable which is present in the data, whose levels match the rownames of the corrMatrix. As illustrated by the second example, the order of the data does not matter in that case, because the factor levels are used to match the data rows to the appropriate row and columns of the corrMatrix. The corrMatrix may even contain rows (and columns) in excess of the levels of the grouping term, in which case these rows are ignored.

These convenient properties no longer hold when the grouping term is not a single variable from the data (third example below), or when its levels do not correspond to row names of the matrix. In these cases, (1) no attempt is made to match the data rows to the row and column names of the corrMatrix. Such attempt could succeed only if the user had given names to the matrix matching those that the called function could create from the information in the data, in which case the user should find easier to specify a single variable that can be matched; (2) the order of data and corrMatrix matter; Internally, a single factor variable is constructed from all levels of the variables in the grouping term (i.e., from all levels of latitude and longitude, in the third example), with levels 1,2,3... that are matched to rows 1,2,3... of the corrMatrix. Thus the first row of the data is always associated to the first row of the matrix; (3) further, the dimension of the matrix must match the number of levels implied by the grouping term. For example, one might consider the case of 14 response values but of correlations between only 7 levels of a random effect, with two responses for each level. Then the matrix must be of dimension 7x7.

Examples

```
data("blackcap")
## Here we manually reconstruct the correlation matrix
##  of the ML fit produced by corrHLfit:
MLcorMat <- MaternCorr(proxy::dist(blackcap[,c("latitude","longitude")]),
                        nu=0.6285603,rho=0.0544659)
blackcap$name <- as.factor(rownames(blackcap))
## (1) Single variable present in the data
HLCor(migStatus ~ means+ corrMatrix(1|name),data=blackcap,
      corrMatrix=MLcorMat,method="ML")
## (2) Same, permuted: still gives correct result
perm <- sample(14)
# Permuted matrix (with permuted names)
pmat <- as.matrix(MLcorMat)[perm,perm]
HLCor(migStatus ~ means+ corrMatrix(1|name),data=blackcap,
      corrMatrix=as.dist(pmat),method="ML")
## (3) Other grouping terms (note the messages):
HLCor(migStatus ~ means+ corrMatrix(1|latitude+longitude),data=blackcap,
      corrMatrix=MLcorMat,method="ML")
```

---

corr_family   corr_family *objects*

Description

corr_family objects provide a convenient way to implement correlation models handled by spaMM, analogous to family objects. These objects are undocumented (but there are documentation pages for each of the models implemented).

Usage

```
# Matern(...)          # see help(Matern)
# Cauchy(...)          # see help(Cauchy)
# corrMatrix(...)      # see help(corrMatrix)
# AR1(...)             # see help(AR1)
# adjacency(...)       # see help(adjacency)
# IMRF(...)            # see help(IMRF)
## S3 method for class 'corr_family'
print(x,...)
```

Arguments

x	corr_family object.
...	arguments that may be needed by some corr_family object or some print method.

---

covStruct *Specifying correlation structures*

Description

covStruct is a formal argument of HLCor, also handled by fitme and corrHLfit, that allows one to specify the correlation structure for different types of random effects, It is an alternative to other ad hoc formal arguments such as corrMatrix or adjMatrix. It replaces the deprecated function Predictor(...) which has served as an interface for specifying the design matrices for random effects in early versions of spaMM.

The main use of covStruct is to specify the correlation matrix of levels of a given random effect term, or its inverse (a precision matrix). Assuming that the design matrix of each random effect term follows the structure **ZAL** described in random-effects, it is thus an indirect way of specifying the "square root" **L** of the correlation matrix. The optional **A** factor can also be given by the optional "AMatrices" attribute of covStruct.

covStruct is a list of matrices with names specifying the type of matrix considered: covStruct=list(corrMatrix=<some matrix>) or covStruct=list(adjMatrix=<some matrix>), where the "corrMatrix" or "adjMatrix" labels are used to specify the type of information provided (accordingly, the names can be repeated: covStruct=list(corrMatrix=<.>,corrMatrix=<.>)).
The covariance structure of a corrMatrix(1|<grouping factor>) formula term can be specified in two ways (see Examples): either by a correlation matrix factor (covStruct=list(corrMatrix=<some matrix>)), or by a precision matrix factor **Q** such that the covariance factor is $\lambda\mathbf{Q}^{-1}$, using the type name "precision": covStruct=list(precision=<some matrix>). The function as_precision can be used to perform the conversion from correlation information to precision factor (using a crude solve() that may not always be efficient), but fitting functions may also perform such conversions automatically.

NULL list members may be necessary, e.g.
covStruct=list(corrMatrix=<.>,"2"=NULL,corrMatrix=<.>))
when correlations matrices are required only for the first and third random effect.

"AMatrices" is a list of matrices. The names of elements of the list does not matter, but the *i*th A matrix, and its row names, should match the *i*th **Z** matrix, and its column names. This implies that NULL list members may be necessary, as for the covStruct list.

Usage

```
as_precision(corrMatrix)
```

Arguments

corrMatrix	Correlation matrix, specified as matrix or as dist object

Details

covStruct can also be specified as a list with an optional "types" attribute, e.g. structure(list(<some matrix>,types="corrMatrix")).

Value

as_precision returns a list with addition class precision and with single element a symmetric matrix of class dsCMatrix.

See Also

pedigree for a type of applications where declaring a precision matrix is useful.

Examples

```
## Not run:
data("blackcap")
# a 'dist' object can be used to specify a corrMatrix:
MLdistMat <- MaternCorr(proxy::dist(blackcap[,c("latitude","longitude")]),
                        nu=0.6285603,rho=0.0544659) # a 'dist' object!
blackcap$name <- as.factor(rownames(blackcap))
fitme(migStatus ~ means + corrMatrix(1|name), data=blackcap,
      corrMatrix=MLdistMat)

#### Same result by different input and algorithm:
fitme(migStatus ~ means + corrMatrix(1|name), data=blackcap,
      covStruct=list(precision=as_precision(MLdistMat)))

# Manual version of the same:
as_mat <- proxy::as.matrix(MLdistMat, diag=1)
prec_mat <- solve(as_mat) ## precision factor matrix
fitme(migStatus ~ means + corrMatrix(1|name), data=blackcap,
      covStruct=list(precision=prec_mat))

# Since no correlation parameter is estimated,
# HLcor(., method="ML")  is here equivalent to fitme()

## End(Not run)
```

dopar *Interface for parallel computations*

Description

An interface to apply some function fn in parallel on columns of a matrix. Depending on the nb_cores argument, parallel or serial computation is performed, and depending whether the doSNOW package is attached, foreach or pbapply is called (doSNOW allows more efficient load balancing than pbapply). It wraps these calls, handling all cluster management. It is used internally in spaMM, but is not logically restricted to mixed-effect applications, hence it can be used more widely.

Usage

```
dopar(newresp, fn, nb_cores = NULL, fit_env, control = list(),
      cluster_args = NULL, debug. = FALSE, iseed = NULL,
      showpbar = eval(spaMM.getOption("barstyle")),
      pretest_cores =NULL, ...)
```

Arguments

newresp	A matrix on whose columns fn will be applied (e.g., as used internally in spaMM, the return value of a simulate.HLfit() call).
fn	Function, whose first argument is named y, to be applied to each column of newresp.
nb_cores	Integer. Number of cores to use for parallel computations. If >1, a cluster of nb_cores nodes is used. Otherwise, no parallel computation is performed.
fit_env	A environment, or a list, containing variables to be exported on the nodes of the cluster (by parallel::clusterExport).
control	A list. In particular if the doSNOW package is attached, foreach is called with default arguments including i = 1:ncol(newresp), .combine = "cbind", .inorder = TRUE, .errorhandling = "remove", .packages = "spaMM". control may be used to provide non-default values of these arguments. For example, .errorhandling = "pass" is useful to get error messages from the nodes. If doSNOW is not attached, the result is still in the format returned by foreach with default .combine="cbind" or possible non-default .combine="rbind".
cluster_args	A list of arguments passed to parallel::makeCluster. E.g., outfile="log.txt" may be useful to collect output from the nodes.
debug.	For debugging purposes. Effect, if any, is to be defined by the fn as provided by the user.
iseed	Integer, or NULL. If an integer, parallel::clusterSetRNGStream(cl = cl,iseed) is called, setting and initializing "L'Ecuyer-CMRG" random-number generator on the nodes. This is useful if random numbers are used on the nodes (see clusterSetRNGStream). If iseed is NULL, the default generator is selected on each node, where its seed is not controlled.

showpbar — Controls display of progress bar. See barstyle option for details.

pretest_cores — A function to run on the cores before running fn. It may be used to check that all arguments of the fn can be evaluated in the cores' environments (the internal function .pretest_fn_on_cores provides an example).

... — Further arguments to be passed (unevaluated) to fn.

Value

The result of calling foreach or pbapply, as dependent on the control argument. A side-effect of dopar is to show a progress bar that informs about the type of parallelisation performed: "P" for parallel computation via foreach and doSNOW, "p" for parallel computation via pbapply, "s" for serial computation via pbapply.

Examples

```
## See source code of spaMM_boot()
```

---

eval_replicate — *Evaluating bootstrap replicates*

Description

eval_replicate is the default simuland function applied to simulated bootstrap samples by likelihood-ratio testing functions (fixedLRT,LRT,anove.HLfit). This documentation presents the requirements and possible features of this function and of possible user-defined alternatives.

An alternative function spaMM:::.eval_replicate2 is also provided. It is slower, as it refits the models compared with different initial values for random-effect parameters, which is useful in some difficult cases where initial values matter. The eval_replicate function may also refit the "full" models with different initial values when the logLik of the refitted full model is subtstantially lower than that of the refitted null model. "Substantially" means that a tolerance of 1e-04 is applied to account for inaccuracies of numerical maximization.

Usage

```
eval_replicate(y)
```

Arguments

y — a response vector on which a previously fitted model may be refitted.

Details

likelihood-ratio testing functions have a debug. argument whose effect depends on the simuland function. The default behaviour is thus defined by eval_replicate, as: if debug.=TRUE, upon error in the fitting procedures, dump.frames will be called, in which case **a dump file will be written on disk**; and a **list** with debugging information will be returned (so that, say, pbapply will not return a matrix). This behaviour may change in later versions, so non-default debug. values should not be used in reproducible code. In serial computation, debug.=2 may induce a stop; this should not happen in parallel computation because the calling functions check debug. values to prevent it.

Essential information such as the originally fitted models is passed to the function not as arguments but through its environment, which is controlled by the calling functions (see the eval_replicate source code to know which are these arguments). Users should thus not assume that they can control their own simuland function's environment as this environment will be altered.

Advanced users can define their own simuland function. The eval_replicate source code provides a template showing how to use the function's environment. The Example below illustrates another approach augmenting eval_replicate. A further example is provided in the file tests/testthat/test-LRT-boot.R, using ... to pass additional arguments beyond response values.

Value

A vector of the form c(full=logLik(<refitted full model>),null=logLik(<refitted null model>); or possibly in debugging contexts, a list with the same elements each with some additional information provided as attribute.

See Also

Calling functions fixedLRT, LRT.

Examples

```
## Not run:
# Simple wrapper enhancing the default 'simuland'
#  with a call to some obscure option, and dealing with
#  the need to pass the environment assigned to 'simuland'
eval_with_opt <- function(y) {
  spaMM.options(some_obscure_option="some_obscure_value")
  eval_rep <- spaMM:::.eval_replicate
  environment(eval_rep) <- parent.env(environment()) # passing the environment
  eval_rep(y)
}

## End(Not run)
```

---

external-libraries *Installing external libraries*

Description

spaMM is conceived to minimize installation issues but it nevertheless suggests using some external libraries. These are all accessed through R packages so their installation should be easy when installing binary packages. The Details below give hints for installing packages from source. For all cases not considered below, help yourself.

Details

The ROI.plugin.glpk package requires the Rglpk package, which itself requires the external glpk library. For the latter, Debian-ists and alikes should sudo apt-get install libglpk-dev. MacOSX users should brew install glpk if using brew; Windows users may need to install glpk from https://sourceforge.net/projects/winglpk/.

The nloptr package requires the external NLopt library. If you need to install it from source on Windows and depending on the version of g++ included in your installed version of the Rtools, you may need to be a geek, as available precompiled static libraries may not link.

---

extractors *Functions to extract various components of a fit*

Description

formula extracts the model formula. family extracts the response family. terms extracts the **fixed-effect** terms. nobs returns the length of the response vector. logLik extracts the log-likelihood (exact or approximated). dev_resids returns a vector of squared (unscaled) deviance residuals (the summands in McCullagh and Nelder 1989, p. 34). deviance returns the sum of squares of these (unscaled) deviance residuals, that is (consistently with stats::deviance) the unscaled deviance. fitted extracts fitted values (see fitted.values). residuals extracts residuals of the fit. response extracts the response (as a vector). fixef extracts the fixed effects coefficients, β. ranef extracts the predicted random effects, **Lv** (default since version 1.12.0), or **u** (see Details in HLfit for definitions), print.ranef controls their printing. getDistMat returns a distance matrix for a Matérn correlation model. get_RLRTSim_args returns a list of arguments suitable for calls to RLRsim::RLRTSim()

Usage

```
## S3 method for class 'HLfit'
formula(x, which="hyper", ...)
## S3 method for class 'HLfit'
family(object, ...)
## S3 method for class 'HLfit'
terms(x, ...)
## S3 method for class 'HLfit'
nobs(object, ...)
## S3 method for class 'HLfit'
logLik(object, which, ...)
## S3 method for class 'HLfit'
fitted(object, ...)
## S3 method for class 'HLfit'
fixef(object, ...)
## S3 method for class 'HLfit'
ranef(object, type = "correlated", ...)
## S3 method for class 'ranef'
print(x, max.print = 40L, ...)
```

```
## S3 method for class 'HLfit'
deviance(object, ...)
## S3 method for class 'HLfit'
residuals(object, type = c("deviance", "pearson", "response"), ...)
getDistMat(object, scaled=FALSE, which = 1L)
response(object,...)
dev_resids(object,...)
get_RLRTSim_args(object,...)
```

Arguments

object	An object of class HLfit, as returned by the fitting functions in spaMM.
type	For ranef, use type="correlated" (default) to display the correlated random effects (**Lv**), whether in a spatial model, or a random- coefficient model. Use type="uncorrelated" to pretty-print the elements of the <object>$ranef vector (**u**). For residuals, the type of residuals which should be returned. The alternatives are: "deviance" (default), "pearson", and "response".
which	For logLik, the name of the element of the APHLs list to return (see Details for any further possibility). The default depends on the fitting method. In particular, if it was REML or one of its variants, the function returns the log restricted likelihood (exact or approximated). For getDistMat, an integer, to select a random effect from several for which a distance matrix may be constructed. For formula, by default the model formula with non-expanded multIMRF random-effect terms is returned, while for which="" a formula with multIMRF terms expanded as IMRF terms is returned.
scaled	If FALSE, the function ignores the scale parameter rho and returns unscaled distance.
x	For print.ranef: the return value of ranef.HLfit.
max.print	Controls options("max.print") locally.
...	Other arguments that may be needed by some method.

Details

See residuals.glm for more information about the types of residuals.

With which="LogL_Lap", logLik() returns a Laplace approximation of log-likelihood based on the observed Hessian, rather than the expected Hessian. This is implemented only for the case family=Gamma(log), for demonstration purposes.

Value

Return values are numeric (for logLik), vectors (most cases), matrices or dist objects (for getDistMat), or a family object (for family). ranef returns a list of vectors or matrices (the latter for random-coefficient terms). terms returns an object of class c("terms","formula") which contains the *terms* representation of a symbolic model. See terms.object for its structure.

get_RLRTSim_args extracts a list of arguments suitable for a call to RLRsim::RLRTSim() for a small-sample test of the presence of a random effect by an efficient simulation procedure. The test can be run by
do.call("RLRTSim",<get_RLRTSim_args return value>).

References

McCullagh, P. and Nelder J. A. (1989) Generalized linear models. Second ed. Chapman & Hall: London.

Lee, Y., Nelder, J. A. (2001) Hierarchical generalised linear models: A synthesis of generalised linear models, random-effect models and structured dispersions. Biometrika 88, 987-1006.

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006) Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

See Also

See `hatvalues` to extract leverages, `get_matrix` to extract the model matrix and derived matrices, and `vcov.HLfit` to extract covariances matrices from a fit.

Examples

```
data("wafers")
m1 <- HLfit(y ~ X1+X2+(1|batch), resid.model = ~ 1,
            data=wafers, method="ML")
fixef(m1)
ranef(m1)
```

`fitme` *Fitting function for fixed- and mixed-effect models with GLM response.*

Description

This is a common interface for fitting most models that spaMM can fit, from linear models to mixed models with non-gaussian random effects, therefore substituting to `corrHLfit`, `HLCor` and `HLfit`. By default, it uses ML rather than REML (differing in this respect from the other fitting functions). It may use “outer optimization”, i.e., generic optimization methods for estimating all dispersion parameters, rather than the iterative methods implemented in `HLfit`. The results of REML fits of non-gaussian mixed models by these different methods may (generally slightly) differ. Outer optimization should generally be faster than the alternative algorithms for large data sets when the residual variance model is a single constant term (no structured dispersion). For mixed models, `fitme` by default tries to select the fastest method when both can be applied, but precise decision criteria are subject to change in the future. `corrHLfit` (with non-default arguments to control the optimization method most suitable to a particular problem) may be used to ensure better consistency over successive versions of `spaMM`.

Usage

```
fitme(formula, data, family = gaussian(), init = list(), fixed = list(),
      lower = list(), upper = list(), resid.model = ~1, init.HLfit = list(),
      control = list(), control.dist = list(), method = "ML",
      HLmethod = method, processed = NULL, nb_cores = NULL, objective = NULL,
      ...)
```

Arguments

- **formula**: Either a linear model formula (as handled by various fitting functions) or a predictor, i.e. a formula with attributes (see Predictor and examples below). See Details in spaMM for allowed terms in the formula.
- **data**: A data frame containing the variables in the response and the model formula.
- **family**: Either a family or a multi value.
- **init**: An optional list of initial values for correlation and/or dispersion parameters and/or response family parameters, e.g. `list(rho=1,nu=1,lambda=1,phi=1)` where `rho` and `nu` are parameters of the Matérn family (see Matern), and `lambda` and `phi` are dispersion parameters (see Details in spaMM for the meaning of these parameters). All are optional, but giving values for a dispersion parameter changes the ways it is estimated (see Details and Examples). `rho` may be a vector (see make_scaled_dist) and, in that case, it is possible that some or all of its elements are `NA`, for which `fitme` substitutes automatically determined values.
- **fixed**: A list similar to `init`, but specifying fixed values of the parameters not estimated. See fixed for further information; and keep in mind that fixed fixed-effect coefficients can be passed as the `etaFix` argument as part of the '...'.
- **lower**: An optional (sub)list of values of the parameters specified through `init`, in the same format as `init`, used as lower values in calls to `optim`. See Details for default values.
- **upper**: Same as `lower`, but for upper values.
- **resid.model**: See identically named HLfit argument.
- **init.HLfit**: See identically named HLfit argument.
- **control**: A list of control parameters, with two possible elements:
  - `nloptr`, itself a list of control parameters to be copied in the `opts` argument of nloptr. Default controls are given by `spaMM.getOption('nloptr')`
  - `refit`, a boolean, or a list of booleans with possible elements `$phi`, `$lambda` and `$ranCoefs`. If either element is set to TRUE, then the corresponding parameters are refitted by the internal `HLfit` methods (see Details). If `$refit` is a single boolean, it affects of parameters. By default only `lambda` is refitted, but this default may change in the future.
- **control.dist**: See `control.dist` in HLCor
- **method, HLmethod**: Character: the fitting method to be used, such as `"ML"`, `"REML"` or `"PQL/L"`. `"ML"` is the default, in contrast to `"REML"` for the other fitting functions. Other possible values of `HLfit`'s `method` argument are handled.
- **nb_cores**: **Not yet operative**, only for development purposes. Number of cores to use for parallel computations.
- **processed**: For programming purposes, not documented.
- **objective**: For development purpose, not documented.
- **...**: Optional arguments passed to HLCor, HLfit or mat_sqrt, for example the `distMatrix` argument of `HLCor`.

Details

For approximations of likelihood, see method. For the possible structures of random effects, see random-effects,

For phi, lambda, and ranCoefs, fitme may or may not use the internal fitting methods of HLfit. The latter methods are well suited for structured dispersion models, but require computations which can be slow for large datasets. Therefore, fitme tends to outer-optimize by default for large datasets, unless there is a non-trivial resid.model. The precise criteria for selection of default method by fitme are liable to future changes.

Further, the internal fitting methods of HLfit also provide some more information such as the “cond. SE” (about which see warning in Details of HLfit). To force the evaluation of such information after an outer-optimization by a fitme call, use the control$refit argument (see Example). Alternatively (and possibly of limited use), one can force inner-optimization of lambda for a given random effect, or of phi, by setting it to NaN in init (see Example using ‘blackcap’ data). The same syntax may be tried for phi.

Value

The return value of an HLCor or an HLfit call, with additional attributes. The HLCor call is evaluated at the estimated correlation parameter values. These values are included in the return object as its $corrPars member. The attributes added by fitme include the original call of the function (which can be retrived by getCall(<fitted object>), and information about the optimization call within fitme.

Examples

```
## See many examples in help("spaMM")

## Contrasting different optimization methods:
# We simulate Gamma deviates with mean mu=3 and variance=2,
#  ie. phi= var/mu^2= 2/9 in the (mu, phi) parametrization of a Gamma
#  GLM; and shape=9/2, scale=2/3 in the parametrisation of rgamma().
#  Note that phi is not equivalent to scale:
#  shape = 1/phi and scale = mu*phi.
set.seed(123)
gr <- data.frame(y=rgamma(100,shape=9/2,scale=2/3))
# Here fitme uses HLfit methods which provide cond. SE for phi by default:
fitme(y~1,data=gr,family=Gamma(log))
# To force outer optimization of phi, use the init argument:
fitme(y~1,data=gr,family=Gamma(log),init=list(phi=1))
# To obtain cond. SE for phi after outer optimization, use the 'refit' control:
fitme(y~1,data=gr,family=Gamma(log),,init=list(phi=1),
      control=list(refit=list(phi=TRUE))) ## or ...refit=TRUE...

## Outer-optimization is not necessarily the best way to find a global maximum,
#  particularly when there is little statistical information in the data:
if (spaMM.getOption("example_maxtime")>1.6) {
  data("blackcap")
  fitme(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap) # poor
  #  Compare with the following two ways of avoiding outer-optimization of lambda:
  corrHLfit(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
```

```
            method="ML")
  fitme(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
        init=list(lambda=NaN))
}

## see help("COMPoisson"), help("negbin"), help("Loaloa"), etc., for further examples.
```

---

fixed *Fixing some parameters*

Description

The fitting functions allow some parameters to be fixed rather than estimated. Fixed-effect coefficients can be set by by way of the `etaFix` argument (linear predictor coefficents) for all fitting functions. Random-effect parameters can be set by an argument with a different name for the different fitting functions: `fixed` for `fitme`, `ranFix` for `HLfit` and `corrHLfit`, and `ranPars` for `HLCor`. This diversity of names may be confusing, but keep in mind that `ranFix` allows one to fix parameters that `HLfit` and `corrHLfit` would otherwise estimate, while `ranPars` can be used to set correlation parameters that `HLCor` does not estimate but nevertheless requires (e.g., Matérn parameters).

Details

etaFix is a list with principal element `beta`, which should be a vector of (a subset of) the coefficients (β) of the fixed effects, with names as shown in a fit without such given values. If REML is used to fit random effect parameters, then `etaFix` affects by default the REML correction for estimation of dispersion parameters, which depends only on which β coefficients are estimated rather than given. This default behaviour will be overridden whenever a non-null `REMLformula` is provided to the fitting functions (see Example). `REMLformula` is the preferred way to control non-standard REML fits. Alternatively, with a non-NULL `etaFix$beta`, REML can also be performed as if all β coefficients were estimated, by adding attribute `keepInREML=TRUE` to `etaFix$beta`. Using an `REMLformula` will override such a specification.

Despite its different name for different fitting functions, the argument for fixing random-effect parameters has a common syntax for all functions. It is a list, with the following possible elements, whose nature is further detailed below: **phi** (variance of residual error, for gaussian and Gamma HGLMs), **lambda** (random-effect variances, except for random-coefficient terms), **ranCoefs** (random-coefficient parameters), and **corrPars** (correlation parameters, when handled by the fitting function). Individual correlation parameters such as **rho, nu, Nugget, ARphi**... are also possible When there is no ambiguity as to which random effect these correlation parameters apply. This was the original syntax, conceived when `spaMM` handled a single spatial random effect, and it is still convenient when applicable, but it should not be mixed with `corrPars` usage.

phi may be a single value or a vector of the same length as the response vector (the number of rows in the `data`, once non-informative rows are removed).

lambda may be a single value (if there is a single random effect, or a vector allowing to specify unambiguously variance values for some random effect(s). It can thus take the form `lambda=c(NA,1)` or `lambda=c("2"=1)` (note the name) to assign a value only to the variance of the second of two random effects.

ranCoefs is a `list` of numeric vectors, each numeric vector specifying the variance and correlation parameters for a random-coefficient term. As for `lambda`, it may be incomplete, using names to specify the random effect to which the parameters apply. For example, to assign variances values 3 and 7, and correlation value -0.05, to a second random effect, one can use `ranCoefs=list("2"=c(3,-0.05,7))` (note the name). The elements of each vector are variances and correlations, matching those of the printed summary of a fit. The order of these elements must be the order of the `lower.tri` of a covariance matrix, as shown e.g. by
`m2 <-matrix(NA,ncol=2,nrow=2); m2[lower.tri(m2,diag=TRUE)] <-seq(3); m2`.

corrPars is a list, and it may also be incomplete, using names to specify the affected random effect as shown for `lambda` and `ranCoefs`. For example, `ranFix=list(corrPars=list("1"=list(nu=0.5)))` makes explicit that `nu=0.5` applies to the first (`"1"`) random effect in the model formula. Its elements may be the correlation parameters of the given random effect. For the Matérn model, these are the correlation parameters `rho` (scale parameter(s)), `nu` (smoothness parameter), and (optionally) `Nugget` (see `Matern`). The `rho` parameter can itself be a vector with different values for different geographic coordinates. For the `adjacency` model, the only correlation parameter is a scalar `rho` (see `adjacency`). For the `AR1` model, the only correlation parameter is a scalar `ARphi` (see `AR1`). Consult the documentation for other types of random effects, such as `Cauchy` or `IMRF`, for any information missing here.

