Package: clustGLMM (via r-universe)

November 17, 2026

Title Model-Based Clustering of Mixed-Type Longitudinal Data

Version 1.0.1

Encoding UTF-8

Maintainer Jan Vavra <vavraj@karlin.mff.cuni.cz>

Description Provides tools for Bayesian estimation and inference for modelling clusterwise multivariate regression models for numeric, count, binary, ordinal and count outcomes observed repeatedly on the same units and where possible relations among outcomes are captured through a joint distribution of random effects. The clusters are defined through cluster-specific parameters, which the analyst can choose, e.g., with respect to the regression coefficients. In particular, the model specification for each regression model via the formula is specific to the outcome and consists of four parts: (1) fixed - regression coefficients common to all clusters, (2) group - group-specific regression coefficients, (3) random - random effects specific for each unit, (3) offset - name of an offset variable (if needed). Estimation is performed using MCMC sampling combining Gibbs and Metropolis-Hastings steps. Post-processing tools allow to assess convergence and address label switching and provide visual diagnostics. Units may be classified based on sampled allocation indicators or by exploiting the posterior distribution of the classification probabilities. For more details see Vavra et al. (2024) <doi:10.1007/s11222-023-10304-5>.

License GPL-2

Imports coda, colorspace, gaussquad, graphics, grDevices, HDInterval, MASS, methods, mvtnorm, nnet, RcppHungarian, splines, stats, utils

Depends R (>= 4.0.0)

Suggests knitr, rmarkdown, bookdown, kableExtra, dplyr, DiagrammeR, DiagrammeRsvg, rsvg, mclust, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation yes

Author Jan Vavra [aut, cre] (ORCID:
<https://orcid.org/0000-0003-0068-5356>), Bettina Gruen [aut]
(ORCID: <https://orcid.org/0000-0001-7265-4773>), Gertraud
Malsiner-Walli [aut] (ORCID:
<https://orcid.org/0000-0002-1213-4749>), Arnost Komarek [aut]
(ORCID: <https://orcid.org/0000-0001-8778-3762>)

Repository https://vavrajan.r-universe.dev

Date/Publication 2026-08-19 16:00:14 UTC

RemoteUrl https://github.com/cran/clustGLMM

RemoteRef HEAD

RemoteSha b6be2a7658def0271c0dd4839f5bab08688837c8

Contents

clustGLMM-package 2
as_coda 4
clustering_probabilities_and_deviance 7
clustGLMM 12
default_varying 17
generate_longitudinal_mixed_type_data 21
get_scalar_samples 25
longitudinal_mixed_type_data 27
nice_nrow_ncol 28
plot.clustglmm 29
plot_cat_vs_x_grouped 32
plot_diagnostics 36
post_processing 42
predict.clustglmm 47
print.clustglmm 51
psurvey 53
slategray.colors 55
summary.clustglmm 56

Index **60**

`clustGLMM-package` *Model-Based Clustering of Mixed-Type Longitudinal Data*

Description

Provides tools for Bayesian estimation and inference for modelling clusterwise multivariate regression models for numeric, count, binary, ordinal and count outcomes observed repeatedly on the same units and where possible relations among outcomes are captured through a joint distribution of random effects. The clusters are defined through cluster-specific parameters, which the analyst can choose, e.g., with respect to the regression coefficients. In particular, the model specification for each regression model via the formula is specific to the outcome and consists of four parts: (1) fixed - regression coefficients common to all clusters, (2) group - group-specific regression coefficients, (3) random - random effects specific for each unit, (3) offset - name of an offset variable (if needed). Estimation is performed using MCMC sampling combining Gibbs and Metropolis-Hastings steps. Post-processing tools allow to assess convergence and address label switching and provide visual diagnostics. Units may be classified based on sampled allocation indicators or by exploiting the posterior distribution of the classification probabilities. For more details see Vavra et al. (2024) <doi:10.1007/s11222-023-10304-5>.