Examples

```
## Not run:
data("wafers")
# Fixing random-coefficient parameters:
HLfit(y~X1+(X2|batch), data=wafers, ranFix=list(ranCoefs=list("1"=c(2760, -0.1, 1844))))
# fixing coefficients of the linear predictor, but with REML as if they were not fixed:
HLfit(y ~X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch), data=wafers, family=Gamma(log),
      etaFix=list(beta=c("(Intercept)"=5.61208, X1=0.08818, X2=-0.21163, X3=-0.13948,
                         "I(X2^2)"=-0.10378, "X1:X3"=-0.08987, "X2:X3"=-0.08779)),
      REMLformula=y ~X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch))

data("Loaloa")
# Fixing some Matern correlation parameters, in corrHLfit:
corrHLfit(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),
             data=Loaloa,family=binomial(),ranFix=list(nu=0.5,Nugget=2/7))
# Fixing all mandatory Matern correlation parameters, in HLCor:
HLCor(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),
             data=Loaloa,family=binomial(),ranPars=list(nu=0.5,rho=0.7))

## End(Not run)
```

`fixedLRT` *Likelihood ratio test of fixed effects.*

Description

`fixedLRT` performs a likelihood ratio (LR) test between two models, the "full" and the "null" models, currently differing only in their fixed effects. Parametric bootstrap p-values can be computed, either using the raw bootstrap distribution of the likelihood ratio, or a bootstrap estimate of the Bartlett correction of the LR statistic. This function differs from `LRT` in its arguments (model fits for `LRT`, versus all arguments required to fit the models for `fixedLRT`), and in the format of its return value.

Usage

```
fixedLRT(null.formula, formula, data, method, HLmethod = method,
         REMLformula = NULL, boot.repl=0, control="DEPRECATED",
         control.boot="DEPRECATED", fittingFunction, seed=NULL,
         resp_testfn = NULL, ...)
```

Arguments

`null.formula` Either a `formula` (as in `glm`) or a `predictor` (see `Predictor`) for the null model.

`formula` Either a `formula` or a `predictor` for the full model.

`data` A data frame containing the variables in the model.

`method` A method to fit the full and null models. See `method` information about such methods. The two most meaningful values of `method` in `fixedLRT` calls are: `'ML'` for an LRT based on ML fits (generally recommended); and `'PQL/L'` for an LRT based on PQL/L fits (recommended for spatial binary data).

Also feasible, but more tricky, and not really recommended (see Rousset and Ferdy, 2014), is `'REML'`. This will perform an LRT based on two REML fits of the data, *both* of which use the same conditional (or "restricted") likelihood of residuals for estimating dispersion parameters λ and ϕ (see `REMLformula` argument). Further, REML will not be effective on a given dispersion parameter if a non-trivial init.corrHLfit value is provided for this parameter.

`HLmethod` Kept for back-compatibility. Same as `method`, but may work only for `fittingFunction=corrHLfit`.

`REMLformula` a formula specifying the fixed effects which design matrix is used in the REML correction for the estimation of dispersion parameters, if these are estimated by REML. This formula is by default that for the *full* model.

`boot.repl` the number of bootstrap replicates.

`control` Deprecated.

`control.boot` Deprecated.

`fittingFunction` Character string giving the function used to fit each model: either `"corrHLfit"` or `"fitme"`. Default is `"corrHLfit"` for small data sets (fewer than 300 observations), and `"fitme"` otherwise, but this may change in future versions.

`seed` Passed to `simulate.HLfit`

`resp_testfn` See argument `resp_testfn` of `spaMM_boot`

`...` Further arguments passed to or from other methods; presently, additional arguments passed to fitting functions.

Details

Comparison of REML fits is a priori not suitable for performing likelihood ratio tests. Nevertheless, it is possible to contrive them for testing purposes (Wehlam & Thompson 1997). This function generalizes some of Wehlam & Thompson's methods to GLMMs.

See Details in LRT for details of the bootstrap procedures.

Value

An object of class `fixedLRT`, actually a list with as-yet unstable format, but here with typical elements (depending on the options)

`fullfit`	the HLfit object for the full model;
`nullfit`	the HLfit object for the null model;
`LRTori`	A likelihood ratio chi-square statistic
`LRTprof`	Another likelihood ratio chi-square statistic, after a profiling step, if any.
`df`	the number of degrees of freedom of the test.
`trace.info`	Information on various steps of the computation.

and, if a bootstrap was performed, the additional elements described in LRT.

References

Rousset F., Ferdy, J.-B. (2014) Testing environmental and genetic effects in the presence of spatial autocorrelation. Ecography, 37: 781-790. http://dx.doi.org/10.1111/ecog.00566

Welham, S. J., and Thompson, R. (1997) Likelihood ratio tests for fixed model terms using residual maximum likelihood, J. R. Stat. Soc. B 59, 701-714.

See Also

See also corrHLfit and LRT.

Examples

```
if (spaMM.getOption("example_maxtime")>1.9) {
 data("blackcap")
 ## result comparable to the corrHLfit examples based on blackcap
 fixedLRT(null.formula=migStatus ~ 1 + Matern(1|latitude+longitude),
       formula=migStatus ~ means + Matern(1|latitude+longitude),
       method='ML',data=blackcap)
}
if (spaMM.getOption("example_maxtime")>156) {
 ## longer version with bootstrap
 fixedLRT(null.formula=migStatus ~ 1 + Matern(1|latitude+longitude),
       formula=migStatus ~ means + Matern(1|latitude+longitude),
       method='ML',data=blackcap, boot.repl=100, seed=123)
 }
```

fix_predVar *Prediction from models with nearly-singular covariance matrices*

Description

This explains how to handle a warning occur in computation of prediction variance, where the user is directed here.

For **Matern or Cauchy** correlation models with vanishing scale factor for distances, a warning may be produced when `predict.HLfit` (or `get_predVar`, etc.) is called with non-NULL `newdata`, because a nearly-singular correlation matrix of the random effect is met. **To decide what to do** in that case, users should compare the values of `get_predVar(.)` and `get_predVar(.,newdata=myfit$data)`(see Example below). In the absence of numerical inacuracies, The two values should be identical, and in the presence of such inaccuracies, the more reliable value is the first one. In really poor cases, the second syntax may yield negative prediction variances. If users deem the inaccuracies too large, they should use `control=list(fix_predVar=TRUE)` in the next call to `predict.HLfit` (or `get_predVar`, etc.) as shown in the Example. The drawback of this control is that the computation may be slower, and might even exceed memory capacity for large problems (some matrix operations being performed with exact rational arithmetic, which is memory-consuming for large matrices). it is also still experimental, in the sense that I fear that bugs (`stop`) may occur. If the user instead chooses `control=list(fix_predVar=FALSE)`, the default standard floating-point arithmetic is used, but no warning is issued.

For `fix_predVar` left NULL (the default), standard floating-point arithmetic is also used. But in addition (with exceptions: see Details), the warning keeps being issued, and the (possibly costly) computation of the inverse of the correlation matrix is not stored in the fitted model object, hence is repeated for each new prediction variance computation. This is useful to remind users that something needs to be done, but for programming purposes where repeated warnings may be a nuisance, one can use `control=list(fix_predVar=NA)` which will issue a warning then perform as `control=list(fix_predVar=FALSE)`, i.e. store an approximate inverse so the warning is not issued again. Finally, `control=list(fix_predVar=NaN)` will remove the inverse of the correlation matrix from the fitted model object, and start afresh as if the control was NULL.

Details

Nearly-singular correlation matrices of random effects occur in several contexts. For random-slope models, it commonly occurs that the fitted correlation between the random effects for Intercept and slope is 1 or -1, in which case the correlation matrix between these random effects is singular. This led to quite inaccurate computations of prediction variances in spaMM prior to version 3.1.0, but this problem has been fixed.

`control=list(fix_predVar=NaN)` may be more appropriate than `control=list(fix_predVar=NULL)` when `predict.HLfit` is called through code that one cannot control. For this reason, spaMM provides another mode of control of the default. It will convert `control=list(fix_predVar=NULL)` to other values when the call stack has call names matching the patterns given by `spaMM.getOption("fix_predVar")` (as understood by `grep`). Thus if `spaMM.getOption("fix_predVar")$"NA"=="MSL|bboptim"`, the default behaviour is that defined by `control=list(fix_predVar=NA)` when `predict.HLfit` is called through `Infusion::MSL` or `blackbox::bboptim`. FALSE or TRUE are handled in a similar way.

Examples

```
data("blackcap")
fitobject <- corrHLfit(migStatus ~ 1 + Matern(1|latitude+longitude),data=blackcap,
                       ranFix=list(nu=10,rho=0.001)) ## numerically singular C
get_predVar(fitobject,newdata=blackcap[6,])
## => warning => let us apply the recommended procedure:
get_predVar(fitobject)
get_predVar(fitobject,newdata=fitobject$data)
# Negative values again in the second case => easy decision:
get_predVar(fitobject,newdata=blackcap[1:6,],
            control=list(fix_predVar=TRUE)) # now it's accurate
            # and the accuracy control is stored in the object:
get_predVar(fitobject,newdata=blackcap[1:6,])
# Clean and start afresh:
get_predVar(fitobject,newdata=blackcap[1:6,],
            control=list(fix_predVar=NaN))
```

`freight` *Freight dataset*

Description

A set of data on airfreight breakage. Data are given on 10 air shipments, each carrying 1000 ampules of some substance. For each shipment, the number of ampules found broken upon arrival, and the number of times the shipments were transferred from one aircraft to another, are recorded.

Usage

```
data("freight")
```

Format

The data frame includes 10 observations on the following variables:

broken number of ampules found broken upon arrival.

transfers number of times the shipments were transferred from one aircraft to another.

id Shipment identifier.

Source

The data set is reported by Kutner et al. (2003) and used by Sellers and Shmueli (2010) to illustrate COMPoisson analyses.

References

Kutner MH, Nachtsheim CJ, Neter J, Li W (2005, p. 35). Applied Linear Regression Models, Fourth Edition. McGraw-Hill.

Sellers KF, Shmueli G (2010) A Flexible Regression Model for Count Data. Ann. Appl. Stat. 4: 943–961

Examples

```
## see ?COMPoisson for examples
```

---

get_cPredVar *Estimation of prediction variance with bootstrap correction*

Description

This function is similar to get_predVar except that is uses a bootstrap procedure to correct for bias in the evaluation of the prediction variance.

Usage

```
get_cPredVar(pred_object, newdata, nsim, seed, type = "residual",
             variances=NULL, nb_cores = NULL, fit_env = NULL,
             sim_object=pred_object)
```

Arguments

pred_object	an object of class HLfit, as returned by the fitting functions in spaMM.
newdata	passed to predict.HLfit (it thus represents a prediction design, not to be confused with the bootstrap samples)
nsim	passed to simulate.HLfit
seed	passed to simulate.HLfit
type	passed to simulate.HLfit
variances	NULL or list; variances["cov"] will be passed to predict.HLfit to control whether a covariance matrix is computed or not. Other elements are currently ignored.
nb_cores	integer: number of cores to use for parallel computation of bootstrap. The default is spaMM.getOption("nb_cores"), and 1 if the latter is NULL. nb_cores=1 prevents the use of parallelisation procedures.
fit_env	For parallel computations: an environment containing objects to be passed to the cores. They should have the same name in fit_env as in the environment they are passed from.
sim_object	an object of class HLfit, passed to simulate.HLfit as its object argument. Simulating from this object must produce response values that can be used as replacement to those of the original fitted pred_object. In standard usage, sim_object=pred_object (the default).

Details

The result provided by `get_cPredVar` is similar to the CMSEP (Conditional Mean Standard Error of Prediction) introduced by Booth and Hobert (1998; “B&H”). This paper is known for pointing the importance of using conditional variances when they differ from unconditional ones. This is hard to miss in spatial models, where the relevant prediction variance typically depends on the variance of random effects conditional on the data. Thus, the alternative function `get_predVar` already accounts for this and returns a prediction variance that depends on a joint covariance of fixed-effect estimates and of random effects given the data.

B&H also used a conditional bootstrap procedure to correct for some bias. `get_cPredVar` implements a similar procedure, in contrast to `get_predVar`. Their conditional bootstrap procedure is not applicable for autocorrelated random effects, and parametric bootstrapping of the residuals of the fitted model (as implied by the default value of argument `type`) is used instead here. Apart from this difference, the returned value includes exactly the same terms as those discussed by B&H: their “naive estimate” ν_i and its bootstrap correction b_i, their correction β for uncertainty in fixed-effect coefficients, and their correction σ^2 for uncertainty in dispersion parameters.

The “naive estimate” ν_i is not generally an estimate of anything uniquely defined by the model parameters: for correlated random effects, it depends on the “root” of the correlation matrix of the random effects, which is not unique. Thus ν_i is not unique, and may differ for example for equivalent fits by sparse-precision methods vs. other methods. Nevertheless, `attr(cpredvar,"info")$naive` does recover published values in the Examples below, as they involve no correlation matrix.

Value

A vector of prediction variances, with an attribute `info` which is an **environment** containing variables:

`SEs`	the standard errors of the estimates (which are those of the bootstrap replicates)
`bias`	the bias term
`maive`	B&H’s “naive” ν_i

References

Booth, J.G., Hobert, J.P. (1998) Standard errors of prediction in generalized linear mixed models. J. Am. Stat. Assoc. 93: 262-272.

Examples

```
## Not run:
if(requireNamespace("rsae", quietly = TRUE)) {
  # LMM example from Booth & Hobert 1998 JASA
  data("landsat", package = "rsae")
 fitCorn <- fitme(HACorn ~ PixelsCorn + PixelsSoybeans + (1|CountyName),data=landsat[-33,])
  newXandZ <- unique(data.frame(PixelsCorn=landsat$MeanPixelsCorn,
                                PixelsSoybeans=landsat$MeanPixelsSoybeans,
                                CountyName=landsat$CountyName))
 (cpredvar <- get_cPredVar(fitCorn, newdata=newXandZ, nsim=200L, seed=123)) # serial computation
  (cpredvar <- get_cPredVar(fitCorn, newdata=newXandZ, nsim=200L, seed=123,
        nb_cores=parallel::detectCores()-1L, fit_env=list2env(list(newXandZ=newXandZ))))
}
```

```
# GLMM example from Booth & Hobert 1998 JASA
npos <- c(11,16,14,2,6,1,1,4,10,22,7,1,0,0,1,6)
ntot <- c(36,20,19,16,17,11,5,6,37,32,19,17,12,10,9,7)
treatment <- c(rep(1,8),rep(0,8))
clinic <-c(seq(8),seq(8))
clinics <- data.frame(npos=npos,nneg=ntot-npos,treatment=treatment,clinic=clinic)
#
fitClinics <- HLfit(cbind(npos,nneg)~treatment+(1|clinic),family=binomial(),data=clinics)
#
(get_cPredVar(fitClinics, newdata=clinics[1:8,], nsim=200L, seed=123)) # serial computation
(get_cPredVar(fitClinics, newdata=clinics[1:8,], nsim=200L, seed=123,
      nb_cores=parallel::detectCores()-1, fit_env=list2env(list(clinics=clinics))))

## End(Not run)
```

get_inits_from_fit *Initiate a fit from another fit*

Description

get_inits_from_fit is an extractor of some fitted values from a fit in a convenient format to initiate a next fit. This function is work in progress, and may not extract all values that can be used to initiate a fit (see Value).

Usage

```
get_inits_from_fit(from, template = NULL, to_fn = NULL)
```

Arguments

from	Fit object (inheriting from class "HLfit") from which fitted values are taken.
template	Another fit object. Usage with a template fit object is suitable for refitting this object using fitted values from the from object as starting values.
to_fn	NULL or character: the name of the function to be used the next fit. If NULL, taken from template (if available), else from from. It is meaningful to provide a to_fn distinct from the function used to fit a template.

Value

A list with elements

init, init.corrHLfit
(depending on the fitting function) giving initial values for outer-optimization;

init.HLfit giving initial values for the iterative algorithms in HLfit. It is itself a list with possible elements:

- fixef for the coefficients of the linear predictor, adjusted to the format of the coefficients of the linear predictor of the template object, if available;
- ranCoefs random-coefficients parameters (if **not** outer-optimized).

See Also

get_ranPars and VarCorr.

Examples

```
## Not run:
data("blackcap")
(corrhlfit <- corrHLfit(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
                        method="ML"))
inits <- get_inits_from_fit(corrhlfit, to_fn = "fitme")
(fitfit <- fitme(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
                 init=inits$init))
inits <- get_inits_from_fit(corrhlfit, template = fitfit)
fitme(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
      init=inits$init)
# In these examples, inits$init.HLfit is useless
# as it is ignored when LMMs are fitted by fitme().

## End(Not run)
```

---

get_matrix *Extract matrices from a fit*

Description

get_matrix is a first attempt at a unified extractor of various matrices from a fit. All augmented matrices follow (Henderson's) block order (upper blocks: X,Z; lower blocks: 0,I). get_ZALMatrix returns the design matrix for the random effects v.

Usage

```
get_matrix(object, which="model.matrix", augmented=TRUE, ...)
get_ZALMatrix(object, as_matrix, force_bind=FALSE)
```

Arguments

object	An object of class HLfit, as returned by the fitting functions in spaMM.
augmented	Boolean; whether to return an augmented matrix for all model coefficients (fixed-effects coefficients and random-effect predictions) or only for fixed effects. Not operative for all which values (currently only for which="left_ginv").
which	Which element to extract. For "model.matrix", the design matrix for fixed effects (similarly to stats::model.matrix); for "ZAL", the design matrix for random effects (same as get_ZALMatrix()); for "AugX", the (unweighted) augmented design matrix of the least-square problem; for "hat_matrix", the projection matrix that gives model predictions from the (augmented) response vector; for "left_ginv", the pseudo-inverse that gives the model coefficients from the (augmented) response vector. See Details for definitions and further options.

as_matrix	Deprecated.
force_bind	Boolean; with the default value FALSE, the function may return an object of class ZAXlist, which is poorly documented and for development purposes only.
...	Other arguments that may be needed in some future versions of spaMM.

Details

(Given the pain that is to to write maths in R documentation files, readers are gently asked to be tolerant about any imperfections of the following).

Model coefficents estimates of a (weighted) linear model can be written as $(\mathbf{X'WX})^{-1}\mathbf{X'Wy}$ where $\mathbf{X}$ is the design matrix for fixed effects, $\mathbf{W}$ a diagonal weight matrix, and $\mathbf{y}$ the response vector. In a linear mixed model, the same expression holds in terms of Henderson's augmented design matrix, of an augmented (still diagonal) weight matrix, and of an augmented response vector. For GLMMs and hierarchical GLMs generally, the solution of each step of the iteratively reweighted least squares algorithm again has the same expression in terms of appropriately defined augmented matrices and vectors.

get_matrix returns, for given values of the which argument, the following matrices from the model fit: "AugX": $\mathbf{X}$; "wei_AugX": $\mathbf{WX}$; "wAugX": $\sqrt{(\mathbf{W})}\mathbf{X}$; "left_ginv": $\mathbf{X}^{-}$=$(\mathbf{X'WX})^{-1}\mathbf{X'W}$ (viewed as a pseudo-inverse since $\mathbf{X}^{-}\mathbf{X}$ is an identity matrix); "hat_matrix": $\mathbf{XX}^{-}$=$\mathbf{X}\,(\mathbf{X'WX})^{-1}\mathbf{X'W}$.

Value

A matrix, possibly in sparse-matrix format.

---

get_ranPars   *Operations on lists of parameters. A distinct documentation exists for get_inits_from_fit which is most useful to extract parameters from a fit in a form suitable to initialize another fit. VarCorr is yet another extractor for dispersion parameters.*

Description

get_ranPars returns various subsets of random-effect parameters. remove_from_parlist removes elements from a list of parameters, and from its type attribute.

Usage

```
get_ranPars(object, which=NULL, ...)
remove_from_parlist(parlist, removand=NULL, rm_names=names(unlist(removand)))
```

Arguments

object	An object of class HLfit, as returned by the fitting functions in spaMM.
which	For get_ranPars, the only non-null value is "corrPars", to return correlation parameters of random effects.
...	Other arguments that may be needed by some method.

parlist	A list of parameters. see Details.
removand	Optional. A list of parameters to be removed from parlist.
rm_names	Names of parameters to be removed from parlist. Mandatory if removand is not given.

Details

remove_from_parlist is designed to manipulate structured lists of parameters, such as a list with elements phi, lambda, and corrPars, the latter being itself a list structured as the return value of get_ranPars(.,which="corrPars"). remove_from_parlist may have an attribute type, also with elements phi, lambda, and corrPars... If given, removand must have the same structure (but typically not all the elements of parlist); otherwise, rm_names must have elements which match names of unlist(names(parlist)).

The bias estimated by bootstrap is one of two terms included in the get_cPredVar computation, which together account for uncertainty in fixed effects and dispersion parameters. The computation does not account for uncertainty in correlation parameters "outer-optimized" by fitme or corrHLfit. The bias computation would not be sufficient to account for the latter uncertainty, and thus the correlation parameters are fixed when the model is refitted on the bootstrap replicates. To account for uncertainty in correlation parameters, one should rather perform a parametric boostrap of the full model (typically using spaMM_boot(.,type="residual")), which may take much more time.

Value

get_ranPars(.,which="corrPars") returns a (possibly nested) list of correlation parameters (or NULL if there is no such parameter). Top-level elements correspond to the different random effects. The list has a "type" attribute having the same nested-list structure and describing whether and how the parameters where fitted: "fix" means they where fixed, not fitted; "var" means they were fitted by HLfit's specific algorithms; "outer" means they were fitted by a generic optimization method. Note that get_ranPars does not provide random-coefficient parameters when these were fitted by HLfit's specific algorithms, but that get_inits_from_fit can do that.

remove_from_parlist returns a list of model parameters with given elements removed, and likewise for its (optional) type attribute. See Details for context of application.

See Also

get_inits_from_fit and VarCorr.

Examples

```
data("wafers")
m1 <- HLfit(y ~X1+X2+(1|batch), resid.model = ~ 1, data=wafers, method="ML")
get_ranPars(m1,which="corrPars") # NULL since no correlated random effect

parlist1 <- list(lambda=1,phi=2,corrPars=list("1"=list(rho=3,nu=4),"2"=list(rho=5)))
parlist2 <- list(lambda=NA,corrPars=list("1"=list(rho=NA))) # values of elements do not matter
remove_from_parlist(parlist1,parlist2) ## same result as:
remove_from_parlist(parlist1,rm_names = names(unlist(parlist2)))
```

good-practice *Clear and trustworthy formulas*

Description

Base fitting functions in R will seek variables in the global environment (or more generally in the environment where a call to `~` was made, defining the model formula) if they are not in the `data`. This easily leads to errors (see example in the discussion of `update.HLfit`). Indeed Chambers (2008, p.221), after describing how the environment is defined, comments that "Where clear and trustworthy software is a priority, I would personally avoid such tricks. Ideally, all the variables in the model frame should come from an explicit, verifiable data source...". Hence, the main fitting functions in spaMM depart from the sloppy practice. They strip the formula environment from any variable, and seek all variables from the `formula` in the data frame given by their `data` argument. **One never needs to specify the `data` in the `formula`.** The variables defining the `prior.weights` should also be in the `data`. Variables used in other arguments such as `ranFix` are looked up neither in the data nor in the formula environment, but in the calling environment as usual.

References

Chambers J.M. (2008) Software for data analysis: Programming with R. Springer-Verlag New York

hatvalues.HLfit *Leverage extractor for HLfit objects*

Description

This gets "leverages" or "hat values" from an object. However, ther is hidden complexity in what this may mean, so care must be used in selecting proper arguments for a given use.

Usage

```
## S3 method for class 'HLfit'
hatvalues(model, type = "projection", which = "resid", force=FALSE, ...)
```

Arguments

`model`	An object of class `HLfit`, as returned by the fitting functions in `spaMM`.
`type`	Character: `"projection"`, `"std"`, or more cryptic values not documented here.
`which`	Character: `"resid"` for the traditional leverages of the observations, `"ranef"` for random-effect leverages, or `"both"` for both.
`force`	Boolean: to force recomputation of the leverages even if they are available in the object, for checking purposes.
`...`	For consistency with the generic.

Details

Leverages may have distinct meaning depending on context. The textbook version for linear models is that leverages (q_i) are the diagonal elements of a projection matrix (“hat matrix”), and that they may be used to standardize (“studentize”) residuals as follows. If the residual variance ϕ is known, then the variance of each fitted residual $\hat{e}_i$ is $\phi(1 - q_i)$. Standardized residuals, all with variance 1, are then $\hat{e}_i/\sqrt(\phi(1 - q_i))$.

This no longer holds exactly with estimated ϕ, but if one uses here an unbiased (REML) estimator of ϕ, the studentized residuals may still practically have a unit expected variance. By comparison, one expects a distinct bias if one uses an ML estimator of ϕ: the expected variance of such standardized residuals is no longer 1. For example, when a simple linear model is fitted by ML, the variance of the fitted residuals is less than ϕ, but $\hat{\phi}$ is downward biased so that residuals standardized by $\sqrt(\phi)$, without any leverage correction, more closely have expected unit variance.

Leverages also appear in expressions for derivatives, with respect to the dispersion parameters, of the logdet(Hessian) term of Laplace approximations for marginal or restricted likelihood (Lee et al. 2006). This provides a basis to generalize the concept of standardizing leverages for ML and REML in mixed-effect models. In particular, in an ML fit, one considers leverages $(q*_i)$ that are no longer the diagonal elements of the projection matrix for the mixed model (they are zero in a simple LM). The generalized standardizing leverages may include corrections for non-Gaussian response, for non-Gaussian random effects, and for taking into account the variation of the GLM weights in the logdet(Hessian) derivatives. Leverages are also defined for the random effects. Which corrections are included depend on the precise method used to fit the model (e.g., EQL vs PQL vs REML).

These distinctions suggest breaking the usual synonymy between “leverages” or “hat values”: the term “hat values” better stands for the diagonal elements of a projection matrix, while “leverages” better stands for the standardizing values. `hatvalues(.,type="std")` returns the standardizing leverages. By contrast, `hatvalues(.,type="projection")` will always return hat values from the fitted projection matrix. Note that these values still differs between ML and REML fit because the fitted projection matrix differs between them.

Value

A list with separate components `resid` (leverages of the observations) and `ranef` if `which="both"`, and a vector otherwise.

References

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006) Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

Examples

```
if (spaMM.getOption("example_maxtime")>0.8) {
data("Orthodont",package = "nlme")
rnge <- (107:108)

# all different:
#
hatvalues(rlfit <- fitme(distance ~ age+(age|Subject),
                         data = Orthodont, method="REML"))[rnge]
```

```
hatvalues(mlfit <- fitme(distance ~ age+(age|Subject),
                        data = Orthodont))[rnge]
hatvalues(mlfit,type="std")[rnge]
}
```

HLCor *Fits a (spatially) correlated mixed model, for given correlation parameters*

Description

A convenient interface for HLfit, constructing the correlation matrix of random effects from the arguments, then estimating fixed effects and dispersion parameters using HLfit.

Usage

```
HLCor(formula, data, family = gaussian(), ranPars = NULL, distMatrix,
      uniqueGeo = NULL, adjMatrix, corrMatrix, covStruct=NULL,
      method = "REML", verbose = c(trace=FALSE),
      control.dist = list(), ...)
```

Arguments

formula	A predictor, i.e. a formula with attributes (see Predictor), or possibly simply a simple formula if an offset is not required.
ranPars	A list of values for correlation parameters (some of which are mandatory), and possibly also dispersion parameters (optional, but passed to HLfit if present). See ranPars for further information.
data	The data frame to be analyzed.
family	A family object describing the distribution of the response variable. See HLfit for further information.
distMatrix	A distance matrix between geographic locations, forwarded to MaternCorr
uniqueGeo	A matrix of non-redundant geographic locations. Such a matrix is typically constructed automatically from the data and the model formula, but otherwise could be useful if further the rho parameter is a vector with different values for different coordinates, in which case a scaled distance matrix has to be reconstructed from uniqueGeo and rho.
adjMatrix	An adjacency matrix, used if a random effect of the form y ~ adjacency(1|<location index>) is present. See adjacency for further details.
corrMatrix	A matrix **C** used if a random effect term of the form corrMatrix(1|<stuff>) is present. This allows to analyze non-spatial model by giving for example a matrix of genetic correlations. Each row corresponds to levels of a variable <stuff>. The covariance matrix of the random effects for each level is then $\lambda\mathbf{C}$, where as usual λ denotes a variance factor for the random effects (if **C** is a correlation matrix, then λ is the variance, but other cases are possible). See corrMatrix for further details.

`covStruct` An interface for specifying correlation structures for different types of random effect (`corrMatrix` or `adjacency`). See `covStruct` for details.

`method` Character: the fitting method to be used, such as `"ML"`, `"REML"` or `"PQL/L"`. `"REML"` is the default. Other possible values of `HLfit`'s `method` argument are handled.

`verbose` A vector of booleans. `trace` controls various diagnostic (possibly messy) messages about the iterations.

`control.dist` A list of arguments that control the computation of the distance argument of the correlation functions. Possible elements are

- **rho.mapping** a set of indices controlling which elements of the `rho` scale vector scales which dimension(s) of the space in which (spatial) correlation matrices of random effects are computed. See same argument in `make_scaled_dist` for details and examples.
- **dist.method** `method` argument of `proxy::dist` function (by default, `"Euclidean"`, but see `make_scaled_dist` for other distances such as spherical ones.)

`...` Further parameters passed to `HLfit` or to `mat_sqrt`.

Details

For approximations of likelihood, see `method`. For the possible structures of random effects, see `random-effects`, but note that `HLCor` cannot adjust parameters of correlation models (with the exception of conditional autoregressive ones). Any such parameter must be specified by the `ranPars` argument. More generally, the correlation matrix for random effects can be specified by various combinations of formula terms and other arguments (see Examples):

Basic Matérn model `Matern(1|<...>)`, using the spatial coordinates in `<...>`. This will construct a correlation matrix according to the Matérn correlation function (see `MaternCorr`);

Matérn model with given distance matrix `Matern(1|<...>)` with `distMatrix`;

Given correlation matrix `corrMatrix(1|<...>)` with `corrMatrix` argument. See `corrMatrix` for further details.

CAR model with given adjacency matrix `adjacency(1|<...>)` with `adjMatrix`. See `adjacency` for further details;

AR1 model `AR1(1|<...>)` See `AR1` for further details.

Value

The return value of an `HLfit` call, with the following additional attributes:

`HLCorcall` the HLCor call

`info.uniqueGeo` Unique geographic locations.

See Also

`autoregressive` for additional examples, `MaternCorr`, `HLfit`, and `corrHLfit`

Examples

```
# Example with an adjacency matrix (autoregressive model):
# see 'adjacency' documentation page

#### Matern correlation using only the Matern() syntax
if (spaMM.getOption("example_maxtime")>0.8) {
 data("Loaloa")
 HLCor(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
              +Matern(1|longitude+latitude), data=Loaloa,
       family=binomial(), ranPars=list(nu=0.5,rho=1/0.7))
}


#### Matern correlation using a distMatrix
data("blackcap")
MLdistMat <- as.matrix(proxy::dist(blackcap[,c("latitude","longitude")]))
HLCor(migStatus ~ means+ Matern(1|latitude+longitude),data=blackcap,
      distMatrix=MLdistMat, method="ML", ranPars=list(nu=0.6285603,rho=0.0544659))
```

`HLfit` *Fit mixed models with given correlation matrix*

Description

This function fits GLMMs as well as some hierarchical generalized linear models (HGLM; Lee and Nelder 2001). `HLfit` fits both fixed effects parameters, and dispersion parameters i.e. the variance of the random effects (full covariance for random-coefficient models), and the variance of the residual error. The linear predictor is of the standard form `offset+ X beta + Z b`, where X is the design matrix of fixed effects and Z is a design matrix of random effects (typically an incidence matrix with 0s and 1s, but not necessarily). Models are fitted by an iterative algorithm alternating estimation of fixed effects and of dispersion parameters. The residual dispersion may follow a "structured-dispersion model" modeling heteroscedasticity. Estimation of the latter parameters is performed by a form of fit of debiased residuals, which allows fitting a structured-dispersion model (Smyth et al. 2001). However, evaluation of the debiased residuals can be slow in particular for large datasets. For models without structured dispersion, it is then worth using the `fitme` function (or the `corrHLfit` function with non-default arguments). These functions can optimize the likelihood of `HLfit` fits for different given values of the dispersion parameters ("outer optimization"), thereby avoiding the need to estimate debiased residuals.

Usage

```
HLfit(formula, data, family = gaussian(), rand.family = gaussian(),
      resid.model = ~1, REMLformula = NULL, verbose = c(trace = FALSE),
      HLmethod = "HL(1,1)", method="REML", control.HLfit = list(),
      control.glm = list(), init.HLfit = list(), ranFix = list(),
      etaFix = list(), prior.weights = NULL, processed = NULL)
## see 'rand.family' argument for inverse.Gamma
```

Arguments

formula
: A formula; or a predictor, i.e. a formula with attributes created by Predictor, if design matrices for random effects have to be provided. See Details in spaMM for allowed terms in the formula (except spatial ones).

data
: A data frame containing the variables named in the model formula.

family
: A family object describing the distribution of the response variable. See Details in spaMM for handled families.

rand.family
: A family object describing the distribution of the random effect, or a list of family objects for different random effects (see Examples). Possible options are gaussian(), Gamma(log), Gamma(identity) (see Details), Beta(logit), inverse.Gamma(-1/mu), and inverse.Gamma(log). For discussion of these alternatives see Lee and Nelder 2001 or Lee et al. 2006, p. 178-. Here the family gives the distribution of a random effect u and the link gives v as function of u (see Details). If there are several random effects and only one family is given, this family holds for all random effects.

resid.model
: **Either** a formula (without left-hand side) for the dispersion parameter phi of the residual error. A log link is assumed by default;
or a list, with at most three possible elements if its formula involves only fixed effects:

  formula model formula as in formula-only case, without left-hand side

  family Always Gamma, with by default a log link. Gamma(identity) can be tried but may fail because only the log link ensures that the fitted ϕ is positive.

  fixed can be used to specify the residual dispersion parameter of the residual dispersion model itself. The default value is 1; this argument can be used to set another value, and fixed=list(phi=NA) will force estimation of this parameter.

  and additional possible elements (all named as fitme arguments) if its formula involves random effects: see phiHGLM.

REMLformula
: A model formula that allows the estimation of dispersion parameters, and computation of restricted likelihood (p_bv) under a model different from the predictor formula.

  For example, if only random effects are included in REMLformula, an ML fit is performed and p_bv equals the marginal likelihood (or its approximation), p_v. This ML fit can be performed more simply by setting method="ML" and leaving REMLformula at its default NULL value.

verbose
: A vector of booleans. trace controls various diagnostic messages (possibly messy) about the iterations. TRACE=TRUE is most useful to follow the progress of a long computation, particularly in fitme or corrHLfit calls, for which it displays some mysterious output for each set of correlation and dispersion parameter values considered by the optimiser. Non-boolean values of TRACE are meaningful, but the source code of spaMM:::.doTRACE should be consulted for their meaning. phifit (which defaults to TRUE) controls messages about the progress of residual dispersion fits in DHGLMs.

method
: Character: the fitting method. allowed values are "REML", "ML", "EQL-" and "EQL+" for all models; "PQL" (="REPQL") and "PQL/L" for GLMMs only; and further values for those curious to experiment (see method). **The default is REML** (standard REML for LMMs, an extended definition for other models). REML can be viewed as a form of conditional inference, and non-standard conditionings can be called by using a non-standard REMLformula.

HLmethod
: Same as method. It is useless to specify HLmethod when method is specified. The default value "HL(1,1)" means the same as method="REML", but more accurately relates to definitions of approximations of likelihoo in the h-likelihood literature.

control.HLfit
: A list of parameters controlling the fitting algorithms, which should be ignored in routine use. In addition, a resid.family parameter was previously documented here (before version 2.6.40), and will still operate as previously documented, but should not be used in new code.

  Possible parameters are:

  conv.threshold and spaMM_tol: spaMM_tol is a list of tolerance values, with elements Xtol_rel and Xtol_abs that define thresholds for relative and absolute changes in parameter values in iterative algorithms (used in tests of the form "d(param)< Xtol_rel * param + Xtol_abs", so that Xtol_abs is operative only for small parameter values). conv.threshold is the older way to control Xtol_rel. Default values are given by spaMM.getOption("spaMM_tol");

  break_conv_logL, a boolean specifying whether the iterative algorithm should terminate when log-likelihood appears to have converged (roughly, when its relative variation over on iteration is lower than 1e-8). Default is FALSE (convergence is then assessed on the parameter estimates rather than on log-likelihood).

  iter.mean.dispFix, the number of iterations of the iterative algorithm for coefficients of the linear predictor, if no dispersion parameters are estimated by the iterative algorithm. Defaults to 200 except for Gamma(log)-family models;

  iter.mean.dispVar, the number of iterations of the iterative algorithm for coefficients of the linear predictor, if some dispersion parameter(s) is estimated by the iterative algorithm. Defaults to 50 except for Gamma(log)-family models;

  max.iter, the number of iterations of the iterative algorithm for joint estimation of dispersion parameters and of coefficients of the linear predictor. Defaults to 200. This is typically much more than necessary, unless there is little information to separately estimate λ and ϕ parameters.

control.glm
: List of parameters controlling GLM fits, passed to glm.control; e.g. control.glm=list(maxit=100). See glm.control for further details.

init.HLfit
: A list of initial values for the iterative algorithm, with possible elements of the list are fixef for fixed effect estimates (beta), v_h for random effects vector **v** in the linear predictor, lambda for the parameter determining the variance of random effects u as drawn from the rand.family distribution phi for the residual variance. However, this argument can be ignored in routine use.

ranFix
: A list of fixed values of random effect parameters. See ranFix for further information.

etaFix
: A list of given values of the coefficients of the linear predictor. See etaFix for further information.

prior.weights An optional vector of prior weights as in glm. This fits the data to a probability model with residual variance phi/prior.weights, and all further outputs are defined to be consistent with this (see section IV in Details).

processed A list of preprocessed arguments, for programming purposes only (as in corrHLfit).

Details

I. Approximations of likelihood: see method.

II. Possible structure of Random effects: see random-effects, but note that HLfit does not fit models with autocorrelated random effects).

III. The standard errors reported may sometimes be misleading. For each set of parameters among β, λ, and ϕ parameters these are computed assuming that the other parameters are known without error. This is why they are labelled Cond. SE (conditional standard error). This is most uninformative in the unusual case where λ and ϕ are not separately estimable parameters. Further, the SEs for λ and ϕ are rough approximations as discussed in particular by Smyth et al. (2001; V_1 method).

IV. prior weights. This controls the likelihood analysis of heteroscedastic models. In particular, changing the weights by a constant factor *f* should, and will, yield a fit with unchanged likelihood and (Intercept) estimates of phi also increased by *f* (except if a non-trivial resid.formula with log link is used). This is consistent with what glm does, but other packages may not follow this logic (whatever their documentation may say: check by yourself by changing the weights by a constant factor).

Value

An object of class HLfit, which is a list with many elements, not all of which are documented.

A few extractor functions are available (see extractors), and should be used as far as possible as they should be backward-compatible from version 1.4 onwards, while the structure of the return object may still evolve. The following information will be useful for extracting further elements of the object.

Elements include **descriptors of the fit**:

eta Fitted values on the linear scale (including the predicted random effects);

fv Fitted values (μ =<inverse-link>(η)) of the response variable (returned by the fitted function);

fixef The fixed effects coefficients, β (returned by the fixef function);

ranef The random effects u (returned by ranef(*,type="uncorrelated");

v_h The random effects on the linear scale, v;

phi The residual variance ϕ;

phi.object A possibly more complex object describing ϕ;

lambda The random-effect (u) variance(s) λ in compact form;

lambda.object A possibly more complex object describing λ;

corrPars Agglomerates information on correlation parameters, either fixed, or estimated by HLfit, corrHLfit or fitme;

APHLs A list which elements are various likelihood components, include conditional likelihood, h-likelihood, and the two adjusted profile h-likelihoods: the (approximate) marginal **likelihood** p_v and the (approximate) **restricted likelihood** p_bv (the latter two available through the logLik function). See the extractor function get_any_IC for information criteria ("AIC") and effective degrees of freedom;

The covariance matrix of β estimates is not included as such, but can be extracted by vcov;

Information about the input is contained in output elements named as HLfit or corrHLfit arguments (data,family,resid.family,ranFix,prior.weights), with the following notable exceptions or modifications:

predictor The formula, possibly reformatted;

resid.predictor
Analogous to predictor, for the residual variance;

rand.families corresponding to the rand.family input;

Further miscellaneous diagnostics and descriptors of model structure:

X.pv The design matrix for fixed effects;

ZAlist,strucList
Two lists of matrices, respectively the design matrices "**Z**", and the "**L**" matrices, for the different random-effect terms. The extractor get_ZALMatrix can be used to reconstruct a single "**ZL**" matrix for all terms.

BinomialDen (binomial data only) the binomial denominators;

y the response vector; for binomial data, the frequency response.

models Additional information on model structure for η, λ and ϕ;

HL A set of indices that characterize the approximations used for likelihood;

leve_phi,lev_lambda
Leverages;

dfs degrees of freedom for different components of the model;

warnings A list of warnings for events that may have occurred during the fit.

Finally, the object includes programming tools: call,spaMM.version,fit_time and envir.

References

Lee, Y., Nelder, J. A. (2001) Hierarchical generalised linear models: A synthesis of generalised linear models, random-effect models and structured dispersions. Biometrika 88, 987-1006.

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006). Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

Smyth GK, Huele AF, Verbyla AP (2001). Exact and approximate REML for heteroscedastic regression. Statistical Modelling 1, 161-175.

See Also

HLCor for estimation with given spatial correlation parameters; corrHLfit for joint estimation with spatial correlation parameters; fitme as an alternative to all these functions.

Examples

```
data("wafers")
## Gamma GLMM with log link

HLfit(y ~ X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch), family=Gamma(log),
          resid.model = ~ X3+I(X3^2) ,data=wafers)

## Gamma - inverseGamma HGLM with log link
HLfit(y ~ X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch), family=Gamma(log),
          rand.family=inverse.Gamma(log),
          resid.model = ~ X3+I(X3^2) , data=wafers)
```

---

how — *Extract information about how an object was obtained*

Description

how is defined as a generic with currently only one non-default method, for objects of class HLfit. This method provide information about how such a fit was obtained.

Usage

```
how(object, ...)
## S3 method for class 'HLfit'
how(object, devel=FALSE, verbose=TRUE, format=print, ...)
```

Arguments

object	Any R object.
devel	Boolean; Whether to provide additional cryptic information. For development purposes, not further documented.
verbose	Boolean; Whether to print information about the input object.
format	wrapper for printing format. E.g., cat(crayon::yellow(s));cat("\n") could be used instead of the default.
...	Other arguments that may be needed by some method.

Value

A list, returned invisibly, whose elements are not further described here. If verbose is TRUE, the function prints a message presenting these elements, some of which may be slightly cryptic (it is a clean way of getting the fit time, though). This function is work in progress.

Examples

```
foo <- HLfit(y~x, data=data.frame(x=runif(3), y=runif(3)), method="ML", ranFix=list(phi=1))
how(foo)
```

---

inverse.Gamma *Distribution families for Gamma and inverse Gamma-distributed random effects*

Description

For dispersion parameter λ, `Gamma` means that random effects are distributed as u Gamma(`shape`=1/λ,**scale=**λ), so u has mean 1 and variance λ. Both the log ($v = log(u)$) and identity ($v = u$) links are possible, though in the latter case the variance of u is constrained below 1 (otherwise Laplace approximations fail).

The two-parameter inverse Gamma distribution is the distribution of the reciprocal of a variable distributed according to the Gamma distribution Gamma with the same shape and scale parameters. `inverse.Gamma` implements the one-parameter inverse Gamma family with shape=1+1/λ and **rate**=1/λ) (rate=1/scale). It is used to model the distribution of random effects. Its mean=1; and its variance =$\lambda/(1-\lambda))$ if $\lambda < 1$, otherwise infinite. The default link is `"-1/mu"`, in which case `v=-1/u` is "-Gamma"-distributed with the same shape and rate, hence with mean $-(\lambda+1)$ and variance $\lambda(\lambda+1)$, which is a different one-parameter Gamma family than the above-described `Gamma`. The other possible link is `v=log(u)` in which case
$v \; - log(X$ Gamma$(1+1/\lambda, 1/\lambda))$, with mean $-(log(1/\lambda)+$digamma$(1+1/\lambda))$ and variance trigamma$(1+1/\lambda)$.

Usage

```
inverse.Gamma(link = "-1/mu")
# Gamma(link = "inverse") using stats::Gamma
```

Arguments

`link`	For `Gamma`, allowed links are `log` and `identity` (the default link from `Gamma`, `"inverse"`, cannot be used for the random effect specification). For `inverse.Gamma`, allowed links are `"-1/mu"` (default) and `log`.

Examples

```
# see help("HLfit") for fits using the inverse.Gamma distribution.
```

is_separated *Checking for (quasi-)separation in binomial-response model.*

Description

Separation occurs in binomial response models when a combination of the predictor variables perfectly predict a level of the response. In such a case the estimates of the coefficients for these variables diverge to (+/-)infinity, and the numerical algorithms typically fail. To anticipate such a problem, the fitting functions in spaMM try to check for separation by default. The check may take much time, and is skipped if the "problem size" exceeds a threshold defined by spaMM.options(separation_max=<.>), in which case a message will tell users by how much they should increase separation_max to force the check (its exact meaning and default value are subject to changes without notice but the default value aims to correspond to a separation check time of the order of 1s on the author's computer).

is_separated is a convenient interface to procedures from the ROI package, which can be called explicitly by the user to check bootstrap samples (see Example in anova). is_separated.formula is a variant (not yet a formal S3 method) that performs the same check, but using arguments similar to those of fitme(.,family=binomial()).

Usage

```
is_separated(x, y, verbose = TRUE, solver=spaMM.getOption("sep_solver"))
is_separated.formula(formula, ..., separation_max=spaMM.getOption("separation_max"),
                     solver=spaMM.getOption("sep_solver"))
```

Arguments

x	Design matrix for fixed effects.
y	Numeric response vector
formula	A model formula
...	data and possibly other arguments of a fitme call. family is ignored if present.
separation_max	numeric: non-default value allow for easier local control of this spaMM option.
solver	character: name of linear programming solver used to assess separation; passed to ROI_solve's solver argument. One can select other solvers if the corresponding ROI plugin is installed.
verbose	Whether to print some messages or not.

Value

Returns a boolean; TRUE means there is (quasi-)separation.

References

The method accessible by solver="glpk" implements algorithms described by

Konis, K. 2007. Linear Programming Algorithms for Detecting Separated Data in Binary Logistic Regression Models. DPhil Thesis, Univ. Oxford. https://ora.ox.ac.uk/objects/uuid:8f9ee0d0-d78e-4101-9ab4-f9cbceed2a2a.

See Also

See also the 'safeBinaryRegression' and 'detectseparation' package.

Examples

```
set.seed(123)
d <- data.frame(success = rbinom(10, size = 1, prob = 0.9), x = 1:10)
is_separated.formula(formula= success~x, data=d) # FALSE
is_separated.formula(formula= success~I(success^2), data=d) # TRUE
```

`Loaloa` *Loa loa prevalence in North Cameroon, 1991-2001*

Description

This data set describes prevalence of infection by the nematode *Loa loa* in North Cameroon, 1991-2001. This is a superset of the data discussed by Diggle and Ribeiro (2007) and Diggle et al. (2007). The study investigated the relationship between altitude, vegetation indices, and prevalence of the parasite.

Usage

```
data("Loaloa")
```

Format

The data frame includes 197 observations on the following variables:

latitude latitude, in degrees.

longitude longitude, in degrees.

ntot sample size per location

npos number of infected individuals per location

maxNDVI maximum normalised-difference vegetation index (NDVI) from repeated satellite scans

seNDVI standard error of NDVI

elev1 altitude, in m.

elev2,elev3,elev4 Additional altitude variables derived from the previous one, provided for convenience: respectively, positive values of altitude-650, positive values of altitude-1000, and positive values of altitude-1300

maxNDVI1 a copy of maxNDVI modified as `maxNDVI1[maxNDVI1>0.8] <-0.8`

Source

The data were last retrieved on March 1, 2013 from P.J. Ribeiro's web resources at `www.leg.ufpr.br/doku.php/pessoais:` The current (2019-06-10) link seems to be `http://www.leg.ufpr.br/lib/exe/fetch.php/pessoais:paulojus:mbgbook:datasets:loaloa.txt`).

References

Diggle, P., and Ribeiro, P. 2007. Model-based geostatistics, Springer series in statistics, Springer, New York.

Diggle, P. J., Thomson, M. C., Christensen, O. F., Rowlingson, B., Obsomer, V., Gardon, J., Wanji, S., Takougang, I., Enyong, P., Kamgno, J., Remme, J. H., Boussinesq, M., and Molyneux, D. H. 2007. Spatial modelling and the prediction of Loa loa risk: decision making under uncertainty, Ann. Trop. Med. Parasitol. 101, 499-509.

Examples

```
data("Loaloa")
if (spaMM.getOption("example_maxtime")>5) {
  fitme(cbind(npos,ntot-npos)~1 +Matern(1|longitude+latitude),
        data=Loaloa, family=binomial())
}

### Variations on the model fit by Diggle et al.
###    on a subset of the Loaloa data
### In each case this shows the slight differences in syntax,
###    and the difference in 'typical' computation times,
###    when fit using corrHLfit() or fitme().

if (spaMM.getOption("example_maxtime")>4) {
  corrHLfit(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
                  +Matern(1|longitude+latitude),method="HL(0,1)",
                 data=Loaloa,family=binomial(),ranFix=list(nu=0.5))
}
if (spaMM.getOption("example_maxtime")>1.6) {
  fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
                  +Matern(1|longitude+latitude),method="HL(0,1)",
                 data=Loaloa,family=binomial(),fixed=list(nu=0.5))
}

if (spaMM.getOption("example_maxtime")>5.8) {
  corrHLfit(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
            +Matern(1|longitude+latitude),
              data=Loaloa,family=binomial(),ranFix=list(nu=0.5))
}
if (spaMM.getOption("example_maxtime")>2.5) {
  fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
            +Matern(1|longitude+latitude),
              data=Loaloa,family=binomial(),fixed=list(nu=0.5),method="REML")
}

## Diggle and Ribeiro (2007) assumed (in this package notation) Nugget=2/7:
if (spaMM.getOption("example_maxtime")>7) {
  corrHLfit(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),
             data=Loaloa,family=binomial(),ranFix=list(nu=0.5,Nugget=2/7))
}
```

```
if (spaMM.getOption("example_maxtime")>1.3) {
  fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),method="REML",
             data=Loaloa,family=binomial(),fixed=list(nu=0.5,Nugget=2/7))
}

## with nugget estimation:
if (spaMM.getOption("example_maxtime")>17) {
  corrHLfit(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),
             data=Loaloa,family=binomial(),
             init.corrHLfit=list(Nugget=0.1),ranFix=list(nu=0.5))
}
if (spaMM.getOption("example_maxtime")>5.5) {
  fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
           +Matern(1|longitude+latitude),
             data=Loaloa,family=binomial(),method="REML",
             init=list(Nugget=0.1),fixed=list(nu=0.5))
}
```

LRT *Likelihood ratio test of fixed effects.*

Description

LRT performs a likelihood ratio (LR) test between two model fits, the "full" and the "null" model fits, currently differing only in their fixed effects. Parametric bootstrap p-values can be computed, either using the raw bootstrap distribution of the likelihood ratio, or a bootstrap estimate of the Bartlett correction of the LR statistic. This function differ from fixedLRT in its arguments (model fits for LRT, but all arguments required to fit the models for fixedLRT), and in the format of its return value. The function will stop or return possibly incorrect results for models differing beyond their fixed effects. By conceptual drift, anova works as an alias for LRT.