Details

The key function is `clustGLMM` which creates MCMC samples. The data, formulae and other auxiliary parameters need to be set up carefully before calling this function. The examples and tutorials may be used for guidance. Other functions contained in the package consist of:

- a function for generating longitudinal data of mixed-type: `generate_longitudinal_mixed_type_data`,
- functions to generate plots: `plot_traceplots`, `plot_traceplots_param`, `plot_kerneldensity`, `plot_kerneldensity_param`, `plot_ACF`, `plot_ACF_param`, `plot_ECDF`, `plot_ECDF_param`, `plot_clusters`, `plot_clusters_ECDF_param`, `plot_clusters_kerneldensity_param`, `plot.clustglmm`, `plot_num_vs_x_grouped`, `plot_cat_vs_x_grouped`,
- functions for post-processing the samples: `post_processing`, `permute_cluster_labels`,
- functions for internal computations.

Author(s)

Jan Vavra [aut, cre] (ORCID: <https://orcid.org/0000-0003-0068-5356>), Bettina Gruen [aut] (ORCID: <https://orcid.org/0000-0001-7265-4773>), Gertraud Malsiner-Walli [aut] (ORCID: <https://orcid.org/0000-0002-1213-4749>), Arnost Komarek [aut] (ORCID: <https://orcid.org/0000-0001-8778-3762>)

Maintainer: Jan Vavra <vavraj@karlin.mff.cuni.cz>

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

as_coda *Change the Output Formats of* clustGLMM

Description

These functions are used to process or rearrange data between different formats. The main function as_coda converts the draws item of the clustglmm class into a mcmc.list object compatible with functions from package **coda**. Other functions (usually internally used) allow us to change between different saving formats (mcnc$howsave = "list" or mcmc$howsave = "data.frame").

Usage

```
as_coda(mcmc, burnin=0, thin=1)
from_C_to_list(values, p, settings, yspecd1=NULL, yspecd2=NULL, d2spec = NULL, family)
from_C_to_matrix(values, p, settings, yspecd1=NULL, yspecd2=NULL, d2spec = NULL, family)
from_list_to_C(draws, iterations, p, settings, yspecd1=NULL, yspecd2=NULL,
               d2spec = NULL, family)
from_list_to_coda(mcmc, burnin=0, thin=1)
from_list_to_matrix(mcmc)
from_matrix_to_C(draws, iterations, p, settings)
from_matrix_to_coda(mcmc, burnin=0, thin=1)
from_matrix_to_list(mcmc)
```

Arguments

mcmc	an object of class clustglmm.
values	a vector of values to be stored.
p	a string containing the name of the parameter.
settings	a matrix containing the information about model parameters.
yspecd1	a list of outcome-specific sizes along the first dimension.
yspecd2	a list of outcome-specific sizes along the second dimension.
d2spec	a numeric vector containing the dimension sizes if the parameter has different second dimension for every element of the first dimension (applicable only for dynamic clustering probabilities).
family	a vector of distributional families named by the outcome name in data.
iterations	what iterations from a given chain to include.
draws	either a list or a matrix; e.g., an element of clustGLMM$draws for a certain chain.
burnin	the length of burn-in period (default 0), i.e., the number of initial samples to omit.
thin	thinning rate (default 1); only every thinth value is kept (must be a positive integer).

Details

Functions with C in the name are used internally in clustGLMM or clustering_probabilities_and_deviance to process data coming from C functions. These functions only convert only draws from specific chain for a specific parameter, which explains the arguments values, p or draws. They do not convert the whole clustglmm object.

The functions that change between howsave = "list", howsave = "data.frame" and coda output go through every chain and every saved parameter, thus, convert the whole clustglmm object.

Function as_coda is a wrapper function that triggers the respective function from_..._to_coda depending on the current value of mcmc$howsave.

Value

For functions ...to_list, the output is a carefully structured list. Each chain ch in 1:mcmc$chains is saved in mcmc$draws[[ch]]. Then one can select a specific parameter which may have even other sublists. For example, group-specific effects for a numeric outcome would be accessed by mcmc$draws[[ch]]$beta_num[[g]][[mcmc$Nums[1]]] where g is the group.

For functions ...to_matrix, the output is a separate list for each chain. Each element of this list is a matrix with carefully named columns forming a data.frame. The first column is m and contains the iteration number to which the row belongs to. Other columns are model parameters. For example, the first effect specific for group 2 for numeric outcome called "ynum" would be named "beta_num_ynum(2)[1]". Note that the name of the variable has been pasted after the parameter name, then follows a group number in parentheses (only if parameter group-specific) and it ends with the dimensional coordinates in square brackets.

For functions ...to_C, the output is a long vector where all the parameter values are stacked together in the order assumed in the C functions.