Usage

```
## S3 method for class 'HLfit'
anova(object, object2, ..., method="")
LRT(object, object2, boot.repl = 0, resp_testfn = NULL,
    simuland = eval_replicate,
    #     many further arguments can be passed to spaMM_boot via the '...'
    #     These include arguments for parallel computations, such as
    # nb_cores, fit_env,
    #     as well as other named arguments and spaMM_boot's own '...'
    ...)
```

Arguments

object,object2	Two models fits being compared (their order does not matter).
boot.repl	the number of bootstrap replicates.
resp_testfn	See argument resp_testfn of spaMM_boot
method	For development purposes, not documented.
simuland	a function, passed to spaMM_boot. See argument eval_replicate for default value and requirements.
...	Further arguments, passed to spaMM_boot since version 3.1.0.

Details

A raw bootstrap p-value can be computed from the simulated distribution as (1+sum(t >= t0))/(N+1) where t0 is the original likelihood ratio, t the vector of bootstrap replicates and N its length. See Davison & Hinkley (1997, p. 141) for discussion of the adjustments in this formula. However, a computationally more economical use of the bootstrap is to provide a Bartlett correction for the likelihood ratio test in small samples. According to this correction, the mean value m of the likelihood ratio statistic under the null hypothesis is computed (here estimated by a parametric bootstrap) and the original LR statistic is multiplied by n/m where n is the number of degrees of freedom of the test.

Value

An object of class fixedLRT, actually a list with as-yet unstable format, but here with typical elements (depending on the options)

fullfit	the HLfit object for the full model;
nullfit	the HLfit object for the null model;
basicLRT	A data frame including values of the likelihood ratio chi2 statistic, its degrees of freedom, and the p-value;

and, if a bootstrap was performed:

rawBootLRT	A data frame including values of the likelihood ratio chi2 statistic, its degrees of freedom, and the raw bootstrap p-value;
BartBootLRT	A data frame including values of the Bartlett-corrected likelihood ratio chi2 statistic, its degrees of freedom, and its p-value;
bootInfo	a list with the following elements:

bootreps A table of fitted likelihoods for bootstrap replicates;

meanbootLRT The mean likelihood ratio chi-square statistic for boostrap replicates;

References

Bartlett, M. S. (1937) Properties of sufficiency and statistical tests. Proceedings of the Royal Society (London) A 160: 268-282.

Davison A.C., Hinkley D.V. (1997) Bootstrap methods and their applications. Cambridge Univ. Press, Cambridge, UK.

See Also

See also fixedLRT.

Examples

```
data("wafers")
## Gamma GLMM with log link
m1 <- HLfit(y ~X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch),family=Gamma(log),
          resid.model = ~ X3+I(X3^2) ,data=wafers,method="ML")
m2 <- update(m1,formula.= ~ . -I(X2^2))
anova(m1,m2)

# Using resp_testfn argument:
## Not run:
set.seed(1L)
d <- data.frame(success = rbinom(10, size = 1, prob = 0.9), x = 1:10)
xx <- cbind(1,d$x)
table(d$success)
m_x <- fitme(success ~ x, data = d, family = binomial())
m_0 <- fitme(success ~ 1, data = d, family = binomial())
anova(m_x, m_0, boot.repl = 100,
      resp_testfn=function(y) {! is_separated(xx,as.numeric(y),verbose=FALSE)})

## End(Not run)
```

---

make_scaled_dist *Scaled distances between unique locations*

Description

This fonction computes scaled distances from whichever relevant argument it can use (see Details). The result can directly by used as input for computation of the Matérn correlation matrix. It is usually called internally by HLCor, so that users may ignore it, except if they wish to control the distance used through `control.dist$method`, or the parametrization of the scaling through `control.dist$rho.mapping`. `control.dist$method` provide access to the distances implemented in the `proxy` package, as well as to "EarthChord" and "Earth" methods defined in `spaMM` (see Details).

Usage

```
make_scaled_dist(uniqueGeo, uniqueGeo2=NULL, distMatrix, rho,
                 rho.mapping=seq_len(length(rho)),
                 dist.method="Euclidean",
                 return_matrix=FALSE)
```

Arguments

`uniqueGeo` A matrix of geographical coordinates (e.g. 2 columns for latitude and longitude), without replicates of the same location.

`uniqueGeo2` NULL, or a second matrix of geographical coordinates, without replicates of the same location. If NULL, scaled distances among `uniqueGeo` locations are computed. Otherwise, scaled distances between locations in the two input matrices are computed.

`distMatrix` A distance matrix.

`rho` A scalar or vector of positive values. Scaled distance is computed as `<distances in each coordinate> * rho`, unless a non-trivial `rho.mapping` is used.

`rho.mapping` A set of indices controlling which elements of the `rho` scale vector scales which dimension(s) of the space in which (spatial) correlation matrices of random effects are computed. Scaled distance is generally computed as `<distances in each coordinate> * rho[rho.mapping]`. As shown inthe Example, if one wishes to combine isotropic geographical distance and some environmental distance, the coordinates being latitude, longitude and one environmental variable, the scaled distance may be computed as (say) `(lat,long,env) *rho[c(1,1,2)]` so that the same scaling `rho[1]` applies for both geographical coordinates. In this case, `rho` should have length 2 and `rho.mapping` should be `c(1,1,2)`.

`dist.method` `method` argument of `proxy::dist` function (by default, `"Euclidean"`, but other distances are possible (see Details).

`return_matrix` Whether to return a `matrix` rather than a `proxy::dist` or `proxy::crossdist` object.

Details

The function uses the `distMatrix` argument if provided, in which case rho must be a scalar. Vectorial `rho` (i.e., different scaling of different dimensions) is feasible only by providing `uniqueGeo`.

The `dist.method` argument gives access to distances implemented in the `proxy` package, or to user-defined ones that are made accessible to `proxy` through its database. Of special interest for spatial analyses are distances computed from longitude and latitude (`proxy` implements `"Geodesic"` and `"Chord"` distances but they do not use such coordinates: instead, they use Euclidean distance for 2D computations, i.e. Euclidean distance between points on a circle rather than on a sphere). spaMM implements two such distances: `"Earth"` and `"EarthChord"`, using longitude and latitude inputs **in that order** (see Examples). The `"EarthChord"` distance is the 3D Euclidean distance "through Earth". The `"Earth"` distance is also known as the orthodromic or great-circle distance, on the Earth surface. Both distances return values in km and are based on approximating the Earth by a sphere of radius 6371.009 km.

Value

A matrix or `dist` object. If there are two input matrices, rows of the return value correspond to rows of the first matrix.

Examples

```
data("blackcap")
## a biologically not very meaningful, but syntactically correct example of rho.mapping
fitme(migStatus ~ 1 + Matern(1|latitude+longitude+means),
      data=blackcap, fixed=list(nu=0.5,phi=1e-6),
      init=list(rho=c(1,1)), control.dist=list(rho.mapping=c(1,1,2)))
## Using orthodromic distances: order of variables in Matern(.|longitude+latitude) matters
fitme(migStatus ~ 1 + Matern(1|longitude+latitude),data=blackcap,
      method="ML", fixed=list(nu=0.5,phi=1e-6),
      control.dist=list(dist.method="Earth"))
```

---

mapMM *Colorful plots of predicted responses in two-dimensional space.*

Description

These functions provide either a map of predicted response in analyzed locations, or a predicted surface. `mapMM` is a straightforward representation of the analysis of the data, while `filled.mapMM` uses interpolation to cope with the fact that all predictor variables may not be known in all locations on a fine spatial grid. Both functions takes an `HLfit` object as input. `mapMM` calls `spaMMplot2D`, which is similar but takes a more conventional (x,y,z) input.

Using `filled.mapMM` may involve questionable choices. Plotting a filled contour generally requires prediction in non-observed locations, where predictor variables used in the original data analysis may be missing. In that case, the original model formula cannot be used and an alternative model (controlled by the `map.formula` argument) must be used to interpolate (not smooth) the predicted values in observed locations (these predictions still resulting from the original analysis based on predictor variables). `filled.mapMM` always performs such interpolation (it does not allow one to provide values for the predictor variables). As a result (1) `filled.mapMM` will be slower than a mere plotting function, since it involves the analysis of spatial data; (2) the results may have little useful meaning if the effect of the original predictor variables is not correctly represented by this interpolation step. For example, prediction by interpolation may be biased in a way analogous to prediction of temperature in non-observed locations while ignoring effect of variation in altitude in such locations. Likewise, the`variance` argument of `filled.mapMM` allows one only to plot the prediction variance of its own interpolator, rather than that of the input object.

Usage

```
spaMMplot2D(x, y, z,xrange=range(x, finite = TRUE),
      yrange=range(y, finite = TRUE),
      margin=1/20,add.map= FALSE, nlevels = 20,
      color.palette = spaMM.colors,map.asp=NULL,
      col = color.palette(length(levels) - 1),
      plot.title=NULL, plot.axes=NULL, decorations=NULL,
      key.title=NULL, key.axes=NULL, xaxs = "i",
      yaxs = "i", las = 1, axes = TRUE, frame.plot = axes, ...)

mapMM(fitobject,Ztransf=NULL,coordinates,
```

```
      add.points,decorations=NULL,plot.title=NULL,plot.axes=NULL,envir=-3, ...)

filled.mapMM(fitobject, Ztransf = NULL, coordinates, xrange = NULL,
             yrange = NULL, margin = 1/20, map.formula, phi =
             1e-05, gridSteps = 41, decorations =
             quote(points(pred[, coordinates], cex = 1, lwd = 2)),
             add.map = FALSE, axes = TRUE, plot.title = NULL,
             plot.axes = NULL, map.asp = NULL, variance = NULL,
             var.contour.args = list(), smoothObject = NULL, ...)
```

Arguments

`fitobject` The return object of a corrHLfit or fitme call.

`x,y,z` Three vectors of coordinates, with `z` being expectedly the response.

`Ztransf` A transformation of the predicted response, given as a function whose only required argument can be a one-column matrix. The name of this argument must be `Z` (not `x`), as is appropriate for use in `do.call(Ztransf,list(Z=Zvalues))`.

`coordinates` The geographical coordinates. By default they are deduced from the model formula. For example if this formula is `resp ~ 1 + Matern(1| x + y )` the default coordinates are c("x","y"). If this formula is `resp ~ 1 + Matern(1| x + y + z )`, the user must choose two of the three coordinates.

`xrange` The `x` range of the plot (a vector of length 2); by default defined to cover all analyzed points.

`yrange` The `y` range of the plot (a vector of length 2); by default defined to cover all analyzed points.

`margin` This controls how far (in relative terms) the plot extends beyond the `x` and `y` ranges of the analyzed points, and is overriden by explicit `xrange` and `yrange` arguments.

`map.formula` NULL, or a formula whose left-hand side is ignored. Provides the formula used for interpolation. If NULL, a default formula with the same spatial effect(s) as in the input `fitobject` is used.

`phi` This controls the phi value assumed in the interpolation step. Ideally `phi` would be zero, but problems with numerically singular matrices may arise when `phi` is too small.

`gridSteps` The number of levels of the grid of x and y values

`variance` Either NULL, or the name of a component of variance of prediction *by the interpolator* to be plotted. Must name one of the components that can be returned by `predict.HLfit`. `variance="predVar"` is suitable for uncertainty in point prediction.

`var.contour.args`
A list of control parameters for rendering of prediction variances. See `contour` for possible arguments (except `x`, `y`, `z` and `add`).

`add.map` Either a boolean or an explicit expression, enclosed in `quote` (see Examples). If `TRUE`, the `map` function from the `maps` package (which much therefore the loaded) is used to add a map from its default `world` database. `xrange` and yrange are used to select the area, so it is most convenient if the coordinates are longitude and latitude (in this order and in standard units). An explicit expression can also be used for further control.

levels
: a set of levels which are used to partition the range of z. Must be strictly increasing (and finite). Areas with z values between consecutive levels are painted with the same color.

nlevels
: if levels is not specified, the range of z, values is divided into *approximately* this many levels (a call to pretty determines the actual number of levels).

color.palette
: a color palette function to be used to assign colors in the plot.

map.asp
: the y/x aspect ratio of the 2D plot area (not of the full figure including the scale). By default, the scales for x and y are identical unless the x and y ranges are too different. Namely, the scales are identical if (plotted y range)/(plotted x range) is 1/4 < . < 4, and map.asp is 1 otherwise.

col
: an explicit set of colors to be used in the plot. This argument overrides any palette function specification. There should be one less color than levels

plot.title
: statements which add titles to the main plot. See Details for differences between functions.

plot.axes
: statements which draw axes (and a box) on the main plot. See Details for differences between functions.

decorations
: Either NULL or Additional graphic statements (points, polygon, etc.), enclosed in quote (the default value illustrates the latter syntax). .

add.points
: Obsolete, use decorations instead.

envir
: Controls the environment in which plot.title, plot.axes, and decorations are evaluated. mapMM calls spaMM2Dplot from where these graphic arguments are evaluated, and the default value -3 means that they are evaluated within the environment from where mapMM was called.

key.title
: statements which add titles for the plot key.

key.axes
: statements which draw axes on the plot key.

xaxs
: the x axis style. The default is to use internal labeling.

yaxs
: the y axis style. The default is to use internal labeling.

las
: the style of labeling to be used. The default is to use horizontal labeling.

axes, frame.plot
: logicals indicating if axes and a box should be drawn, as in plot.default.

smoothObject
: Either NULL, or an object inheriting from class HLfit (hence, an object on which predict.HLfit can be called), predicting the response surface in any coordinates. See Details for typical usages.

...
: further arguments passed to or from other methods. For mapMM, all such arguments are passed to spaMMplot2D; for spaMMplot2D, currently only additional graphical parameters passed to title() (see Details). For filled.mapMM, these parameters are those that can be passed to spaMM.filled.contour.

Details

The smoothObject argument may be used to redraw a figure faster by recycling the predictor of the response surface returned invisibly by a previous call to filled.mapMM.

For smoothObject=NULL (the default), filled.mapMM interpolates the predicted response, with sometimes unpleasant effects. For example, if one interpolates probabilities, the result may not be within [0,1], and then (say) a logarithmic Ztransf may generate NaN values that would otherwise not occur. The smoothObject argument may be used to overcome the default behaviour, by providing an alternative predictor.

If you have values for all predictor variables in all locations of a fine spatial grid, filled.mapMM may not be a good choice, since it will ignore that information (see map.formula argument). Rather, one should use predict(<fitobject>,newdata= <all predictor variables >) to generate all predictions, and then either spaMM.filled.contour or some other raster functions.

The different functions are (currently) inconsistent among themselves in the way they handle the plot.title and plot.axes argument:

spaMM.filled.contour behaves like graphics::filled.contour, which (1) handles arguments which are calls such as title(.) or {axis(1);axis(2)}; (2) ignores ... arguments if plot.title is missing; and (3) draws axes by default when plot.axes is missing, given axes = TRUE.

By contrast, **filled.mapMM** handles arguments which are language expressions such as produced by quote(.) or substitute(.) (see Examples).

mapMM can handles language expressions, but also accepts at least some calls.

Value

filled.mapMM returns invisibly a predictor of the response surface. mapMM has no return value. Plots are produced as side-effects.

See Also

http://kimura.univ-montp2.fr/~rousset/spaMM/example_raster.html for more elaborate plot procedures.

Examples

```
data("blackcap")
bfit <- fitme(migStatus ~ means+ Matern(1|longitude+latitude),data=blackcap,
              fixed=list(lambda=0.5537,phi=1.376e-05,rho=0.0544740,nu=0.6286311))
  mapMM(bfit,color.palette = function(n){spaMM.colors(n,redshift=1/2)},add.map=TRUE)

if (spaMM.getOption("example_maxtime")>1) {
  ## filled.mapMM takes a bit longer
  # showing 'add.map', 'nlevels', and contour lines for 'variance'
  filled.mapMM(bfit, nlevels=30, add.map=TRUE, plot.axes=quote({axis(1);axis(2)}),
             variance="respVar",
             plot.title=title(main="Inferred migration propensity of blackcaps",
                              xlab="longitude",ylab="latitude"))
}

if (spaMM.getOption("example_maxtime")>3) {
```

```
 data("Loaloa")
 lfit <- fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
                  +Matern(1|longitude+latitude), method="PQL", data=Loaloa,
                  family=binomial(), fixed=list(nu=0.5,rho=2.255197,lambda=1.075))

 ## longer computation requiring interpolation of 197 points
 filled.mapMM(lfit,add.map=TRUE,plot.axes=quote({axis(1);axis(2)}),
             decorations=quote(points(pred[,coordinates],pch=15,cex=0.3)),
             plot.title=title(main="Inferred prevalence, North Cameroon",
                              xlab="longitude",ylab="latitude"))
}
```

MaternCorr *Matern correlation function and Matern formula term.*

Description

The Matérn correlation function describes realizations of Gaussian spatial processes with different smoothnesses (i.e. either smooth or rugged surfaces, controlled by the ν parameter). It also includes a ρ scaling parameter and an optional 'nugget' parameter. A random effect specified in a model formula as `Matern(1|<...>)` has pairwise correlations given by the Matérn function at the scaled Euclidean distance between coordinates specified in <...>, using "+" as separator (e.g., `Matern(1|latitude + longitude)`). The Matern family can be used in Euclidean spaces of any dimension; and also for correlations on a sphere (with maximum smoothness `nu=0.5`).

By default, `fitme` and `corrHLfit` performs optimization over the ρ and ν parameters. It is possible to estimate different scaling parameters for the different Euclidean dimensions: see examples in `make_scaled_dist`.

The `MaternCorr` function may be used to vizualise these correlations, using distances as input.

Usage

```
## Default S3 method:
MaternCorr(d, rho = 1, smoothness, nu = smoothness, Nugget = NULL)
# Matern(1|...)
```

Arguments

`d`	A distance or a distance matrix.
`rho`	A scaling factor for distance. The 'range' considered in some formulations is the reciprocal of this scaling factor
`smoothness`	The smoothness parameter, >0. $\nu = 0.5$ corresponds to the exponential correlation function, and the limit function when μ goes to ∞ is the squared exponential function (as in a Gaussian).
`nu`	Same as smoothness

Nugget (Following the jargon of Kriging) a parameter describing a discontinuous decrease in correlation at zero distance. Correlation will always be 1 at $d = 0$, and from which it immediately drops to (1-Nugget)

... Names of coordinates, using "+" as separator (e.g., `Matern(1|latitude + longitude)`

Details

The correlation at distance $d > 0$ is

$$(1 - \text{Nugget})\frac{(\rho d)^{\nu} K_{\nu}(\rho d)}{2^{(\nu-1)}\Gamma(\nu)}$$

where K_{ν} is the `besselK` function of order ν.

By default the Nugget is set to 0. See one of the examples on data set `Loaloa` for a fit including the estimation of the Nugget.

Value

Scalar/vector/matrix depending on input.

References

Stein, M.L. (1999) Statistical Interpolation of Spatial Data: Some Theory for Kriging. Springer, New York.

See Also

See `corMatern` for an implementation of this correlation function as a `corSpatial` object for use with `lme` or `glmmPQL`.

Examples

```
## See examples in help("HLCor"), help("Loaloa"),  help("make_scaled_dist"), etc.
## Matern correlations in 4-dimensional space:
set.seed(123)
randpts <- matrix(rnorm(20),nrow=5)
distMatrix <- as.matrix(proxy::dist(randpts))
MaternCorr(distMatrix,nu=2)
```

`mat_sqrt` *Computation of "square root" of symmetric positive definite matrix*

Description

mat_sqrt is not usually directly called by users, but arguments may be passed to it through higher-level calls (see Examples). For given matrix **C**, it computes a factor **L** such that **C** = **L** * t(**L**), handling issues with nearly-singular matrices. The default behavior is to try Cholesky factorization, and use eigen if it fails. Matrix roots are not unique (for example, they are lower triangular for t(chol(.)), and symmetric for svd(.). As matrix roots are used to simulate samples under the fitted model (in particular in the parametric bootstrap implemented in fixedLRT), this implies that for given seed of random numbers, these samples will differ with these different methods (although their distribution should be identical).

Usage

```
mat_sqrt(m = NULL, symSVD = NULL, try.chol = TRUE, condnum=1e12)
```

Arguments

m	The matrix which 'root' is to be computed. This argument is ignored if symSVD is provided.
symSVD	A list representing the symmetric singular value decomposition of the matrix which 'root' is to be computed. Must have elements $u, a matrix of eigenvectors, and $d, a vector of eigenvalues.
try.chol	If try.chol=TRUE, the Cholesky factorization will be tried.
condnum	(large) numeric value. In the case chol() was tried and failed, the matrix is regularized so that its (matrix 2-norm) condition number is reduced to condnum.

Value

For non-NULL m, its matrix root, with rows and columns labelled according to the columns of the original matrix. If eigen was used, the symmetric singular value decomposition (a list with members u (matrix of eigenvectors) and d (vector of eigenvalues)) is given as attribute.

Examples

```
## Not run:
## try.chol argument passed to mat_sqrt
## through the '...' argument of higher-level functions
## such as HLCor, corrHLfit, fixedLRT:
data("scotlip")
HLCor(cases~I(prop.ag/10) +adjacency(1|gridcode)+offset(log(expec)),
      ranPars=list(rho=0.174),adjMatrix=Nmatrix,family=poisson(),
      data=scotlip,try.chol=FALSE)

## End(Not run)
```

method *Fitting methods (objective functions maximized)*

Description

Many approximations for likelihood have been defined to fit mixed models (e.g. Noh and Lee (2007) for some overview), and fitting functions in spaMM implement several of them, and some additional ones. In particular, PQL as originally defined by Breslow and Clayton (1993) uses REML to estimate dispersion parameters, but spaMM allows one to use an ML variant of PQL. Moreover, it allows some non-standard specification of the model formula that determines the conditional distribution used in REML.

EQL stands for the EQL method of Lee and Nelder (2001). The '+' version includes the d v/ d tau correction described p. 997 of that paper, and the '-' version ignores it. PQL can be seen as the version of EQL- for GLMMs. It estimates fixed effects by maximizing h-likelihood and dispersion parameters by an approximation of REML, i.e. by maximization of an approximation of restricted likelihood. PQL/L is PQL without the leverage corrections that define REML estimation of random-effect parameters. Thus, it estimates dispersion parameters by an approximation of marginal likelihood.

The `method` (or `HLmethod`) argument of fitting functions also accepts value of the form `"HL(<...>)"`, `"ML(<...>)"` and `"RE(<...>)"`, e.g. `method="RE(1,1)"`, which allow a more direct specification of the approximations used. HL and RE are equivalent (both imply an REML correction). The first '1' means that a first order Laplace approximation to the likelihood is used to estimate fixed effects (a '0' would instead mean that the h likelihood is used as the objective function). The second '1' means that a first order Laplace approximation to the likelihood or restricted likelihood is used to estimate dispersion parameters, this approximation including the dv/d tau term specifically discussed by Lee & Nelder 2001, p. 997 (a '0' would instead mean that these terms are ignored).

It is possible to enforce the EQL approximation for estimation of dispersion parameter (i.e., Lee and Nelder's (2001) method) by adding a third index with value 0. `"EQL+"` is thus `"HL(0,1,0)"`, while `"EQL-"` is `"HL(0,0,0)"`. `"PQL"` is EQL- for GLMMs. `"REML"` is `"HL(1,1)"`. `"ML"` is `"ML(1,1)"`.

Some of these distinctions make sense for **GLMs**, and `glm` methods use approximations, which make a difference for Gamma GLMs. This means in particular that, (as stated in `stats::logLik`) the logLik of a Gamma GLM fit by `glm` differs from the exact likelihood. Further, the dispersion estimate returned by `summary.glm` differs from the one implied by `logLik`, because `summary.glm` uses Pearson residuals instead of deviance residuals. This may be confusing, and no `method` in spaMM tries to reproduce simultaneously these distinct features (however, `spaMM_glm` may do so). An `"ML(0,0,0)"` approximation of true ML provides the same log likelihood as `stats::logLik`, and the dispersion estimate returned by an `"HL(.,.,0)"` fit matches what can be computed from residual deviance and residual degrees of freedom of a glm fit, but this is not the estimate displayed by `summary.glm`. With a log link, the fixed effect estimates are unaffected by these distinctions.

References

Breslow, NE, Clayton, DG. (1993). Approximate Inference in Generalized Linear Mixed Models. Journal of the American Statistical Association 88, 9-25.

Lee, Y., Nelder, J. A. (2001) Hierarchical generalised linear models: A synthesis of generalised linear models, random-effect models and structured dispersions. Biometrika 88, 987-1006.

Noh, M., and Lee, Y. (2007). REML estimation for binary data in GLMMs, J. Multivariate Anal. 98, 896-915.

MSFDR *Multiple-Stage False Discovery Rate procedure*

Description

This implements the procedure described by Benjamini and Gavrilov (2009) for model-selection **of fixed-effect terms** based on False Discovery Rate (FDR) concepts. It uses forward selection based on penalized likelihoods. The penalization for the number of parameters is distinct from that in Akaike's Information Criterion, and variable accross iterations of the algorithm (but functions from the `stats` package for AIC-based model-selection are still called, so that some screen messages refer to AIC).

Usage

```
MSFDR(nullfit, fullfit, q = 0.05, verbose = TRUE)
```

Arguments

`nullfit`	An ML fit to the minimal model to start the forward selection from; an object of class `HLfit`.
`fullfit`	An ML fit to the maximal model; an object of class `HLfit`.
`q`	Nominal error rate of the underlying FDR procedure (expected proportion of incorrectly rejected null out of the rejected). Benjamini and Gavrilov (2009) recommend `q=0.05` on the basis of minimizing mean-squared prediction error in various simulation conditions considering only linear models.
`verbose`	Whether to print information about the progress of the procedure.

Value

The fit of the final selected model; an object of class `HLfit`.

References

A simple forward selection procedure based on false discovery rate control. Ann. Appl. Stat, 3, 179-198 (2009).

Examples

```
if (spaMM.getOption("example_maxtime")>1.4) {
data("wafers")
nullfit <- fitme(y~1+(1|batch), data=wafers,family=Gamma(log))
fullfit <- fitme(y ~X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch), data=wafers, family=Gamma(log))
MSFDR(nullfit=nullfit,fullfit=fullfit)
}
```

multIMRF *Interpolated Markov Random Field models*

Description

IMRF is a syntax to specify random-effect terms of the forms considered by Lindgren et al. (2011) or Nychka et al. (2015, 2019). For example, using IMRF with its model argument provides good approximations of random effects with Matern correlation structure win smoothness=1.

The random effects considered here all involve a multivariate Gaussian random effect over a lattice, from which the random-effect value in any spatial position is determined by interpolation of values on the lattice. **IMRF** stands for **I**nterpolated **M**arkov **R**andom **F**ield because the specific process considered on the lattice is currently known as a Gaussian Markov Random Field (see the Details for further information). Lindgren et al. considered irregular lattices that can be specified by the model argument, while Nychka et al. considered regular grids that can be specified by the other arguments.

The multIMRF syntax implements the multiresolution model of Nychka et al. Any multIMRF term in a formula is immediately converted to IMRF terms, which should be counted as distinct random effects for all purposes (e.g., for fixing the variances of other random effects). However, the arguments that control multIMRF terms are lists with names referring to successive multIMRF terms in the un-expanded formula, not to successive random-effect terms in the expanded formula.

Usage

```
# IMRF( 1 | <coordinates>, model, nd, m, no, ce, ...)
# multIMRF( 1 | <coordinates>, levels, margin, coarse=10L,
#             norm=TRUE, centered=TRUE )
```

Arguments

model	An inla.spde2 object as produced by INLA::inla.spde2.matern (see Examples below, and http://www.r-inla.org for further information).
levels	integer; Number of levels in the hierarchy, i.e. number of component IMRFs.
margin, m	integer; width of the margin, as a number of additional grid points on each side (applies to all levels of the hierarchy).
coarse	integer; number of grid points (excluding the margins) per dimension for the coarsest IMRF. The number of grids steps nearly doubles with each level of the hierarchy (see Details).
nd	integer; number of grid steps (excluding the margins) per dimension for the given IMRF.
norm, no	Boolean; whether to apply normalization (see Details), or not.
centered, ce	Boolean; whether to center the grid in all dimensions, or not.
...	Not documented, for programming purposes

Details

Gaussian Markov Random Field (MRF) and conditional autoregressive models are essentially the same thing, apart from details of specification. adjacency and AR1 random effects can be seen as specific MRFs. The common idea is the Markov-like property that the distribution of each element b_i of the random-effect **b**, given values of a few specific elements (the "neighbours" of i), is independent of other elements (i.e., of non-neighbours). The non-zero non-diagonal elements of a precision matrix characterize the neighbours.

Given the inferred vector **b** of values of the MRF on the lattice, the interpolation of the MRF in any focal point is of the form **Ab** where each row of **A** weights the values of **b** according to the position of the focal point relative to the vertices of the lattice. Following the original publications, for regular grids (NULL `model`), the weights are computed as <Wendland function>(<scaled Euclidean distances beween focal point and vertices>); and for grids given by `model=<inla.spde2 object>`, the non-zero weights are the barycentric coordinates of the focal point in the enclosing triangle from the mesh triangulation (points from outside the mesh would have zero weights, so the predicted effect **Ab=0**).

The `IMRF` model defines both a lattice in space, the precision matrix for a Gaussian MRF over this lattice, and the **A** weights. The full specification of the MRF on **irregular lattices** is complex. The κ parameter considered by `spaMM` is the κ considered by Lindgren et al.; only the case *d=2* and $alpha = 1$ or 2, approximating a Matérn correlation model with $nu = 0$ or $nu = 1$ ($alpha = nu + d/2$), is currently implemented in `spaMM`.

For the MRFs on **regular grids** implemented here, the precision matrix is defined (up to a variance parameter) as **M'M** where the diagonal elements m_{ii} of **M** are 4+κ^2 and the m_{ij} for the four nearest neighbours are -1 (note that **M'M** involves more than these four neighbours).

The precision matrix defined in this way is the inverse of an heteroscedastic covariance matrix **C**, but by default a normalization is applied so that the random effect is homoscedastic. As for other random effects, the variance is further controlled by a multiplicative factor λ. The **ormalization** is as follows: the design matrix of the random effect term is viewed as **WAL** where **W** is a diagonal normalization matrix, **A** is the above-described weight matrix, and **L** is a "square root" of **C**. If no normalization is applied, the covariance matrix of the random effect is of the form λ**ALL'A'**, which is heteroscedastic; λ may then be quite different from the marginal variance of the random effect, and is difficult to describe in a simple way. Hence, by default, **W** is defined such that **WALL'A'W'** has unit diagonal; then, λ is the marginal variance of the random effect.

By default, the IMRF lattice is rectangular (currently the only option) and is made of a core lattice, to which margins of `margin` steps are added on each side. The core lattice is defined as follows: in each of the two spatial dimensions, the range of axial coordinates is determined. The largest range is divided in `nd-1` steps, determining `nd` values and step length L. The other range is divided in steps of the same length L. If it extends over (say) $2.5L$, a grid of 2 steps and 3 values is defined, and by default centered on the range (the extreme points therefore typically extend slightly beyond the grid, within the first of the additional steps defined by the `margin`; if not centered, the grid start from the lower coordinate of the range).

`multIMRF` implements multilevel IMRFs. It defines a sequence of IMRFs, with progressively finer lattices, a common κ value `hy_kap` for these IMRFs, and a single variance parameter `hy_lam` that determines λ values decreasing by a factor of 4 for successive IMRF terms. By default, each component `IMRF` is normalized independently as described above (as in Nychka et al. 2019), and `hy_lam` is the sum of the variances of these terms (e.g., if there are three levels and `hy_lam=1`, the successive variances are (1,1/4,1/16)/(21/16)). The `nd` of the first IMRF is set to the `coarse` value, and its lattice is defined accordingly. If `coarse=4` and `margin=5`, a grid of 14 coordinates is therefore defined over the largest range. In the second IMRF, the grid spacing is halved, so that new steps are defined halfway between the previous ones (yielding a grid of 27 stepin the widest range). The third IMRF proceeds from the second in the same way, and so on.

To control initial or fixed values of `multIMRF` κ and variance parameters, the `hyper` syntax shown in the Examples should be used. `hyper` possible elements are named `"1"`, `"2"`, ... referring to successive `multIMRF` terms in the formula.

References

D. Nychka, S. Bandyopadhyay, D. Hammerling, F. Lindgren, S. Sain (2015) A multiresolution gaussian process model for the analysis of large spatial datasets. Journal of Computational and Graphical Statistics 24 (2), 579-599. doi: 10.1080/10618600.2014.914946

D. Nychka, D. Hammerling, Mitchel. Krock, A. Wiens (2018) Modeling and emulation of nonstationary Gaussian fields. Spat. Stat. 28: 21-38. doi: 10.1016/j.spasta.2018.08.006

Lindgren F., Rue H., Lindström J. (2011) An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach Journal of the Royal Statistical Society: Series B (Statistical Methodology), 73: 423-498. doi: 10.1111/j.1467-9868.2011.00777.x

Examples

```
if (spaMM.getOption("example_maxtime")>6) {

data("blackcap") ## toy examples; but IMRF may be useful only for much larger datasets

########################## Irregular lattice specified by 'model':
#
data("small_spde") ## load object of class 'inla.spde2', created and saved by :
  # spd <- sp::SpatialPointsDataFrame(coords = blackcap[, c("longitude", "latitude")],
  #                             data = blackcap)
  # small_mesh <- INLA::inla.mesh.2d(loc = INLA::inla.mesh.map(sp::coordinates(spd)),
  #                            max.n=100, # only for demonstration purposes
  #                            max.edge = c(3, 20))
  # small_spde <- INLA::inla.spde2.matern(small_mesh)
  # save(small_spde, file="small_spde.RData", version=2)
#
fit_SPDE <- fitme(migStatus ~ means + IMRF(1|longitude+latitude, model=small_spde),
                  data=blackcap)

########################## Regular lattices:
#
#Using 'hyper' to control fixed hyper-parameters
#
(mrf <- fitme(migStatus ~ 1 + (1|pos) +
                          multIMRF(1|latitude+longitude,margin=5,levels=2),
              data=blackcap, fixed=list(phi=1,lambda=c("1"=0.5),
              hyper=list("1"=list(hy_kap=0.1,hy_lam=1)))) )
```

```
# Using 'hyper' to control initial hyper-parameters
#
(mrf <- fitme(migStatus ~ 1 + multIMRF(1|latitude+longitude,margin=5,levels=2),
                data=blackcap, method="ML", fixed =list(phi=1),
                init=list(hyper=list("1"=list(hy_kap=0.1,hy_lam=1)))) )

# *Independent* IMRF terms (often giving dubious results)
#
(mrf <- fitme(migStatus ~ 1 + IMRF(1|latitude+longitude,margin=5, nd=4L)
                              + IMRF(1|latitude+longitude,margin=5, nd=7L),
          data=blackcap,
          fixed=list(phi=1,lambda=c(1/4,1/16),
                      corrPars=list("1"=list(kappa=0.1),"2"=list(kappa=0.1)))))

}
```

`multinomial` *Analyzing multinomial data*

Description

These functions facilitate the conversion and analysis of multinomial data as as series of nested binomial data. The main function is `multi`, to be used in the `family` argument of the fitting functions. It calls `binomialize`, which can be called directly to check how the data are converted to nested binomial data. The `fitted.HLfitlist` method of the `fitted` generic function returns a matrix of fitted multinomial probabilities. The `logLik.HLfitlist` method of the `logLik` generic function returns a log-likelihood for the joint fits.

Usage

```
multi(binResponse=c("npos","nneg"),binfamily=binomial(),input="types",...)
binomialize(data,responses,sortedTypes=NULL,binResponse=c("npos","nneg"),
             depth=Inf,input="types")
## S3 method for class 'HLfitlist'
fitted(object,...)
## S3 method for class 'HLfitlist'
logLik(object,which,...)
```

Arguments

- `data` The data frame to be analyzed.
- `object` A list of binomial fits returned by a multinomial analysis
- `responses` column names of the data, such that `<data>[,<responses>]` contain the multinomial response data, as levels of factor variables.
- `sortedTypes` Names of multinomial types, i.e. levels of the multinomial response factors. Their order determines which types are taken first to define the nested binomial samples. By default, the most common types are considered first.

`binResponse` The names to be given to the number of "success" and "failures" in the binomial response.

`depth` The maximum number of nested binomial responses to be generated from the multinomial data.

`binfamily` The family applied to each binomial response.

`input` If `input="types"`, then the `responses` columns must contain factor levels of the binomial response. If `input="counts"`, then the `responses` columns must contain counts of different factor levels, and the column names are the types.

`which` Which element of the `APHLs` list to return. The default depends on the fitting method.In particular, if it was REML or one of its variants, the function returns the log restricted likelihood (exact or approximated).

`...` Other arguments passed from or to other functions.

Details

A multinomial response, say counts 17, 13, 25, 8, 3, 1 for types `type1` to `type6` can be represented as a series of nested binomials e.g. `type1` against others (17 vs 50) then among these 50 others, `type2` versus others (13 vs 37), etc. The `binomialize` function generates such a representation. By default the representation considers types in decreasing order of the number of positives, i.e. first `type3` against others (25 vs 42), then `type1` against others within these 42, etc. It stops if it has reached `depth` nested binomial responses. This can be modified by the `sortedTypes` argument, e.g. `sortedTypes=c("type6","type4","type2")`. `binomialize` returns a list of data frames which can be directly provided as a `data` argument for the fitting functions, with binomial response.

Alternatively, one can provide the multinomial response data frame, which will be internally converted to nested binomial data if the `family` argument is a call to `multinomial` (see examples).

For mixed models, the multinomial data can be fitted to a model with the same correlation parameters, and either the same or different variances of random effects, for all binomial responses. Which analysis is performed depends on the `init.corrHLfit` argument (see `corrHLfit` and the Examples).

Value

`binomialize` returns a list of data frames appropriate for analysis as binomial response. Each data frame contains the original one plus Two columns named according to `binResponse`. `multi` returns a list.

Examples

```
## An example considering pseudo-data at one diploid locus for 50 individuals
set.seed(123)
genecopy1 <- sample(4,size=50,prob=c(1/2,1/4,1/8,1/8),replace=TRUE)
genecopy2 <- sample(4,size=50,prob=c(1/2,1/4,1/8,1/8),replace=TRUE)
alleles <- c("122","124","126","128")
genotypes <- data.frame(type1=alleles[genecopy1],type2=alleles[genecopy2])
## Columns "type1","type2" each contains an allele type => input is "types" (the default)
datalist <- binomialize(genotypes,responses=c("type1","type2"))

## two equivalent fits:
```

```
f1 <- HLfit(cbind(npos,nneg)~1,data=datalist, family=binomial())
f2 <- HLfit(cbind(npos,nneg)~1,data=genotypes, family=multi(responses=c("type1","type2")))
fitted(f2)

## distinct fits for spatial data
## Not run:
genoInSpace <- data.frame(type1=alleles[genecopy1],type2=alleles[genecopy2],x=runif(50),y=runif(50))
## Fitting distinct variances of random effects for each binomial response
corrHLfit(cbind(npos,nneg)~1+Matern(1|x+y),data=genoInSpace,
          family=multi(responses=c("type1","type2")),
          ranFix=list(rho=1,nu=0.5))
## Fitting the same variance for all binomial responses
corrHLfit(cbind(npos,nneg)~1+Matern(1|x+y),data=genoInSpace,
          family=multi(responses=c("type1","type2")),
          ranFix=list(rho=1,nu=0.5),init.corrHLfit=list(lambda=1))

## End(Not run)
```

---

negbin — *Family function for GLMs and mixed models with negative binomial and zero-truncated negative binomial response.*

Description

family object that specifies the information required to fit a negative binomial generalized linear model, with known or unknown underlying Gamma shape parameter. The zero-truncated variant can be specified either as `Tnegbin(.)` or as `negbin(.,trunc = 0L)`.

Usage

```
negbin(shape = stop("negbin's 'shape' must be specified"), link = "log", trunc = -1L)
Tnegbin(shape = stop("negbin's 'shape' must be specified"), link = "log")
# (the shape parameter is actually not requested unless this is used in a glm() call)
```

Arguments

shape	Shape parameter of the underlying Gamma distribution, given that the `negbin` family can be represented as a Poisson-Gamma mixture, where the conditional Poisson mean is μ times a Gamma random variable with mean 1 and shape `shape` (as produced by `rgamma(.,shape=shape,scale=1/shape)`).
link	log, sqrt or identity link, specified by any of the available ways for GLM links (name, character string, one-element character vector, or object of class `link-glm` as returned by make.link).
trunc	Either `0L` for zero-truncated distribution, or `-1L` for default untruncated dsitribution.

Details

shape is the k parameter of McCullagh and Nelder (1989, p.373) and the theta parameter of Venables and Ripley (2002, section 7.4). The latent Gamma variable has mean 1 and variance 1/shape, and the negbin with mean mu has variance $mu + mu^2$/shape. The negbin family is sometimes called the NegBin1 model (as the first, historically) in the literature on negative binomial models, and sometimes the NegBin2 model (because its variance is a quadratic function of its mean).

spaMM does not handle models with the “other” negative-binomial response family where the variance is a linear function of the mean, because this is not an exponential-family model. However, it can approximate it, through a Laplace approximation and a bit of additional programming, as a Poisson-Gamma mixture model with an heteroscedastic Gamma random-effect, specified e.g. as (weights-1|.) where the weights need to be updated iteratively as function of predicted response. File test-negbin1.R in the /test directory provides one example. Other mean-variance relationship can be handled in the same way.

The name NB_shape should be used to set values of shape in control arguments of the fitting functions (e.g., fitme(.,init=list(NB_shape=1))).

Value

A family object.

References

McCullagh, P. and Nelder, J.A. (1989) Generalized Linear Models, 2nd edition. London: Chapman & Hall.

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S-PLUS. Fourth Edition. Springer.

Examples

```
## Fitting negative binomial model with estimated scale parameter:
data("scotlip")
fitme(cases~I(prop.ag/10)+offset(log(expec)),family=negbin(), data=scotlip)
negfit <- fitme(I(1+cases)~I(prop.ag/10)+offset(log(expec)),family=Tnegbin(), data=scotlip)
simulate(negfit,nsim=3)
```

---

options *spaMM options settings*

Description

Allow the user to set and examine a variety of *options* which affect operations of the spaMM package.

Usage

```
spaMM.options(..., warn = TRUE)

spaMM.getOption(x)
```

Arguments

x — a character string holding an option name.

warn — Boolean: whether to warn if a previously undefined options is being defined (a protection against typos).

... — A named value or a list of named values. The following values, with their defaults, are used in spaMM:

- **LevenbergM=NULL:** NULL or boolean. Whether to use a Levenberg-Marquardt-like algorithm (see Details) by default in most computations. But it is advised to use instead control.HLfit=list(LevenbergM=...) to control this on a case-by-case basis. The joint default behaviour is that Levenberg-Marquardt is used by default for binomial response data that takes only extreme values (in particular, for binary 0/1 response), and that for other models the fitting algorithm switches to it if divergence is suspected. FALSE inhibits its use; TRUE forces its use for all iterative least-square fits, except when 'confint()' is called.
- **example_maxtime=0.7:** Used in the documentation to control whether the longer examples should be run. The approximate running time of given examples on one author's laptop is compared to this value.
- **optimizer1D="optimize":** Optimizer for one-dimensional optimization. If you want to control the initial value, you should select another optimizer.
- **optimizer=".safe_opt":** Optimizer for optimization in several dimensions. Use optimizer="nloptr" to call nloptr with method "NLOPT_LN_BOBYQA"; use optimizer="bobyqa" to call bobyqa; and use optimizer="L-BFGS-B" to call optim with method "L-BFGS-B". The default ".safe_opt" uses nloptr except in some cases where it expects or detects problems with it (the source coe should be consulted for details). The optimizer can also be specified on a fit-by-fit basis as the value of control$optimizer in a fitme call, or as the value of control.corrHLfit$optimizer.
- **nloptr:** Default control values of nloptr calls.
- **bobyqa:** Default control values of bobyqa calls.
- **maxLambda=1e10:** The maximum value of lambda: higher fitted lambda values in HLfit are reduced to this. Since version 3.1.0, a much smaller lambda bound is deduced from maxLambda for COMPoisson and log-link response families.
- **regul_lev_lambda** Numeric (default: 1e-8); lambda leverages numerically 1 are replaced by 1- regul_lev_lambda
- **COMP_maxn:** Number of terms for truncation of infinite sums that are evaluated in the fitting of COMPoisson models.

CMP_asympto_cond: Condition for applying an approximation or the COM-Poisson response family, as detailed in COMPoisson.

Gamma_min_y=1e-10: A minimum response value in Gamma-response models; used to check data, and in simulate() to correct the simulation results.

QRmethod: A character string, to control whether dense matrix or sparse matrix methods are used in intensive matrix computations, overcoming the defaults choices made by spaMM in this respect. Possible values are "dense" and "sparse".

matrix_method: A character string, to control the factorization of dense model matrices. Default value is "def_sXaug_EigenDense_QRP_scaled". The source code should be consulted for further information.

Matrix_method: A character string, to control the factorization of sparse model matrices. Default value is "def_sXaug_Matrix_QRP_scaled". The source code should be consulted for further information.

stylefns: Default colors of some screen output (notably that of some fitting functions when called with argument verbose=c(TRACE=TRUE))

barstyle: Integer, or Boolean interpreted as Integer, or quoted expression evaluating to such types; controlling the display of some progress bars. If zero, no progress bar should be displayed; otherwise, a bar should be displayed. Further, when txtProgressBar is called, barstyle is passed as its style argument. Default is quote(if(interactive()) {3L} else {0L}).

and many other undocumented values for programming or development purposes. Additional options without default values can also be used (e.g., see sparse_precision).

Details

spaMM.options() provides an interface for changing maximal values of parameters of the Matérn correlation function. However, it is not recommended to change these values unless a spaMM message specifically suggests so.

By default spaMM use Iteratively Reweighted Least Squares (IRLS) methods to estimate fixed-effect parameters (jointly with predictions of random effects). However, a Levenberg-Marquardt algorithm, as described by Nocedal & Wright (1999, p. 266), is also implemented. The Levenberg-Marquardt algorithm is designed to optimize a single objective function with respect to all its parameters. It is thus well suited to compute a PQL fit, which is based on maximization of a single function, the h-likelihood. By contrast, in a fit of a mixed model by (RE)ML, one computes jointly fixed-effect estimates that maximizes marginal likelihood, and random-effect values that maximize h-likelihood given the fixed-effect estimates. The gradient of marginal likelihood with respect to fixed-effect coefficients does not generally vanishes at the solution (although it remains close to zero except in “difficult” cases with typically little information in the data). The Levenberg-Marquardt algorithm is not directly applicable in this case, as it may produce random-effect values that increases marginal likelihood rather than h-likelihood. The (RE)ML variant of the algorithm implemented in spaMM may therefore use additional nested h-likelihood-maximizing steps for correcting random-effect values. In version 3.1.0 this variant was revised for improved performance in difficult cases.

Value

For `spaMM.getOption`, the current value set for option `x`, or `NULL` if the option is unset.

For `spaMM.options()`, a list of all set options. For `spaMM.options(name)`, a list of length one containing the set value, or `NULL` if it is unset. For uses setting one or more options, a list with the previous values of the options changed (returned invisibly).

References

Jorge Nocedal and Stephen J. Wright (1999) Numerical Optimization. Springer-Verlag, New York.

Examples

```
  spaMM.options()
  spaMM.getOption("example_maxtime")
  ## Not run:
  spaMM.options(maxLambda=1e06)

## End(Not run)
```

`pedigree` *Fit mixed-effects models incorporating pedigrees*

Description

This illustrates how to use spaMM for quantitative genetic analyses. spaMM appears competitive in terms of speed for GLMMs with large data sets, particularly when using the PQL method, which may be a quite good approximation in such cases. For large pedigrees it may be useful to compute the inverse of the relationship matrix using some efficient ad hoc algorithm, then to provide it as argument of the fit using the `covStruct(list(precision=...))` syntax.

Examples

```
## Not run:
if(requireNamespace("pedigreemm", quietly=TRUE)) {
## derived from help("pedigreemm")
  p1 <- new("pedigree",
            sire = as.integer(c(NA,NA,1, 1,4,5)),
            dam  = as.integer(c(NA,NA,2,NA,3,2)),
            label = as.character(1:6))
  A <- pedigreemm::getA(p1) ## relationship matrix
  ## data simulation
  cholA <- chol(A)
  varU <- 0.4; varE <- 0.6; rep <- 20
  n <- rep*6
  set.seed(108)
  bStar <- rnorm(6, sd=sqrt(varU))
  b <- crossprod(as.matrix(cholA),bStar)
  ID <- rep(1:6, each=rep)
```

```
  e0 <- rnorm(n, sd=sqrt(varE))
  y <- b[ID]+e0
  obs <- data.frame(y=y,IDgen=ID,IDenv=ID) ## two copies of ID for readability of GLMM results
  ## fits
  fitme(y ~ 1+ corrMatrix(1|IDgen) , corrMatrix=A,data=obs,method="REML")
  obs$y01 <- ifelse(y<1.3,0,1)
  fitme(y01 ~ 1+ corrMatrix(1|IDgen)+(1|IDenv), corrMatrix=A,data=obs,
        family=binomial(), method="REML")
  prec_mat <- solve(A)
  colnames(prec_mat) <- rownames(prec_mat) <- rownames(A) # important
  fitme(y01 ~ 1+ corrMatrix(1|IDgen)+(1|IDenv) , covStruct=list(precision=prec_mat),
        data=obs, family=binomial(), method="REML")
}

## End(Not run)
```

---

phiHGLM *Fitting random effects in the residual dispersion model*

Description

ϕ parameters are estimated by fitting a Gamma HGLM to response values computed by the parent fitting function (e.g., by `HLfit` in the Examples). The `fitme` function is used to perform this fit. The `resid.model` of the parent call is used to control the arguments of this `fitme` call.

Usage

```
# 'resid.model' argument of main fitting functions
```

Arguments

`resid.model` is **either** a formula (without left-hand side) for the dispersion parameter `phi` of the residual error (a log link is assumed); **or** a list of arguments similar to those of a standard fit. The following arguments may be useful:

`formula` model formula as in formula-only case, without left-hand side. Random effects can be included and this appears to work well in simple cases (block effects, or geostatistical models) but has not been tested, or hardly so, for other cases.

`family` The family is always Gamma. The default link is log. The identity link can be tried but may fail because only the log link ensures that the fitted ϕ is positive.

`fixed` fixed values of parameters. Same usage as documented in `fitme`

`control.dist` A list of arguments that control the computation of the distance argument of the correlation functions. Same usage as documented in `HLCor`

`rand.family` A `family` object or a `list` of family objects describing the distribution of the random effect(s). Same usage as documented for `HLfit`

`init`, `lower`, `upper`, `control`
with same usage as documented in `fitme`, may be at least partly heeded. Other arguments should be ignored (see Details).

Details

The following elements in `resid.model` should be ignored:

method which is constrained to be identical to the method from the parent call;

control.HLfit, control.glm constrained to be identical to the same-named controls from the parent call;

resid.model constrained: no `resid.model` for a `resid.model`;

REMLformula constrained to NULL;

data The data of the parent call are used, so they must include all the variables required for the `resid.model`;

prior.weights constrained: no prior weights;

verbose constrained: will display a progress line summarizing the results of the `resid.model` fit at each iteration of main loop of the parent call.

init.HLfit Ignored. Users would have hard time guessing good initial values, which would be have to be ignored in most contexts anyway.