For functions ...to_coda, the output is a "mcmc.list" object from **coda**. Then you can use this class and related methods for monitoring, investigating, plotting. However, a minor warning: the number of parameters is quite huge and can get overwhelming. Therefore, a set of plotting functions designed for the different parameter blocks is provided with this package, see plot.clustglmm.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM, plot.clustglmm, clustering_probabilities_and_deviance

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")
```

```
# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###-------------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###-------------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###-------------------------------------------------------------------------------
###     Calling clustGLMM()
###-------------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###-------------------------------------------------------------------------------
###     Changing formats
###-------------------------------------------------------------------------------

## How to change from "data.frame"="matrix" to "list":
mcmc <- from_matrix_to_list(mcmc)

## How to change from "list" to "data.frame"="matrix":
mcmc <- from_list_to_matrix(mcmc)
```

```
## How to change from "data.frame"="matrix" to mcmc from 'coda' package
codamcmc <- from_matrix_to_coda(mcmc, burnin = 10, thin = 1)

## How to change from any howsave type to mcmc from 'coda' package
codamcmc <- as_coda(mcmc, burnin = 10, thin = 1)
```

`clustering_probabilities_and_deviance` *Evaluate Clustering Probabilities and Deviance of Output of* `clustGLMM`

Description

To evaluate the clustering probabilities, all latent variables need to be integrated out. No analytic closed form solution is available for the integral in case of joint random effects for mixed-type outcomes. Numerical approximations are obtained using Laplace's approximation or Adaptive Gaussian Quadrature (AGQ). The samples of the model parameters from `clustGLMM` are sent to C for faster evaluation.

Usage