References

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006) Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

Examples

```
data("crack") # crack data, Lee et al. 2006 chapter 11 etc
hlfit <- HLfit(y~crack0+(1|specimen), family=Gamma(log),
               data=crack, rand.family=inverse.Gamma(log),
               resid.model=list(formula=~cycle+(1|specimen)) )
```

---

plot.HLfit *Model checking plots for mixed models*

Description

This function provides diagnostic plots for residual errors from the mean model and for random effects. Plots for the mean models are similar to those for GLMs, as described in Lee et al. 2006. Plots for residual errors consider the *standardized* deviance residuals (Lee et al. 2006, p.52), and plots for random effects likewise consider standardized values, i.e. each random deviate divided by $\sqrt{(1-q)}$ where q is the corresponding leverage for λ.

Usage

```
## S3 method for class 'HLfit'
plot(x, which = c("mean", "ranef"),
      titles = list(
          meanmodel=list(outer="Mean model",devres="Deviance residuals",
                    absdevres="|Deviance residuals|", resq="Residual quantiles",
                         devreshist="Deviance residuals"),
       ranef=list(outer="Random effects and leverages",qq="Random effects Q-Q plot",
                     levphi=expression(paste("Leverages for ",phi)),
          levlambda=expression(paste("Leverages for ",lambda)))
        ),
      control= list(), ask=TRUE, ...)
```

Arguments

x	An object of class HLfit, as returned by the fitting functions in spaMM.
which	A vector of keywords for different types of plots. By default, two types of plots are presented on different devices: diagnostic plots for mean values, and diagnostic plots for random effects. Either one can be selected using this argument. Use keyword "predict" for a plot of predicted response against actual response.
titles	A list of the main (inner and outer) titles of the plots. See the default value for the format.
control	A list of default options for the plots. Defaults are pch="+" and pcol="blue" for points, and lcol="red" for curves.
ask	Logical; passed to devAskNewPage which is run when a new device is opened by code.HLfit.
...	Options passed from plot.HLfit to par.

Details

The standardized deviance residuals are defined as the deviance residuals divided by $\phi\sqrt(1-q)$, where the deviance residuals are defined as for a GLM, and the leverages considered here are those given by hatvalues(.,type="std") (see hatvalues for details).

In principle the deviance residuals for the mean model should have a nearly Gaussian distribution hence form a nearly straight line on a Q-Q plot. However this is (trivially) not so for well-specified (nearly-)binary response data nor even for well-specified Poisson response data with moderate expectations. Hence this plot is not so useful. The DHARMa package proposes better-behaved diagnostic plots (but the p-value that appears on one of these plots may not stand for a valid goodness-of-fit test). The current version of DHARMa should handle spaMM fit objects; otherwise, see https://github.com/florianhartig/DHARMa/issues/95 for how to run DHARMa procedures on spaMM output.

Value

Returns the input object invisibly.

References

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006). Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

Examples

```
data("blackcap")
fit <- fitme(migStatus ~ 1+ Matern(1|latitude+longitude),data=blackcap,
             fixed=list(lambda=1,nu=1,rho=1))
plot(fit)
```

---

plot_effects *Partial-dependence effects and plots*

Description

The following functions evaluate or plot *partial-dependence* effects. The is a dedicated package for such plots, pdp (https://cran.r-project.org/package=pdp), so if you are not happy with these functions (which, for instance, do not handle pairs of variables and their interactions), try that package (which seems to handle fit object produced by spaMM).

pdep_effects evaluates the effect of a given fixed-effect variable, as (by default, the average of) predicted values on the response scale, over the empirical distribution of all other fixed-effect variables in the data, and of inferred random effects. This can be seen as the result of an experiment where specific treatments (given values of the focal variable) are applied over all conditions defined by the other fixed effects and by the inferred random effects. Thus, apparent dependencies induced by associations between predictor variables are avoided (see Friedman, 2001, from which the name "partial dependence plot" is taken; or Hastie et al., 2009, Section 10.13.2). This also avoids biases of possible alternative ways of plotting effects. In particular, such biases occur if the response link is not identity, and if averaging is performed on the linear-predictor scale or when other variables are set to some conventional value other than its average.

pdep_effects also compute intervals of the type defined by its intervals argument (by default, prediction intervals). By default, it returns a data frame of average values of point predictions and interval bounds for each value of the focal variable, but it can also return lists of all predictions.

plot_effects calls pdep_effects and produces a simple plot (using only base graphic functions) of its results, including prediction bands representing the two average one-sided widths of intervals. If added to the plot, the raw data may appear to depart from the partial-dependence predictions, since the data are a priori affected by the associations between variables which the predictions free themselves from.

Usage

```
pdep_effects(object, focal_var, newdata = object$data, length.out = 20,
             levels = NULL, intervals = "predVar", indiv = FALSE, ...)
plot_effects(object, focal_var, newdata = object$data, effects = NULL,
             xlab = focal_var, ylab = NULL, rgb.args = col2rgb("blue"),
             add = FALSE,  ylim=NULL, ...)
```

Arguments

- `object` — An object of class `HLfit`, as returned by the fitting functions in `spaMM`.
- `focal_var` — Character string: the name of the predictor variable whose effect is to be represented
- `newdata` — If non-NULL, a data frame passed to `predict.HLfit`, whose documentation should be consulted for further details.
- `effects` — If non-NULL, a data frame to substitute to the one produced by default by `pdep_effects`.
- `xlab` — If non-NULL, a character string: X-axis label for the plot.
- `ylab` — If non-NULL, a character string: Y-axis label for the plot.
- `ylim` — The `plot`'s `ylim` argument. Default is based on the (0.025,0.975) quantiles of the response.
- `rgb.args` — Color control arguments, in the format produced by `col2rgb`.
- `add` — Boolean: whether to add graphic elements of a previous plot produced by `plot_effects`
- `length.out` — Numeric: for a numeric predictor variable, the number of values at which predictions are evaluated.
- `levels` — If non-NULL, a character vector: for a factor predictor variable, the levels for which which predictions are evaluated.
- `intervals` — Passed to `predict.HLfit`, whose documentation should be consulted for further details.
- `indiv` — Boolean: whether to return all predictions given the values of other predictors in the `newdata`, or only their means.
- `...` — Further arguments passed by `plot_effects` to `pdep_effects`, or by `pdep_effects` to `predict.HLfit`.

Value

For `pdep_effects`, a nested list, or a data frame storing values of the `focal_var`, average point predictions `pointp` and bounds `low` and `up` of intervals, depending on the `indiv` argument. When `indiv` is `TRUE`, each sublist contains vectors for `pointp`, `low` and `up`.

For `plot_effects`, the same value, returned invisibly.

References

J.H. Friedman (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics 29(5):1189-1232.

J. Friedman, T. Hastie and R. Tibshirani (2009) The Elements of Statistical Learning, 2nd ed. Springer.

Examples

```
data("scotlip")
hlcor <- HLCor(cases~I(prop.ag/10) +adjacency(1|gridcode)+offset(log(expec)),
           adjMatrix=Nmatrix,family=poisson(),data=scotlip)
plot_effects(hlcor,focal_var="prop.ag",ylim=c(0,max(scotlip$cases)))
points(cases~prop.ag, data=scotlip, col="blue",pch=20)
```

Poisson — *Family function for GLMs and mixed models with Poisson and zero-truncated Poisson response.*

Description

`Poisson` (with a capital P) is a `family` that specifies the information required to fit a Poisson generalized linear model. Differs from the base version `stats::poisson` only in that it handles the zero-truncated variant, which can be specified either as `Tpoisson(<link>)` or as `Poisson(<link>,trunc = 0L)`. The truncated poisson with mean μ_T is defined from the un-truncated poisson with mean μ_U, by restricting its response strictly positive value. $\mu_T = \mu_U/(1-p0)$, where $p0 := \exp(-\mu_U)$ is the probability that the response is 0.

Usage

```
Poisson(link = "log", trunc = -1L)
Tpoisson(link="log")
# <Poisson object>$linkfun(mu, mu_truncated = FALSE)
# <Poisson object>$linkinv(eta, mu_truncated = FALSE)
```

Arguments

`link`	log, sqrt or identity link, specified by any of the available ways for GLM links (name, character string, one-element character vector, or object of class `link-glm` as returned by `make.link`).
`trunc`	Either `0L` for zero-truncated distribution, or `-1L` for default untruncated dsitribution.
`eta,mu`	Numeric (scalar or array). The linear predictor; and the expectation of response, truncated or not depending on `mu_truncated` argument.
`mu_truncated`	Boolean. For `linkinv`, whether to return the expectation of truncated (μ_T) or un-truncated (μ_U) response. For `linkfun`, whether the `mu` argument is μ_T, or is μ_U but has μ_T as attribute (μ_U without the attribute is not sufficient).

Details

The `mu.eta` member function is that of the base `poisson` family, hence ignores truncation.

`predict`, when applied on an object with a truncated-response family, by default returns μ_T. The simplest way to predict μ_U is to get the linear predictor value by `predict(.,type="link")`, and deduce μ_U using `linkinv(.)` (with default argument `mu_truncated=FALSE`), since getting μ_U from μ_T is comparatively less straightforward.

Value

A family object.

References

McCullagh, P. and Nelder, J.A. (1989) Generalized Linear Models, 2nd edition. London: Chapman & Hall.

Examples

```
data("scotlip")
logLik(glm(I(1+cases)~1,family=Tpoisson(),data=scotlip))
logLik(fitme(I(1+cases)~1+(1|id),family=Tpoisson(),fixed=list(lambda=1e-8),data=scotlip))
```

---

post-fit *Applying post-fit procedures on spaMM results*

Description

Packages implementing post-fit procedures define helper functions that may handle a limited range of classes of fit results. This documentation topic gives further directions to apply some post-fit procedures that handle objects of class `HLfit`. For other procedures, `spaMM` results may or may not be handled correctly by default. In such cases, diagnosing a failure in a debugging session may suggest a simple solution (such as providing the `coef.` argument to multcomp::glht).

Details

For multiple comparison procedures by multcomp::glht, one has to explicitly give the argument `coef.=fixef.HLfit` (see Examples; `fixef.HLfit` is the spaMM method for the generic function `fixef`);

For `DHARMa` plots, see `plot.HLfit`;

For using `RLRsim::RLRTSim()`, see `get_RLRTSim_args`.

Examples

```
if (requireNamespace("multcomp", quietly = TRUE)) {
  library(multcomp)
  set.seed(123)
  irisr <- cbind(iris,id=sample(4,replace=TRUE,size=nrow(iris)))
  irisfit <- fitme(Petal.Length~ Species +(1|id), data=irisr, family=Gamma(log))
  summary(glht(irisfit,mcp("Species" = "Tukey"), coef.=fixef.HLfit))
}
```

predict *Prediction from a model fit.*

Description

Prediction of the response variable by its expected value obtained as (the inverse link transformation of) the linear predictor (η) and more generally for terms of the form **X**_nβ+**Z**_n**Lv**, for new design matrices **X**_n and **Z**_n. Various components of prediction variances and predictions intervals can also be computed using `predict`. The `get_`... functions are convenient extractors for such components. `get_predCov_var_fix` extracts a block of a prediction covariance matrix. It was conceived for the specific purpose of computing the spatial prediction covariances between two "new" sets of geographic locations, without computing the full covariance matrix for both the new locations and the original (fitted) locations. When one of the two sets of new locations is fixed while the other varies, some expensive computations can be performed once for all sets of new locations, and be provided as the `fix_X_ZAC.object` argument. `preprocess_fix_corr` is designed to provide this argument.

Usage

```
## S3 method for class 'HLfit'
predict(object, newdata = newX, newX = NULL, re.form = NULL,
                        variances=list(), binding = FALSE, intervals = NULL,
                        level = 0.95, blockSize = 2000L, type = "response",
                        verbose=c(showpbar=eval(spaMM.getOption("barstyle"))),
                        control=list(), ...)
get_predCov_var_fix(object, newdata = NULL, fix_X_ZAC.object, fixdata, re.form = NULL,
                    variances=list(disp=TRUE,residVar=FALSE,cov=FALSE),
                    control=list(),  ...)
preprocess_fix_corr(object, fixdata, re.form = NULL,
                   variances=list(residVar=FALSE, cov=FALSE), control=list())
get_fixefVar(...)
get_predVar(..., variances=list(), which="predVar")
get_residVar(...)
get_respVar(...)
get_intervals(..., intervals="predVar")
```

Arguments

`object` The return object of fitting functions `HLfit`,`corrHLfit`,`HLCor`... returning an object inheriting from `HLfit` class.

`newdata` **Either** NULL, a matrix or data frame, or a numeric vector.

If `NULL`, the original data are reused. Otherwise, all variables required to evaluate model formulas must be included. Which variables are required may depend on other arguments: see "prediction with given phi's" example, also illustrating the syntax when formulas include an offset.

or a numeric vector, which names (if any) are ignored. This makes it easier to use `predict` as an objective function for an optimization procedure such as `optim`, which calls the objective function on unnamed vectors. However, one must make sure that the order of elements in the vector is the order of first occurrence of the variables in the model formula. This order can be checked in the error message returned when calling `predict` on a `newX` vector of clearly wrong size, e.g. `predict(<object>,newdata=numeric(0))`.

`newX` equivalent to newdata, available for back-compatibility

`re.form` formula for random effects to include. By default, it is NULL, in which case all random effects are included. If it is NA, no random effect is included. If it is a formula, only the random effects it contains are retained. The other variance components are removed from both point prediction and `variances` calculations. If you want to retain only the spatial effects in the point prediction, but all variances, either use re.form and add missing variances (on linear predictor scale) manually, or ignore this argument and see Details and Examples for different ways of controlling variances.

`variances` A list which elements control the computation of different estimated variances. In particular, `list(linPred=TRUE,disp=TRUE)` is suitable for uncertainty in point prediction.

`predict` can return four components of prediction variance: `fixefVar`, `predVar`, `residVar` and `respVar`, detailed below. They are all returned as attributes of the point predictions. By default, each component is a vector of variances. However, if `variances$cov=TRUE`, a covariance matrix is returned when applicable (i.e. not for `"residVar"`).

`fixefVar` is the (co)variance of fixed effects ($\mathbf{X}\beta$) due to uncertainty in β. It is called by `variances$fixefVar=TRUE`.

`predVar` is the (co)variance of the linear predictor η. It should be noted that the phrase "prediction variance" is used inconsistently in the literature, often to denote the uncertainty in the response (therefore, including the residual variance). Here, this uncertainly is called the response variance, and prediction variance is used to denote the uncertainty in the linear predictor (e.g., Booth & Hobert, 1998). It accounts for uncertainty in fixed effects ($\mathbf{X}\beta$) and random effects ($\mathbf{ZLv}$) for given dispersion parameters (see Details), but it can also accounts for uncertainty in dispersion parameters (λ and ϕ) estimates if `variances$disp=TRUE`, for models in which the effect of uncertainty in dispersion parameters on the uncertainty of the linear predictor can be computed. This effect can be computed for a scalar residual variance (ϕ) and for several random effects with scalar variances (λ). `variances$predVar=TRUE` will return the sum of the two components, if available; otherwise it returns only the (co)variance for given λ and ϕ. The latter component can be requested by `variances$linPred=TRUE`.

`as_tcrossfac_list=TRUE` can be used to return a list of matrices X_i such that the `predVar`= $\sum_i X_i X_i'$. It thus provides a representation of the `predVar` that may be useful in particular when the `predVar` has large dimension, as the component X_is may require less memory (being possibly non-square or sparse).

`residVar` provides the residual variances (for Gaussian or Gamma responses). It is called by `variances$residVar=TRUE`.

`respVar` is the variance of the response (see Details). It is called by `variances$respVar=TRUE`.

Calling for one (co)variance implies that some of its components may be also returned.

intervals
: NULL or character string or vector of strings. Provides prediction intervals with nominal level `level`, deduced from the given prediction variance term, e.g. `intervals="predVar"`. Currently only intervals from `fixefVar` and `predVar` (and for LMMs `respVar` including the residual variance) may have a probabilistic meaning. Intervals returned in other cases are (currently) meaningless.

which
: any of "predVar","respVar","residVar", "fixefVar", "intervals", or "naive"

level
: Coverage of the intervals.

binding
: If `binding` is a character string, the predicted values are bound with the `newdata` and the result is returned as a data frame. The predicted values column name is the given `binding`, or a name based on it if the `newdata` already include a variable with this name. If `binding` is `FALSE`, The predicted values are returned as a one-column matrix and the data frame used for prediction is returned as an attribute (unless it was `NULL`). If `binding` is `NA`, a vector is returned, without the previous attributes.

fixdata
: A data frame describing reference data which covariances with variable `newdata` may be requested.

fix_X_ZAC.object
: The return value of calling `preprocess_fix_corr` (see trivial Example). This is a more efficient way of providing information about the `fixdata` for repeated calls to `get_predCov_var_fix` with variable `newdata`.

blockSize
: Mainly for development purposes. For original or new data with many rows, it may be more efficient to split these data in small blocks, and this gives the maximum number or rows of the blocks. However, this will be ignored if a prediction covariance matrix is requested.

type
: character string; The returned point prediction is on the response scale if `type="response"` (the default). It is on the linear predictor scale if `type="link"`.

control
: A list; a warning will direct you to relevant usage when needed.

verbose
: A vector of booleans; it single currently used element is "showpbar", which controls whether to show a progress bar in certain prediction variance computations.

...
: further arguments passed to or from other methods. For the `get_...` functions, they are passed to `predict`.

Details

If `newdata` is NULL, `predict` returns the fitted responses, including random effects, from the object. Otherwise it computes new predictions including random effects as far as possible. For spatial random effects it constructs a correlation matrix **C** between new locations and locations in the original fit. Then it infers the random effects in the new locations as $\mathbf{C}\ (\mathbf{L}')^{-1}\ \mathbf{v}$ (see spaMM for notation). For non-spatial random effects, it checks whether any group (i.e., level of a random effect) in the new data was represented in the original data, and it adds the inferred random effect for this group to the prediction for individuals in this group.

`fixefVar` is the (co)variance of $\mathbf{X}\beta$ (or $\mathbf{X}$_nβ), deduced from the asymptotic covariance matrix of β estimates.

`predVar` is the prediction (co)variance of η=$\mathbf{X}\beta$+$\mathbf{Z}\mathbf{v}$ (see HLfit Details for notation), or more generally of $\mathbf{X}$_nβ+$\mathbf{Z}$_n$\mathbf{L}\mathbf{v}$, by default computed for given dispersion parameters.

For levels of the random effects present in the original data, predVar computation

takes into account the joint uncertainty in estimation of β and prediction of **v**.

For new levels of the random effects, predVar computation additionally takes into account uncertainty in prediction of **v** for these new levels. For **prediction covariance** with a new **Z**_n, it matters whether a single or multiple new levels are used: see Examples.

If variances$disp is TRUE, prediction variance may also include a term accounting for uncertainty in ϕ and λ, computed following Booth and Hobert (1998, eq. 19). This computation ignores uncertainties in spatial correlation parameters.

respVar is the sum of predVar (pre- and post-multiplied by $\partial\mu/\partial\eta$ for models with non-identity link) and of residVar.

These variance calculations are approximate except for LMMs, and cannot be guaranteed to give accurate results.

In the **point prediction** of the linear predictor, the unconditional expected value of u is assigned to the realizations of u for unobserved levels of non-spatial random effects (it is zero in GLMMs but not for non-gaussian random effects), and the inferred value of u is assigned in all other cases. Corresponding values of v are then deduced. This computation yields the classical "BLUP" or empirical Bayes predictor in LMMs, but otherwise it may yield less well characterized predictors, where "unconditional" v may not be its expected value when the rand.family link is not identity.

Intervals computations use the relevant variance estimates plugged in a Gaussian approximation, except for the simple linear model where it uses Student's *t* distribution.

Value

See Details in Tpoisson for questions specific to truncated distributions.

For predict, a matrix or data frame (according to the binding argument), with optional attributes frame, intervals, predVar, fixefVar, residVar, and/or respVar, the last four holding one or more variance vector or covariance matrices. The further attribute fittedName contains the binding name, if any.

The get_... extractor functions call predict and extract from its result the attribute implied by the name of the extractor. By default, get_intervals will return prediction intervals using predVar. get_predVar with non-default which argument has the same effect as the get_... function whose name is implied by which.

References

Booth, J.G., Hobert, J.P. (1998) Standard errors of prediction in generalized linear mixed models. J. Am. Stat. Assoc. 93: 262-272.

See Also

get_cPredVar for a bootstrap-corrected version of get_predVar.

Examples

```
data("blackcap")
fitobject <- corrHLfit(migStatus ~ 1 + Matern(1|latitude+longitude),data=blackcap,
                       ranFix=list(nu=4,rho=0.4,phi=0.05))
```

```
predict(fitobject)
getDistMat(fitobject)

#### multiple controls of prediction variances
## (1) fit with an additional random effect
grouped <- cbind(blackcap,grp=c(rep(1,7),rep(2,7)))
fitobject <- corrHLfit(migStatus ~ 1 +  (1|grp) +Matern(1|latitude+longitude),
                       data=grouped,  ranFix=list(nu=4,rho=0.4,phi=0.05))

## (2) re.form usage to remove a random effect from point prediction and variances:
predict(fitobject,re.form= ~ 1 +  Matern(1|latitude+longitude))

## (3) comparison of covariance matrices for two types of new data
moregroups <- grouped[1:5,]
rownames(moregroups) <- paste0("newloc",1:5)
moregroups$grp <- rep(3,5) ## all new data belong to an unobserved third group
cov1 <- get_predVar(fitobject,newdata=moregroups,
                     variances=list(linPred=TRUE,cov=TRUE))
moregroups$grp <- 3:7 ## all new data belong to distinct unobserved groups
cov2 <- get_predVar(fitobject,newdata=moregroups,
                     variances=list(linPred=TRUE,cov=TRUE))
cov1-cov2 ## the expected off-diagonal covariance due to the common group in the first fit.

## Not run:
## Other extractors:
fix_X_ZAC.object <- preprocess_fix_corr(fitobject,fixdata=blackcap)
# ... for use in multiple calls to get_predCov_var_fix():
get_predCov_var_fix(fitobject,newdata=blackcap[14,],fix_X_ZAC.object=fix_X_ZAC.object)

## prediction with distinct given phi's in different locations:
varphi <- cbind(blackcap,logphi=runif(14))
vphifit <- corrHLfit(migStatus ~ 1 + Matern(1|latitude+longitude),
                     resid.model = list(formula=~0+offset(logphi)),
                     data=varphi,  ranFix=list(nu=4,rho=0.4))
# for respVar computation, one needs the resid.model formula to specify phi:
get_respVar(vphifit,newdata=data.frame(latitude=1,longitude=1,logphi=1))
# for predVar computation, phi is not needed
#     (and could have been specified through ranFix):
get_predVar(vphifit,newdata=data.frame(latitude=1,longitude=1))

## point predictions and variances with new X and Z
if(requireNamespace("rsae", quietly = TRUE)) {
  data("landsat", package = "rsae")
  fitobject <- fitme(HACorn ~ PixelsCorn + PixelsSoybeans + (1|CountyName),
                     data=landsat[-33,])
  newXandZ <- unique(data.frame(PixelsCorn=landsat$MeanPixelsCorn,
                                PixelsSoybeans=landsat$MeanPixelsSoybeans,
                                CountyName=landsat$CountyName))
  predict(fitobject,newdata=newXandZ,variances = list(predVar=TRUE))
  get_predVar(fitobject,newdata=newXandZ,variances = list(predVar=TRUE))
}
```

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

---

random-effects *Structure of random effects*

Description

The structure of random-effect models adjustable by spaMM can generally be described by the following steps.

First, independent and identically distributed (iid) random effects **u** are drawn from one of the following distributions: **Gaussian** with zero mean, unit variance, and identity link; **Beta**-distributed, where $u B(1/(2\lambda), 1/(2\lambda))$ with mean=1/2, and var= $\lambda/[4(1+\lambda)]$; and with logit link `v=logit(u)`; **Gamma**-distributed random effects, where u `Gamma(shape=`1+1/λ,**scale=**1/λ): see `Gamma` for allowed links and further details; and **Inverse-Gamma**-distributed random effects, where u inverse-Gamma(`shape=`1+1/λ,**rate=**1/λ): see `inverse.Gamma` for allowed links and further details.

Second, a transformation **v**= f(**u**) is applied (this defines **v** whose elements are still iid).

Third, correlated random effects are obtained as **Mv**, where the matrix **M** can describe spatials correlation between observed locations, block effects (or repeated observations in given locations), and correlations involving unobserved locations. In most cases **M** is determined from the model formula, but it can also be controlled by covStruct argument. **M** takes the form **ZL** or **ZAL**, where **Z** is determined from the model formula, the optional **A** factor is given by the optional `"AMatrices"` attribute of argument `covStruct` of `HLCor` (also handled by `fitme` and `corrHLfit`), and **L** can be determined from the model formula or from `covStruct`. In particular:

Z is typically an incidence matrix: its elements z_{ij} are 1 if the ith observation is affected by the jth element of `ALb`, and zero otherwise.

For spatial random effects, **L** is typically the Cholesky “square root” of a correlation matrix determined by the random effect specification (e.g., `Matern(...)`), or given by the `covStruct` argument. This may be meaningful only for Gaussian random effects. Coefficients for each level of a random-coefficient model can also be represented as **Lv** where **L** is the “square root” of a correlation matrix.

If there is one response value par location, **L** for a spatial random effect is thus a square matrix whose dimension is the number of observations. Alternatively, several observations may be taken in the same location, and a matrix **Z** (automatically constructed) tells which element of **Lv** affects each observation. The linear predictor then contains a term of the form **ZLv**, where `dim(Z)` is (number of observations,number of locations).

in `IMRF` random effects (IMRF for Interpolated Markov Random Fields), the realized random effects in response locations are defined as linear combinations **ALv** of random effects **Lv** in distinct locations. In that case the dimension of **L** is the number of such distinct locations, an automatically constructed **A** matrix maps them to the observed locations, and **Z** again maps them to possibly repeated observations in observed locations.

`rankinfo` *Checking the rank of the fixed-effects design matrix*

Description

By default, fitting functions in `spaMM` check the rank of the design matrix for fixed effects, as `stats::lm` or `stats::glm` do (but not, say, `nlme::lme`). This computation can be quite long. To save time when fitting different models with the same fixed-effect terms to the same data, the result of the check can be extracted from a return object by `get_rankinfo()`, and can be provided as argument `control.HLfit$rankinfo` to another fit. Alternatively, the check will not be performed if `control.HLfit$rankinfo` is set to `NA`.

Usage

```
get_rankinfo(object)
```

Arguments

`object`	An object of class `HLfit`, as returned by the fitting functions in `spaMM`.

Details

The check is performed by a call to `qr()` methods for either dense or sparse matrices. If the design matrix is singular, a set of columns from the design matrix that define a non-singular matrix is identified. Note that different sets may be identified by sparse- and dense-matrix `qr` methods.

Value

A list with elements `rank`, `whichcols` (a set of columns that define a non-singular matrix), and `method` (identifying the algorithm used).

`salamander` *Salamander mating data*

Description

Data from a salamander mating experiment discussed by McCullagh and Nelder (1989, Ch. 14). Twenty males and twenty females from two populations (Rough Butt and Whiteside) were each paired with 6 individuals from their own or from the other population. The experiments were later published by Arnold et al. (1996).

Usage

```
data("salamander")
```

Format

The data frame includes 360 observations on the following variables:

Female Index of the female;

Male Index of the male;

Mate Whether the pair successfully mated or not;

TypeF Population of origin of female;

TypeM Population of origin of male;

Cross Interaction term between `TypeF` and `TypeM`;

Season A factor with levels `Summer` and `Fall`;

Experiment Index of experiment

Source

The data frame was borrowed from the `HGLMMM` package (Molas and Lesaffre, 2011), version 0.1.2.

References

Arnold, S.J., Verrell, P.A., and Tilley S.G. (1996) The evolution of asymmetry in sexual isolation: a model and a test case. Evolution 50, 1024-1033.

McCullagh, P. and Nelder, J.A. (1989). Generalized Linear Models, 2nd edition. London: Chapman & Hall.

Molas, M., Lesaffre, E. (2011) Hierarchical Generalized Linear Models: The R Package HGLMMM. Journal of Statistical Software 39, 1-20.

Examples

```
data("salamander")

## Not run:

HLfit(cbind(Mate,1-Mate)~TypeF+TypeM+TypeF*TypeM+(1|Female)+(1|Male),
      family=binomial(),data=salamander,method="ML")
# equivalent fo using fitme(), but here a bit faster

## End(Not run)
```

scotlip *Lip cancer in Scotland 1975 - 1980*

Description

This data set provides counts of lip cancer diagnoses made in Scottish districts from 1975 to 1980, and additional information relative to these data from Clayton and Kaldor (1987) and Breslow and Clayton (1993). The data set contains (for each district) counts of disease events and estimates of the fraction of the population involved in outdoor industry (agriculture, fishing, and forestry) which exposes it to sunlight.

`data("scotlip")` actually loads a data frame, `scotlip`, and an adjacency matrix, `Nmatrix`, between 56 Scottish districts, as given by Clayton and Kaldor (1987, Table 1).

Usage

```
data("scotlip")
```

Format

The data frame includes 56 observations on the following 7 variables:

gridcode alternative district identifier.

id numeric district identifier (1 to 56).

district district name.

cases number of lip cancer cases diagnosed 1975 - 1980.

population total person years at risk 1975 - 1980.

prop.ag percent of the population engaged in outdoor industry.

expec offsets considered by Breslow and Clayton (1993, Table 6, 'Exp' variable)

The rows are ordered according to `gridcode`, so that they match the rows of `Nmatrix`.

References

Clayton D, Kaldor J (1987). Empirical Bayes estimates of age-standardized relative risks for use in disease mapping. Biometrics, 43: 671 - 681.

Breslow, NE, Clayton, DG. (1993). Approximate Inference in Generalized Linear Mixed Models. Journal of the American Statistical Association: 88 9-25.

Examples

```
data("scotlip")
fitme(cases~I(log(expec)), data=scotlip, adjMatrix=Nmatrix, family=poisson)

## see 'help(autoregressive)' for additional examples involving 'scotlip'.
```

seaMask *Masks of seas or lands*

Description

These convenient masks can be added to maps of (parts of) the world to mask map information for these areas.

Usage

```
data("seaMask")
data("landMask")
# data("worldcountries") # deprecated and removed
# data("oceanmask") # deprecated and removed
```

Format

seaMask and landMask are data frames with two variables, x and y for longitude and latitude. Its contents are suitable for use with polypath: they define different polygones, each separated by a row of NAs.

worldcountries and oceanmask were sp::SpatialPolygonsDataFrame objects previously included in spaMM (see Details for replacement). Such objects were useful for creating land masks for different geographical projections.

Details

The removed objects worldcountries and oceanmask were suitable for plots involving geographical projections not available through map, and more generally for raster plots. A land mask could be produced out of worldcountries by filling the countries, as by fill="black" in the code for country.layer in the Examples in http://kimura.univ-montp2.fr/~rousset/spaMM/example_raster.html. These objects may now be available through the same web page, but a better place to look for the same functionality is the IsoriX package (objects CountryBorders and OceanMask).

seaMask and landMask were created from the world map in the maps package. polypath requires polygons, while map(interior=FALSE,plot=FALSE) returns small segments. landMask is the result of reconnecting the segments into full coastlines of all land blocks.

See Also

http://kimura.univ-montp2.fr/~rousset/spaMM/example_raster.html for access to, and use of worldcountries and oceanmask; https://cran.r-project.org/package=IsoriX for replacement CountryBorders and OceanMask for these objects.

Examples

```
## Predicting behaviour for a land bird: simplified fit for illustration
data("blackcap")
bfit <- fitme(migStatus ~ means+ Matern(1|longitude+latitude),data=blackcap,
```

```
            fixed=list(lambda=0.5537,phi=1.376e-05,rho=0.0544740,nu=0.6286311))

## the plot itself, with a sea mask,
## and an ad hoc 'pointmask' to see better the predictions on small islands
#
def_pointmask <- function(xy,r=1,npts=12) {
  theta <- 2*pi/npts *seq(npts)
  hexas <- lapply(seq(nrow(xy)), function(li){
    p <- as.numeric(xy[li,])
    hexa <- cbind(x=p[1]+r*cos(theta),y=p[2]+r*sin(theta))
    rbind(rep(NA,2),hexa) ## initial NA before each polygon
  })
  do.call(rbind,hexas)
}
ll <- blackcap[,c("longitude","latitude")]
pointmask <- def_pointmask(ll[c(2,4,5,6,7),],r=0.8) ## small islands only
#
data("seaMask")
#
filled.mapMM(bfit,add.map=TRUE,
             plot.title=title(main="Inferred migration propensity of blackcaps",
                              xlab="longitude",ylab="latitude"),
             decorations=quote(points(pred[,coordinates],cex=1,pch="+")),
             plot.axes=quote({axis(1);axis(2);
                        polypath(rbind(seaMask,pointmask),border=FALSE,
                                 col="grey", rule="evenodd")
             }))
```

---

seeds *Seed germination data*

Description

A classic toy data set, “from research conducted by microbiologist Dr P. Whitney of Surrey University. A batch of tiny seeds is brushed onto a plate covered with a certain extract at a given dilution. The numbers of germinated and ungerminated seeds are subsequently counted” (Crowder, 1978). Two seed types and two extracts are here considered in a 2x2 factorial design.

Usage

```
data("seeds")
```

Format

The data frame includes 21 observations on the following variables:

plate Factor for replication;

seed Seed type, a factor with two levels O73 and O75;

extract Root extract, a factor with two levels Bean and Cucumber;

r Number of seeds that germinated;

n Total number of seeds tested

Source

Crowder (1978), Table 3.

References

Crowder, M.J., 1978. Beta-binomial anova for proportions. Appl. Statist., 27, 34-37.

Y. Lee and J. A. Nelder. 1996. Hierarchical generalized linear models (with discussion). J. R. Statist. Soc. B, 58: 619-678.

Examples

```
data("seeds")
fitme(cbind(r,n-r)~seed*extract+(1|plate),family=binomial(),
      rand.family=Beta(),
      ## For an extended quasi-likelihood (EQL) fit as considered by Lee & Nelder (1996):
      # method="HL(0,0)",
      data=seeds)
```

---

simulate.HLfit *Simulate realizations of a fitted model.*

Description

From an HLfit object, `simulate.HLfit` function generates new samples given the estimated fixed effects and dispersion parameters. Simulation may be unconditional (the default, useful in many applications of parametric bootstrap), or conditional on the predicted values of random effects, or may draw from the conditional distribution of random effects given the observed response. Simulations may be run for the original values of fixed-effect predictor variables and of random effect levels (spatial locations for spatial random effects), or for new values of these.

Usage

```
## S3 method for class 'HLfit'
simulate(object, nsim = 1, seed = NULL, newdata = NULL,
                         type = "marginal", re.form, conditional = NULL,
                         verbose = c(type=TRUE,
                                   showpbar=eval(spaMM.getOption("barstyle"))),
                        sizes = NULL, resp_testfn = NULL, phi_type = "predict",
                    prior.weights = object$prior.weights, variances=list(), ...)
## S3 method for class 'HLfitlist'
simulate(object, nsim = 1, seed = NULL,
                             newdata = object[[1]]$data, sizes = NULL, ...)
```

Arguments

object	The return object of HLfit or similar function.
nsim	number of response vectors to simulate. Defaults to '1'.
seed	A seed for `set.seed`. If such a value is provided, the initial state of the random number generator at a global level is restored on exit from simulate.
newdata	A data frame closely matching the original data, except that response values are not needed. May provide new values of fixed predictor variables, new spatial locations, or new individuals within a block.
re.form	formula for random effects to condition on. Default behaviour depends on the `type` argument. The joint default is the latter's default, i.e., unconditional simulation. `re.form` is currently ignored when `type="Vlinpred"` (with a warning). Otherwise, `re.form=NULL` conditions on all random effects (as `type="residual"` does), and `re.form=NA` conditions on none of the random effects (as `type="marginal"` or `re.form=~0` do).
type	character string specifying the type of simulation for mixed models. `"marginal"` is for simulation from the marginal distribution of the random effect; `"residual"` accounts only for the residual variation of the fitted model; and the more speculative `"predVar"` accounts both for residual variation and for the uncertainty of the linear predictor (see Details).
conditional	Obsolete and will be deprecated. Boolean; TRUE and FALSE are equivalent to `type="residual"` and `type="marginal"`, respectively.
verbose	Either a single boolean (which determines `verbose[["type"]]`, or a vector of booleans with possible elements `"type"` (to display basic information about the type of simulation) and `"showpbar"` (see codepredict(.,verbose)).
sizes	A vector of sample sizes to simulate in the case of a binomial fit. Defaults to the sizes in the original data.
resp_testfn	NULL, or a function that tests a condition which simulated samples should satisfy. This function takes a response vector as argument and return a boolean (TRUE indicating that the sampel satisfies the condition).
phi_type	Character string, either `"predict"` or one of the values possible for `type`. This controls the residual variance parameter ϕ. The default is to use predicted ϕ values from the fit, which are the fitted ϕ values except when a structured-dispersion model is involved together with non-NULL `newdata`. However, when a structured-dispersion model is involved, it is also possible to simulate new ϕ values, and for a mixed-effects structured-dispersion model, the same types of simulation controlled by `type` for the main response can be performed as controlled by `phi_type`. For a fixed-effects structured-dispersion model, these types cannot be distinguished, and any `phi_type` distinct from `"predict"` will imply simulation under the fixed-effect model (see Examples).
prior.weights	Prior weights that may be substituted to those of the original fit, with the same effect on the residual variance.
variances	Used when `type="predVar"`: see Details.
...	further arguments passed to or from other methods; currently only passed to `predict` in a speculative bit of code (see Details).

Details

`type="predVar"` accounts for the uncertainty of the linear predictor, by drawing new values of the predictor in a multivariate gaussian distribution with mean and covariance matrix of prediction. In this case, the user has to provide a `variances` argument, passed to `predict`. The covariance matrix represent the joint uncertainty in the fixed-effect coefficients and of any random effects given the response and the point estimates of dispersion and correlation parameters. This distribution is known exactly in LMMs, and otherwise approximated as a Gaussian distribution with mean vector and covariance matrix given as per the Laplace approximation.

`type="(ranef|response)"` can be viewed as a special version of `type="predVar"` where `variances=list(linPred=TRUE` and only the uncertainty in the random effects is taken into account.

A full discussion of the merits of the different `types` is beyond the scope of this documentation, but these different types may not all be useful. `type="marginal"` is typically used for computation of confidence intervals by parametric boostrap methods. `type="residual"` is used by `get_cPredVar` for its evaluation of a bias term. The other `types` may be used to simulate the uncertainty in the random effects, conditionally on the data, and may thereofre be more akin to the computation of prediction intervals conditionnally on an (unknown but inferred) realization of the random effects. But these should presumably not be used in a bootstrap computation of such intervals, as this would represent a double accounting of the uncertainty that the boostrap aims to quantify.

Value

For the `HLfitlist` method (i.e., the result of a multinomial fit), a list of simulated responses. Otherwise, a vector (if nsim=1) or a matrix with nsim columns, each containing a simulated response.

Examples

```
data("Loaloa")
HLC <- HLCor(cbind(npos,ntot-npos)~Matern(1|longitude+latitude),
           data=Loaloa,family=binomial(),
           ranPars=list(lambda=1,nu=0.5,rho=1/0.7))
simulate(HLC,nsim=2)

## Structured dispersion model
data("wafers")
hl <- HLfit(y ~X1+X2+X1*X3+X2*X3+I(X2^2)+(1|batch),family=Gamma(log),
            resid.model = ~ X3+I(X3^2) ,data=wafers)
simulate(hl,type="marginal",phi_type="simulate",nsim=2)
```

`spaMM` *Inference in mixed models, in particular spatial GLMMs*

Description

Fits a range of mixed-effect models, including those with spatially correlated random effects. The random effects are either Gaussian (which defines GLMMs), or other distributions (which defines the wider class of hierarchical GLMs), or simply absent (which makes a LM or GLM). spaMM is designed to be used through the high-level fitting functions `fitme` (the most general function), `HLfit` (sometimes faster, for non-spatial models), HLCor (sometimes faster, for conditional-autoregressive models and fixed-correlation models), corrHLfit (now of lesser interest); and additional functions such as fixedLRT for likelihood-ratio testing, simulate and predict.

Both maximum likelihood (ML) and restricted likelihood (REML) can be used for linear mixed models, and extensions of these methods using Laplace approximations are used for non-Gaussian random response. Several variants of these methods discussed in the literature are included (see Details in HLfit), the most notable of which may be "PQL/L" for binary-response GLMMs (see Example for arabidopsis data). PQL methods implemented in spaMM are closer to (RE)ML methods than those implemented in `MASS::glmmPQL`.

Details

The standard response families `gaussian`, `binomial`, `poisson`, and `Gamma` are handled, as well as negative binomial (see negbin), zero-truncated poisson and negative binomial, and Conway-Maxwell-Poisson response (see Tpoisson, Tnegbin and COMPoisson). A `multi` family look-alike is also available for multinomial response, with some constraints.

The variance parameter of residual error is denoted ϕ (`phi`): this is the residual variance for gaussian response, but for Gamma-distributed response, the residual variance is $\phi\mu^2$ where μ is expected response. A fixed-effects linear predictor for ϕ, modeling heteroscedasticity, can be considered (see Examples).

The package fits models including several nested or crossed random effects, including autocorrelated ones with the following correlation models: Matern, Cauchy, interpolated Markov Random Fields (IMRF), first-order autoregressive (AR1), conditional autoregressive as specified by an adjacency matrix, or any fixed correlation matrix (corrMatrix). GLMMs and HGLMs are fit via Laplace approximations for (1) the marginal likelihood with respect to random effects and (2) the restricted likelihood (as in REML), i.e. the likelihood of random effect parameters given the fixed effect estimates.

All handled models can be formulated in terms of a linear predictor of the traditional form `offset+` $\mathbf{X}\beta$ + $\mathbf{Z}$ $\mathbf{b}$, where $\mathbf{X}$ is the design matrix of fixed effects, β (`beta`) is a vector of fixed-effect coefficients, $\mathbf{Z}$ is a "design matrix" for the random effects (which is instead denoted **M=ZAL** elsewhere in the package documentation), and **b** a vector of random effect values. The general structure of `Mb` is described in random-effects.

The syntax for formulas extends that used in the `lme4` package. In particular, **non-autocorrelated** random effects are specified using the `(1|<block>)` syntax, and *Gaussian* random-coefficient terms by `(<rhs>|<block>)`.The double-vertical syntax, `( rhs || lhs )`, is interpreted as in `lme4`. Any such term is immediately converted to `( ( 1 | lhs ) + ( 0 + lhs | rhs ) )`, and should be counted as two random effects for all purposes (e.g., for fixing the variances of the random effects). **Autocorrelated** random effects are specified by adding some prefix to this syntax, `<prefix>(1|.)`, e.g. `Matern(1|long+lat)`.

Since version 2.6.0, it is possible to fit some "some autocorrelated random-coefficient" models by a syntax consistent with that of random-coefficient terms, `<prefix>(<rhs>|.)`. For example, independent Matérn effects can be fitted for males and females by using the syntax `Matern(male|.) + Matern(female|.)`, where `male` and `female` are TRUE/FALSE factors. A numeric variable `z` can also be considered, in which case the proper syntax is `Matern(0+z|.)`, which represents an autocorrelated random-slope (only) term (or, equivalently, a direct specification of heteroscedasticy of the Matérn random effect). All these effects are achieved by direct control of the elements of the incidence matrix $\mathbf{Z}$ through the `<rhs>` term: for numeric `z`, such elements are multiplied by z values, and thus provide a variance of order O(z **squared**). By contrast, Matern(z|.) is not defined. It could mean that a correlation structure between random intercepts and random slopes is to be combined with a Matérn correlation structure, but no way of combining them is yet defined and implemented.

Since version 2.7.0, the syntax (z-1|.), for **numeric** z only, can also be used to fit **some heteroscedastic non-Gaussian** random effects. For example, a Gamma random-effect term (wei-1|block) specifies an heteroscedastic Gamma random effect u with constant mean 1 and variance wei^2 λ, where λ is still the estimated variance parameter. See Details of negbin for a possible application. Here, this effect is not implemented through direct control of **Z** (multiplying the elements of an incidence matrix **Z** by wei), as this would have a different effect on the distribution of the random effect term. (z|.) is not defined for *non-Gaussian* random effects. It could mean that a correlation structure between random intercepts and random slopes for (say) Gamma-distributed random effects is considered, but such correlation structures are not well-specified by their correlation matrix.

Author(s)

spaMM was initially published by François Rousset and Jean-Baptiste Ferdy, and is continually developed by F. Rousset and tested by Alexandre Courtiol.

References

Lee, Y., Nelder, J. A. and Pawitan, Y. (2006). Generalized linear models with random effects: unified analysis via h-likelihood. Chapman & Hall: London.

Rousset F., Ferdy, J.-B. (2014) Testing environmental and genetic effects in the presence of spatial autocorrelation. Ecography, 37: 781-790. http://dx.doi.org/10.1111/ecog.00566

See Also

The test directory of the package provides many additional examples of spaMM usage beyond those from the formal documentation.

Examples

```
data("wafers")
data("scotlip") ## loads 'scotlip' data frame, but also 'Nmatrix'

##     Linear model
fitme(y ~ X1, data=wafers)

##     GLM
fitme(y ~ X1, family=Gamma(log), data=wafers)
fitme(cases ~ I(log(population)), data=scotlip, family=poisson)

##     Non-spatial GLMMs
fitme(y ~ 1+(1|batch), family=Gamma(log), data=wafers)
fitme(cases ~ 1+(1|gridcode), data=scotlip, family=poisson)
#
# Random-slope model (mind the output!)
fitme(y~X1+(X2|batch),data=wafers, method="REML")
```

```
# Spatial, conditional-autoregressive GLMM
if (spaMM.getOption("example_maxtime")>2) {
  fitme(cases ~ I(log(population))+adjacency(1|gridcode), data=scotlip, family=poisson,
        adjMatrix=Nmatrix) # with adjacency matrix provided by data("scotlip")
}
# see ?adjacency for more details on these models

##     Spatial, geostatistical GLMM:
#see e.g. examples in ?blackcap, ?Loaloa, or ?arabidopsis

##     Hierachical GLMs with non-gaussian random effects
 data("salamander")
if (spaMM.getOption("example_maxtime")>1) {
 # both gaussian and non-gaussian random effects
 fitme(cbind(Mate,1-Mate)~1+(1|Female)+(1|Male),family=binomial(),
        rand.family=list(gaussian(),Beta(logit)),data=salamander)

 # Random effect of Male nested in that of Female:
 fitme(cbind(Mate,1-Mate)~1+(1|Female/Male),
       family=binomial(),rand.family=Beta(logit),data=salamander)
 # [ also allowed is cbind(Mate,1-Mate)~1+(1|Female)+(1|Male %in% Female) ]
}

##    Modelling residual variance ( = structured-dispersion models)
# GLM response, fixed effects for residual variance
fitme( y ~ 1,family=Gamma(log),
      resid.model = ~ X3+I(X3^2) ,data=wafers)
#
# GLMM response, and mixed effects for residual variance
if (spaMM.getOption("example_maxtime")>1.5) {
  fitme(y ~ 1+(1|batch),family=Gamma(log),
        resid.model = ~ 1+(1|batch) ,data=wafers)
}
```

---

spaMM-conventions *spaMM conventions and differences from related fitting procedures*

Description

input arguments are generally similar to those of glm and (g)lmer, in particular for the spaMM::fitme function, with the exception of the prior.weights argument, which is simply weights in the other packages. The name prior.weights seems more consistent, since e.g. glm returns its input weights as output prior.weights, while its output weights are instead the weights in the final iteration of an iteratively weighted least-square fit.

The \bolddefault likelihood target for dispersion parameters is restricted likelihood (REML estimation) for corrHLfit and (marginal) likelihood (ML estimation) for fitme. Model fits may provide restricted likelihood values(ReL) even if restricted likelihood is is not used as an objective function at any step in the analysis.

See good-practice for advice about the proper syntax of formula.

Computation times depend on control parameters given by spaMM.getOption("spaMM_tol") parameters (for iterative algorithms), and spaMM.getOption("nloptr") parameters for the default optimizer. Do not use spaMM.options() to control them globally, unless you know what you are doing. Rather control them locally by the control.HLfit argument to control spaMM_tol, and by the control arguments of corrHLfit and fitme to control nloptr. If nloptr$Xtol_rel is set above 5e-06, fitme will by default refit the fixed effects and dispersion parameters (but not other correlation parameters estimated by nloptr) by the iterative algorithm after nloptr convergence. Increasing nloptr$Xtol_rel value may therefore switches the bulk of computation time from the optimizer to the iterative algorithm, and may increase or decrease computation time depending on which algorithm is faster for a given input. Use control$refit if you wish to inhibit this, but note that by default it provides a rescue to a poor nloptr result due to a too large Xtol_rel.

References

Chambers J.M. (2008) Software for data analysis: Programming with R. Springer-Verlag New York

---

spaMM.colors *A flashy color palette.*

Description

spaMM.colors is the default color palette for some color plots in spaMM.

Usage

```
spaMM.colors(n = 64, redshift = 1, adjustcolor_args=NULL)
```

Arguments

n Number of color levels returned by the function. A calling graphic function with argument nlevels will typically take the first (i.e., bluest) nlevels color levels. If n<nlevels, the color levels are recycled

redshift The higher it is, the more the palette blushes....

adjustcolor_args Either NULL or a list of arguments for adjustcolor, in which case adjustcolor is called to modify spaMM.colors's default vector of colors. See the documentation of the latter function for further information. All arguments except col are possible.

Details

If you don't like this color palette, have a look at the various ones provided by the fields package.

Value

A vector giving the colors in a hexadecimal format.

Examples

```
## see mapMM examples
```

---

spaMM.filled.contour *Level (Contour) Plots with better aspect ratio control (for geographical maps, at least)*

Description

This function is derived from filled.contour in the graphics package, and this documentation is likewise heavily based on that of filled.contour.

This function likewise produces a contour plot with the areas between the contours filled in solid color, and a key showing how the colors map to z values is likewise shown to the right of the plot. The only difference is the way the aspect ratio is determined and can be controlled (using the map.asp parameter instead of asp), They thus easily provide nice-looking maps with meaningful latitude/longitude ratio (see Examples). However, this does not work well with rstudio.

Usage

```
spaMM.filled.contour(x = seq(0, 1, length.out = nrow(z)),
                y = seq(0, 1, length.out = ncol(z)),
                z,
                xrange = range(x, finite = TRUE),
                yrange = range(y, finite = TRUE),
                zrange = range(z, finite = TRUE),
                margin=1/20,
                levels = pretty(zrange, nlevels), nlevels = 20,
                color.palette = spaMM.colors,
                col = color.palette(length(levels) - 1),
                plot.title, plot.axes, key.title=NULL, key.axes=NULL,
                map.asp = NULL, xaxs = "i", yaxs = "i", las = 1,
                axes = TRUE, frame.plot = axes, ...)
```

Arguments

x, y locations of grid lines at which the values in z are measured. These must be in ascending order. (The rest of this description does not apply to .filled.contour.) By default, equally spaced values from 0 to 1 are used. If x is a list, its components x$x and x$y are used for x and y, respectively. If the list has component z this is used for z.

z a numeric matrix containing the values to be plotted.. Note that x can be used instead of z for convenience.

xrange x range of the plot.

yrange y range of the plot.

zrange	z range of the plot.
margin	This controls how far (in relative terms) the plot extends beyond the x and y ranges of the analyzed points, and is overriden by explicit xrange and yrange arguments.
levels	a set of levels which are used to partition the range of z. Must be **strictly** increasing (and finite). Areas with z values between consecutive levels are painted with the same color.
nlevels	if levels is not specified, the range of z, values is divided into approximately this many levels.
color.palette	a color palette function to be used to assign colors in the plot.
col	an explicit set of colors to be used in the plot. This argument overrides any palette function specification. There should be one less color than levels
plot.title	statements which add titles to the main plot.
plot.axes	statements which draw axes (and a box) on the main plot. This overrides the default axes.
key.title	statements which add titles for the plot key.
key.axes	statements which draw axes on the plot key. This overrides the default axis.
map.asp	the y/x aspect ratio of the 2D plot area (not of the full figure including the scale). Default is (plotted y range)/(plotted x range) (i.e., scales for x are identical).
xaxs	the x axis style. The default is to use internal labeling.
yaxs	the y axis style. The default is to use internal labeling.
las	the style of labeling to be used. The default is to use horizontal labeling.
axes, frame.plot	logicals indicating if axes and a box should be drawn, as in plot.default.
...	additional graphical parameters, currently only passed to title().

Details

The values to be plotted can contain NAs. Rectangles with two or more corner values are NA are omitted entirely: where there is a single NA value the triangle opposite the NA is omitted.

Values to be plotted can be infinite: the effect is similar to that described for NA values.

Note

Builds heavily on filled.contour by Ross Ihaka and R-core. spaMM.filled.contour uses the layout function and so is restricted to a full page display.

The output produced by spaMM.filled.contour is actually a combination of two plots; one is the filled contour and one is the legend. Two separate coordinate systems are set up for these two plots, but they are only used internally – once the function has returned these coordinate systems are lost. If you want to annotate the main contour plot, for example to add points, you can specify graphics commands in the plot.axes argument. See the Examples.

References

Cleveland, W. S. (1993) *Visualizing Data*. Summit, New Jersey: Hobart.

See Also

contour, image, palette; contourplot and levelplot from package lattice.

Examples

```
spaMM.filled.contour(volcano, color = spaMM.colors) # simple

## Comparing the layout with that of filled.contour:
#  (except that it does not always achieve the intended effect
#  in RStudio Plots pane).

x <- 10*1:nrow(volcano)
y <- 10*1:ncol(volcano)
spaMM.filled.contour(x, y, volcano, color = terrain.colors,
    plot.title = title(main = "The Topography of Maunga Whau",
    xlab = "Meters North", ylab = "Meters West"),
    plot.axes = { axis(1, seq(100, 800, by = 100))
                  axis(2, seq(100, 600, by = 100)) },
    key.title = title(main = "Height\n(meters)"),
    key.axes = axis(4, seq(90, 190, by = 10)))  # maybe also asp = 1
mtext(paste("spaMM.filled.contour(.) from", R.version.string),
      side = 1, line = 4, adj = 1, cex = .66)

## compare with

filled.contour(x, y, volcano, color = terrain.colors,
    plot.title = title(main = "The Topography of Maunga Whau",
    xlab = "Meters North", ylab = "Meters West"),
    plot.axes = { axis(1, seq(100, 800, by = 100))
                  axis(2, seq(100, 600, by = 100)) },
    key.title = title(main = "Height\n(meters)"),
    key.axes = axis(4, seq(90, 190, by = 10)))  # maybe also asp = 1
mtext(paste("filled.contour(.) from", R.version.string),
      side = 1, line = 4, adj = 1, cex = .66)
```

---

spaMM_boot *Parametric bootstrap*

Description

This simulates samples from a fit object inheriting from class "HLfit", as produced by spaMM's fitting functions, and applies a given function to each simulated sample. Parallelization is supported (see Details). A typical usage of the parametric bootstrap is to fit by one model some samples simulated under another model (see Example).

Usage

```
spaMM_boot(object, simuland, nsim, nb_cores=NULL, seed=NULL,
           resp_testfn=NULL, control.foreach=list(),
           debug. = FALSE, type, fit_env=NULL, cluster_args=NULL,
           showpbar= eval(spaMM.getOption("barstyle")),
           ...)
```

Arguments

object	The fit object to simulate from.
simuland	The function to apply to each simulated sample. See Details for requirements of this function.
nsim	Number of samples to simulate and analyze.
nb_cores	Number of cores to use for parallel computation. The default is spaMM.getOption("nb_cores"), and 1 if the latter is NULL. nb_cores=1 prevents the use of parallelisation procedures.
seed	Passed to simulate.HLfit
resp_testfn	Passed to simulate.HLfit; NULL, or a function that tests a condition which simulated samples should satisfy. This function takes a response vector as argument and return a boolean (TRUE indicating that the sample satisfies the condition).
control.foreach	list of control arguments for foreach. These include in particular .combine (with default value "rbind"), and .errorhandling (with default value "remove", but "pass" is quite useful for debugging).
debug.	Boolean (or integer, interpreted as boolean). For debugging purposes, particularly from parallel computations. The effect of debug.=TRUE depends on what simuland does of it. The default simuland for likelihood ratio testing functions, eval_replicate, shows how debug. can be used.
type	Character: passed to simulate.HLfit. Defaults, with a warning, to code-type="marginal" in order to replicate the behaviour of previous versions. But this is not necessarily the appropriate type for all possible uses. See Details of simulate.HLfit for other implemented options.
fit_env	An environment or list containing variables necessary to evaluate simuland on each sample, and not included in the fit object. E.g., use fit_env=list(phi_fix=phi_fix) if the fit assumed fixed=list(phi=phi_fix)
cluster_args	NULL or a list of arguments, passed to makeCluster.
showpbar	Controls display of progress bar. See barstyle option for details.
...	Further arguments passed to the simuland function.

Details

spaMM_boot handles parallel backends with different features. pbapply::pbapply has a very simple interface (essentially equivalent to apply) and provides progress bars, but (in version 1.4.0, at least) does not have efficient load-balancing. doSNOW also provides a progress bar and allows more efficient load-balancing, but its requires foreach. foreach handles errors diferently from pbapply (which will simply stop if fitting a model to a bootstrap replicate fails): see the foreach documentation.

spaMM_boot calls simulate.HLfit on the fit object and applies simuland on each column of the matrix returned by this call. simulate.HLfit uses the type argument, which must be explicitly provided.

The simuland function must take as first argument a vector of response values, and may have other arguments including '...'. When required, these additional arguments must be passed through the '...' arguments of spaMM_boot. Variables needed to evaluate them must be available from within the simuland function or otherwise provided as elements of fit_env.

Value

A list, with two elements (unless debug. is TRUE):

bootreps nsim return values in the format returned either by apply or parallel::parApply or by foreach::`%dopar%` as controlled by control.foreach$.combine (which is here "rbind" by default).

RNGstate the state of .Random.seed at the beginning of the sample simulation.

Examples

```
if (spaMM.getOption("example_maxtime")>7) {
 data("blackcap")

 # Generate fits of null and full models:
 lrt <- fixedLRT(null.formula=migStatus ~ 1 + Matern(1|latitude+longitude),
                 formula=migStatus ~ means + Matern(1|latitude+longitude),
                 method='ML',data=blackcap)

 # The 'simuland' argument:
 myfun <- function(y, what=NULL, lrt, ...) {
    data <- lrt$fullfit$data
   data$migStatus <- y ## replaces original response (! more complicated for binomial fits)
    full_call <- getCall(lrt$fullfit) ## call for full fit
    full_call$data <- data
    res <- eval(full_call) ## fits the full model on the simulated response
    if (!is.null(what)) res <- eval(what) ## post-process the fit
    return(res) ## the fit, or anything produced by evaluating 'what'
  }
  # where the 'what' argument (not required) of myfun() allows one to control
  # what the function returns without redefining the function.

  # Call myfun() with no 'what' argument: returns a list of fits
 spaMM_boot(lrt$nullfit, simuland = myfun, nsim=1, lrt=lrt, type ="marginal")[["bootreps"]]

  # Return only a model coefficient for each fit:
  spaMM_boot(lrt$nullfit, simuland = myfun, nsim=7,
             what=quote(fixef(res)[2L]), lrt=lrt, type ="marginal")[["bootreps"]]
}
```

spaMM_glm.fit — *Fitting generalized linear models without initial-value or divergence headaches*

Description

`spaMM_glm.fit` is a stand-in replacement for `glm.fit`, which can be called through `glm` by using `glm(<>,method="spaMM_glm.fit")`. Input and output structure are exactly as for `glm.fit`. It uses a Levenberg-Marquardt algorithm to prevent divergence of estimates. If the rcdd package is installed, the function can automatically find valid starting values or else indicate that no parameter value is feasible. `spaMM_glm` is a convenient wrapper, calling `glm` with default method `glm.fit`, then calling method `spaMM_glm.fit`, with possibly different initial values, if `glm.fit` failed.

Usage

```
spaMM_glm.fit(x, y, weights = rep(1, nobs), start = NULL, etastart = NULL,
              mustart = NULL, offset = rep(0, nobs), family = gaussian(),
              control = list(maxit=200), intercept = TRUE, singular.ok = TRUE)
spaMM_glm(formula, family = gaussian, data, weights, subset,
          na.action, start = NULL, etastart, mustart, offset,
        control = list(...), model = TRUE, method = c("glm.fit","spaMM_glm.fit"),
       x = FALSE, y = TRUE, singular.ok = TRUE, contrasts = NULL, strict=FALSE, ...)
```

Arguments

	All arguments except `strict` are common to these functions and their `stats` package equivalents, `glm` and `glm.fit`. Most arguments operate as for the latter functions, whose documentation is repeated below. The `control` argument may operate differently.
`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given in the 'Details' section of `glm`.
`family`	a description of the error distribution and link function to be used in the model. For `spaMM_glm` this can be a character string naming a family function, a family function or the result of a call to a family function. For `spaMM_glm.fit` only the third option is supported. (See `family` for details of family functions.)
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `glm` is called.
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.

na.action	a function which indicates what should happen when the data contain NAs. The default is set by the na.action setting of options, and is na.fail if that is unset. The 'factory-fresh' default is na.omit. Another possible value is NULL, no action. Value na.exclude can be useful.
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	starting values for the vector of means.
offset	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be NULL or a numeric vector of length equal to the number of cases. One or more offset terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See model.offset.
control	a list of parameters for controlling the fitting process. This is passed to glm.control, as for glm.fit. Because one can assume that spaMM_glm.fit will converge in many cases where glm.fit does not, spaMM_glm.fit allows more iterations (200) by default. However, if spaMM_glm.fit is called through glm(. . .,method="spaMM_glm.fit"), then the number of iterations is controlled by the glm.control call within glm, so that it is 25 by default, overriding the spaMM_glm.fit default.
model	a logical value indicating whether *model frame* should be included as a component of the returned value.
method	A 2-elements vector specifying first the method to be used by spaMM_glm in the firt attempt at fitting the model, second the method to be used in a second attempt if the first failed. Possible methods include those shown in the default, "model.frame", which returns the model frame and does no fitting, or user-supplied fitting functions. These functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as glm.fit.
x, y	For spaMM_glm: x is a design matrix of dimension n * p, and y is a vector of observations of length n. For spaMM_glm.fit: x is a design matrix of dimension n * p, and y is a vector of observations of length n.
singular.ok	logical; if FALSE a singular fit is an error.
contrasts	an optional list. See the contrasts.arg of model.matrix.default.
intercept	logical. Should an intercept be included in the *null* model?
strict	logical. Whether to perform a fit by spaMM_glm.fit if glm.fit returned the warning "glm.fit: algorithm did not converge".
...	arguments to be used to form the default control argument if it is not supplied directly.

Value

An object inheriting from class glm. See glm for details.

Note

The source and documentation is derived in large part from those of `glm.fit`.

Examples

```
x <- c(8.752,20.27,24.71,32.88,27.27,19.09)
y <- c(5254,35.92,84.14,641.8,1.21,47.2)

# glm(.) fails:
(check_error <- try(glm(y~ x,data=data.frame(x,y),family=Gamma(log)), silent=TRUE))
if ( ! inherits(check_error,"try-error")) stop("glm(.) call unexpectedly succeeded")

spaMM_glm(y~ x,data=data.frame(x,y),family=Gamma(log))

## Gamma(inverse) examples
x <- c(43.6,46.5,21.7,18.6,17.3,16.7)
y <- c(2420,708,39.6,16.7,46.7,10.8)

# glm(.) fails (can't find starting value)
(check_error <- suppressWarnings(try(glm(y~ x,data=data.frame(x,y),family=Gamma()) , silent=TRUE)))
if ( ! inherits(check_error,"try-error")) stop("glm(.) call unexpectedly succeeded.")

if (requireNamespace("rcdd",quietly=TRUE)) {
  spaMM_glm(y~ x,data=data.frame(x,y),family=Gamma())
}
```

`sparse_precision` *Sparse_precision algorithm*

Description

spaMM includes fitting algorithms formulated in terms of the correlation matrix of random effects, and algorithms formulated in terms of the precision matrix (i.e. inverse covariance matrix) for random effects. Formulation of random effects in terms of their precision or of their correlation matrices are meaningful for Gaussian random effects, but beyond this both classes of algorithms work when the model include random effects with non-Gaussian distribution and no intrinsic correlation structure, and therefore for the full class of HGLMs.

The algorithms based on precision matrices may be more efficient when the precision matrix is sparse but the correlation matrix is dense. However, spaMM does not yet select the fastest algorithm by default, and the default choice has changed over versions without being properly documented here. A non-default choice of fitting algorithm can be selected by using `spaMM.options(sparse_precision= <TRUE|FALSE>)`. Currently it is selected by default in two cases (with exceptions indicated by specific messages): (1) for `IMRF` random effects, but not for other conditional autoregressive models (with a random effect of the form `adjacency(1|<grouping factor>)`); and (2) when the `covStruct` syntax is used to provide a fixed precision matrix (see `pedigree` for an example).

stripHLfit *Reduce the size of fitted objects*

Description

Large matrices and other memory-expensive objects may be stored in a fit object. This function removes them in order to reduce the size of the object, particularly when stored on disk. In principle, the removed objects can be regenerated automatically when needed (e.g., for a predict()).

Usage

```
stripHLfit(object, ...)
```

Arguments

`object`	The result of a fit (an object of class `HLfit`).
`...`	Further arguments, not currently used.

Value

The input fit objects with some elements removed.

Note

The effect may change without notice between versions as the efficiency of the operation is highly sensitive to implementation details.

Examples

```
## Not run:
## rather unconvincing example : quantitative effect is small.

# measure size of saved object:
saveSize <- function (object,...) {
    tf <- tempfile(fileext = ".RData")
    on.exit(unlink(tf))
    save(object, file = tf,...)
    file.size(tf)
  }
data("Loaloa")
lfit <- fitme(cbind(npos,ntot-npos)~elev1+elev2+elev3+elev4+maxNDVI1+seNDVI
                   +Matern(1|longitude+latitude), method="HL(0,1)",
              data=Loaloa, family=binomial(), fixed=list(nu=0.5,rho=1,lambda=0.5))
saveSize(lfit)
pfit <- predict(lfit,newdata=Loaloa,variances=list(cov=TRUE)) # increases size!
saveSize(lfit)
lfit <- stripHLfit(lfit)
saveSize(lfit)

## End(Not run)
```

summary.HLfit *Summary and print methods for fit and test results.*

Description

Summary and print methods for results from HLfit or related functions. summary may also be used as an extractor (see e.g. beta_table).

Usage

```
## S3 method for class 'HLfit'
summary(object, details=FALSE, max.print=100L, verbose=TRUE, ...)
## S3 method for class 'HLfitlist'
summary(object, ...)
## S3 method for class 'fixedLRT'
summary(object, verbose=TRUE, ...)
## S3 method for class 'HLfit'
print(x,...)
## S3 method for class 'HLfitlist'
print(x,...)
## S3 method for class 'fixedLRT'
print(x,...)
```

Arguments

object	An object of class HLfit, as returned by the fitting functions in spaMM.
x	The return object of HLfit or related functions.
verbose	For summary.HLfit, whether to print the screen output that is the primary purpose of summary. verbose=FALSE may be convenient when summary is used as an extractor. For summary.fixedLRT, whether to print the model fits or not.
max.print	Controls options("max.print") locally.
details	A vector with elements controlling whether to print some obscure details. Element ranCoefs=TRUE will print details about random-coefficients terms (see Details); and element p_value="Wald" will print a p-value for the t-value of each fixed-effect coefficient, assuming a gaussian distribution of the test statistic (but, beyond the generally questionable nature of p-value tables, see e.g. LRT and fixedLRT for alternative testing approaches).
...	further arguments passed to or from other methods.

Details

The random effect terms of the linear predictor are of the form **ZLv**. In particular, for **random-coefficients models** (i.e., including random-effect terms such as (z|group) specifying a random-slope component), correlated random effects are represented as **b** = **Lv** for some matrix **L**, and where the elements of **v** are uncorrelated. In the output of the fit, the Var. column gives the variances of the correlated effects, **b=Lv**. The Corr. column(s) give their correlation(s). If details is

TRUE, estimates and SEs of the (log) variances of the elements of **v** are reported as for other random effects in the `Estimate` and `cond.SE.` columns of the table of lambda coefficients. However, this non-default output is potentially misleading as the elements of **v** cannot generally be assigned to specific terms (such as intercept and slope) of the random-effect formula, and the representation of **b** as **Lv** is not unique.

Value

These methods return the object invisibly. They print details of the (lower level) HLfit results in a convenient form.

Examples

```
## see examples of corrHLfit usage
```

`update.HLfit` *Updates a fit*

Description

`update` and `update_resp` will update and (by default) re-fit a model. They do this mostly by extracting the call stored in the object, updating the call and evaluating that call. Using `update` is a risky programming style (see Note). `update_resp` handles a new response vector as produced by `simulate`.

Usage

```
## S3 method for class 'HLfit'
update(object, formula., ..., evaluate = TRUE)
update_resp(object, newresp, ..., evaluate = TRUE)
```

Arguments

`object`	A return object from an HLfit call.
`formula.`	Changes to the formula. Beware of the syntax: see update.formula for details.
`newresp`	New response vector.
`...`	Additional arguments to the call, or arguments with changed values. Use *name* = `NULL` to remove the argument with given *name*.
`evaluate`	If TRUE, evaluate the new call else return the call.

Value

An HLfit fit of the same type as the input object, or a call object, depending on the `evaluate` value.

Note

`update`, as a general rule, is tricky. `update` methods are easily affected in a non-transparent way by changes in variables used in the original call. For example `foo <-rep(1,10)` `m <-lm(rnorm(10)~1,weights=foo)` `rm(foo)` `update(m,.~.)` `# Error` To avoid such problems, `spaMM` tries to avoid references to variables in the global environment, by enforcing that the data are explicitly provided to the fitting functions by the `data` argument, and that any variable used in the `prior.weights` argument is in the data.

`spaMM`'s `update` method was all the more tricky when spaMM called `stats::update.formula` whose results endorse `stats`'s (sometimes annoying) convention that a formula without an explicit intercept term actually includes an intercept. `spaMM::update` now avoids this problem. **Formula updates should still be carefully checked**, as getting them perfect has not been on the priority list.

See Also

See also `HLCor`, `HLfit`.

Examples

```
data("wafers")
## First the fit to be updated:
wFit <- HLfit(y ~X1*X3+X2*X3+I(X2^2)+(1|batch),family=Gamma(log),
          resid.model = ~ X3+I(X3^2) ,data=wafers)

newresp <- simulate(wFit)
update_resp(wFit,newresp=newresp)

# For estimates given by Lee et al., Appl. Stochastic Models Bus. Ind. (2011) 27:  315-328:
# Refit with given beta or/and phi values:

betavals <- c(5.55,0.08,-0.14,-0.21,-0.08,-0.09,-0.09)
# reconstruct fitted phi value from predictor for log(phi)
Xphi <- with(wafers,cbind(1,X3,X3^2)) ## design matrix
phifit <- exp(Xphi %*% c(-2.90,0.1,0.95))
upd_wafers <- wafers
upd_wafers$off_b <- wFit$`X.pv` %*% betavals
update(wFit,formula.= . ~ offset(off_b)+(1|batch), data=upd_wafers,
       ranFix=list(lambda=exp(-3.67),phi=phifit))

## There are subtlety in performing REML fits of constrained models,
##   illustrated by the fact that the following fit does not recover
##   the original likelihood values, because dispersion parameters are
##   estimated but the REML correction changes with the formula:
upd_wafers$off_f <- wFit$`X.pv` %*% fixef(wFit) ## = predict(wFit,re.form=NA,type="link")
update(wFit,formula.= . ~ offset(off_f)+(1|batch), data=upd_wafers)
## To maintain the original REML correction, Consider instead
update(wFit,formula.= . ~ offset(off_f)+(1|batch), data=upd_wafers,
       REMLformula=formula(wFit))  ## recover original p_v and p_bv
## Alternatively, show original wFit as differences from betavals:
update(wFit,formula.= . ~ . +offset(off_f), data=upd_wafers)
```

vcov *Extract covariance or correlation components from a fitted model object*

Description

summary(<fit object>)$beta_table returns the table of fixed-effect coefficients as it is printed by summary, including standard errors and t-values. vcov returns the variance-covariance matrix of the fixed-effects coefficients. Corr returns a correlation matrix of random effects.

VarCorr returns (co)variance parameters of random effects, and residual variance, from a fit object, in a data frame format roughly consistent with the method of objects of class "lme", in particular including columns with consistent names for easier extraction. One may have to consult the summary of the object to check the meaning of the contents of this data frame (e.g., of 'variance' coefficients for non-gaussian random effects). Other extractors to consider are get_ranPars and get_inits_from_fit, the latter providing parameters in a form suitable for initializing a fit.

Usage

```
## S3 method for class 'HLfit'
vcov(object, ...)
## S3 method for class 'HLfit'
VarCorr(x, sigma = 1, ...)
Corr(object, ...)
```

Arguments

object, x	A fitted model object, inheriting from class "HLfit", as returned by the fitting functions in spaMM.
sigma	ignored argument, included for consistency with the generic function.
...	Other arguments that may be needed by some method.

Value

vcov returns a matrix. Corr returns a list, for the different random effect terms. For each random-effect term, the returned element is a non-trivial unconditional correlation matrix of the vector "**v**" of random effects (**v** as defined in see Details of HLfit) for this term, if there is any such matrix. Otherwise the returned element is a information message.

VarCorr returns either NULL (if no variance to report, as for a poisson GLM) or a data frame with columns for the grouping factor, term, variance of random effect, standard deviation (the root of the variance), and optionally for correlation of random effect in random-coefficient terms. Information about the residual variance is included as the last row in the data frame, when relevant (gaussian- or Gamma-response models with single scalar parameter; beware the meaning of the residual variance parameter for Gamma-response models).

See Also

get_inits_from_fit and get_ranPars.

Examples

```
data("wafers")
m1 <- HLfit(y ~ X1+X2+(1|batch), resid.model = ~ 1 ,data=wafers, method="ML")
vcov(m1)

# Example from VarCorr() documentation in 'nlme' package
data("Orthodont",package = "nlme")
sp1 <- fitme(distance ~ age+(age|Subject), data = Orthodont, method="REML")
VarCorr(sp1)
```

`wafers` *Data from a resistivity experiment for semiconductor materials.*

Description

This data set was reported and analyzed by Robinson et al. (2006) and reanalyzed by Lee et al. (2011). The data "deal with wafers in a single etching process in semiconductor manufacturing. Wafers vary through time since there are some variables that are not perfectly controllable in the etching process. For this reason, wafers produced on any given day (batch) may be different from those produced on another day (batch). To measure variation over batch, wafers are tested by choosing several days at random. In this data, resistivity is the response of interest. There are three variables, gas flow rate (x1), temperature (x2), and pressure (x3) and one random effect (batch or day)." (Lee et al 2011).

Usage

```
data("wafers")
```

Format

The data frame includes 198 observations on the following variables:

y resistivity.

batch batch, indeed.

X1 gas flow rate.

X2 temperature.

X3 pressure.

Source

This data set was manually pasted from Table 3 of Lee et al. (2011). Transcription errors may have occurred.

References

Robinson TJ, Wulff SS, Montgomery DC, Khuri AI. 2006. Robust parameter design using generalized linear mixed models. Journal of Quality Technology 38: 38–65.

Lee, Y., Nelder, J.A., and Park, H. 2011. HGLMs for quality improvement. Applied Stochastic Models in Business and Industry 27, 315-328.

Examples

```
## see examples in the main Documentation page for the package.
```

`welding` *Welding data set*

Description

The data give the results of an unreplicated experiment for factors affecting welding quality conducted by the National Railway Corporation of Japan (Taguchi and Wu, 1980, cited in Smyth et al., 2001). It is a toy example for heterocedastic models and is also suitable for illustrating fit of overparameterized models.

Usage

```
data("welding")
```

Format

The data frame includes 16 observations on 10 variables:

Strengh response variable;

... nine two-level factors.

Source

The data were downloaded from http://www.statsci.org/data/general/welding.txt on 2014/08/19 and are consistent with those shown in table 5 of Bergman and Hynén (1997).

References

Bergman B, Hynén A (1997) Dispersion effects from unreplicated designs in the 2^{k-p} series. Technometrics, 39, 191–98.

Smyth GK, Huele AF, Verbyla AP (2001). Exact and approximate REML for heteroscedastic regression. Statistical Modelling 1, 161-175.

Taguchi G, Wu Y (1980) Introduction to off-line quality control. Nagoya, Japan: Central Japan Quality Control Association.

Examples

```
data("welding")
## toy example from Smyth et al.
HLfit(Strength ~ Drying + Material,resid.model = ~ Material+Preheating ,data=welding)
## toy example of overparameterized model
HLfit(Strength ~ Rods+Thickness*Angle+(1|Rods),resid.model = ~ Rods+Thickness*Angle ,data=welding)
```

`ZAXlist` *ZAXlist class and (cross) products for ZAL matrix*

Description

A `ZAXlist` object is a representation of the "ZAL" matrix as a list of descriptors of each `ZAL` block for each random effect.

This documentation is for development purposes and may be incomplete. The objects and methods are not part of the programming interface and are subject to modification without notice.

Usage

```
# new("ZAXlist", LIST=.)
```

Slots

`LIST`: A list whose each block is either a `(M|m)atrix`, or a list with two elements (and addtional class `ZA_QCHM`): `ZA`, and the Cholesky factor `Q_CHMfactor` of the precision matrix (`L=solve(Q_CHMfactor,system="Lt`

Index