```
clustering_probabilities_and_deviance(mcmc, id, data, dynamic_prob = FALSE,
  start = 1, end = mcmc$iter, thin = 1, chains = mcmc$chains,
  howsave = mcmc$howsave,
  tolerance = mcmc$tuning$double$tolerance,
  maxiter = mcmc$tuning$integer$maxiter,
  maxnrep = mcmc$tuning$integer$maxnrep,
  NGQ = 1)
```

Arguments

`mcmc`	an object of class `"clustglmm"` (as returned by function `clustGLMM`).
`id`	a string, name of the column containing the unit ID. New units not used in MCMC sampling may be used here.
`data`	a data.frame containing outcomes and regressors including the unit ID variable `id`.
`dynamic_prob`	a Boolean; indicates if the clustering probabilities are computed dynamically. That is, if unit `i` is observed `ni` times, then `ni` different probabilities are computed, each based on the first `j` observations for `j in 1:ni`.
`start`	an iteration number where to start.
`end`	an iteration number where to end.
`thin`	a thinning that should be used between `start` and `end`; e.g., if `thin = 10` only every 10th iteration is approximated; default is 1.
`chains`	chain number(s) for which the approximation should be computed.
`howsave`	a string declaring how the sampled chains should be returned as `draws`. One of:

"list" draws is a list indexed by chain numbers, parameters will be saved in a structured list in which it is easy to orient,

"matrix" draws is a large matrix, where one row corresponds to one iteration of one specific chain and columns are all model parameters,

"cmcmc" draws is a list indexed by chain numbers and contains the condensed output of the inner C function (not recommended!).

`tolerance` tolerance used when computing the norm of the shift within Newton-Raphson.

`maxiter` maximum number of iterations allowed during Newton-Raphson.

`maxnrep` maximum number of repetitions of Newton-Raphson from different starting points.

`NGQ` number of points for Adaptive Gaussian Quadrature; if `NGQ = 1`, then simple Laplace's approximation is used. Do not use values higher than 7, since there are `NGQ^totnran` summands with `totnran` denoting the dimension of the random effects.

Details

An object of class "clustglmm" can be used regardless of the format, i.e., `howsave = "list"` or `howsave = "matrix"`.

The data.frame `data` containing the outcomes and regressors does not have to be the same as used for MCMC sampling. Even completely new units are allowed here.

The indexing of probabilities is more complicated when `dynamic_prob = TRUE`. In this case, for one unit there will be as many probabilities as observations belong to this unit. If unit `i` is observed 3 times, there will be 3 probabilities: one where only the first observation is used, a second where the first two observations are used and a third where all observations are used. With more available data the classification towards one of the clusters should be more pronounced. In the end, there will be as many probabilities as rows of `data`. However, it does not mean that the rows correspond to each other, e.g., when `data` is not ordered by unit IDs. The reason is that we use a `for` cycle that goes through all units and then for a specific unit through all its observations.

The code can handle the case where `data` is not sorted by ID. However, for one specific unit the data are assumed to be ordered in time. This is internally used as ordering for the dynamic probability calculation. However, we recommend to insert data also sorted by ID so that pairing rows with the probabilities obtained with `dynamic_prob = TRUE` is easier.

Computations are usually expensive and a division of the workload into several jobs where different chains or even subsamples are processed simultaneously might be useful. This can be achieved by setting `start`, `end` and `thin` differently for different jobs. However, you need to divide this on your own als also gather the results back together. To do this properly keep in mind that the used iteration numbers are created by `seq(start, end, by = thin)`.

To approximate the `totnran`-dimensional integral over the random effects, we use Adaptive Gaussian Quadrature. First, the Newton-Raphson method is used to find the Taylor expansion of the log-pdf as a function of random effects. If `NGQ = 1` then the integral is approximated by evaluation only in the central point found by Newton-Raphson, which corresponds to simple Laplace's approximation. If `NGQ = 2, 3, ..., 7` then `hermite.he.quadrature.rules(n = NGQ, normalized = FALSE)` from package **gaussquad** is used to find the roots and weights of the Hermite quadrature rules. These are combined for every dimension of random effects, which yields `NGQ^totnran` points at which the log-pdf needs to be evaluated. So, think carefully whether higher approximation precision with high `NGQ` is worth the expensive evaluation.

Value

An object of class "clustglmm" is returned so that already existing methods are applicable. Most of the outputs are taken from the given mcmc object.

Depending on the choice of howsave = "list" or howsave = "matrix", the $draws element contains a list of outputs for different chains (from chains) organized in either list or matrix format. Each chain then contains the following quantities:

dev_i a vector of individual unit contribution to the model deviance,

deviance a scalar containing the model deviance (sum of all individual contributions),

pUig_int a group-specific vector of length equal to number of unique id labels containing clustering probabilities towards one of mcmc$G clusters. The mapping between ids and numbers is given via the item $numbered_unique_ids.

Outputs $clustering and $draws are derived from pUig_int by averaging and evaluating, which cluster has the highest posterior probability.

Author(s)

Jan Vávra

References

Pinheiro, J.C., Chao, E.C. Efficient Laplacian and adaptive Gaussian quadrature algorithms for multilevel generalized linear mixed models. J. Comput. Graph. Stat. 15(1), 58–81 (2006). https://doi.org/10.1198/106186006x96962

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")
```

```
## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 1  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

## Extract the last state
inits <- mcmc$last

## Initialize with inits = Continue in sampling from the last state
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  inits = inits,
                  G = Gmax, iter = 100, nchains = nchains)

## Note that sampling takes time, here we generate too short chains.
## Sample longer chains for more faithful posterior approximation.
## Parallelization for different chains is recommended.
## See the vignette for more examples.

###---------------------------------------------------------------------------
###     Sample completely new data
###---------------------------------------------------------------------------
nnew <- 100
n_i <- 4
G <- 2
timepar <- "mix"
catpar <- "no"
catsubjfix <- TRUE
xin1 <- 1
```

```
Kord <- 5
Kcat <- 4
kspec_bi_cat <- FALSE
corr <- matrix(c(1, -0.5, -0.5, -0.4, -0.4,
                 -0.5, 1, 0.3, 0.4, 0.5,
                 -0.5, 0.3, 1, 0.2, 0.3,
                 -0.4, 0.4, 0.2, 1, 0.1,
                 -0.4, 0.5, 0.3, 0.1, 1), byrow = TRUE, ncol = 5)
sdSigmas <- c(0.5, 0.5, 0.5, 0.5, 0.5)
Sigma <- diag(sdSigmas)

set.seed(27818)
ldatanew <- generate_longitudinal_mixed_type_data(n = nnew, n_i = 4,
                     G = 2, timepar = "cross", catpar = "fixed",
                     catsubjfix = TRUE,
                     xin1 = 1, Sigma = Sigma,
                     Kord = 5, Kcat = 4, kspec_bi_cat = FALSE)
newdata <- ldatanew$data

###-------------------------------------------------------------------------
###     Evaluate the clustering probabilities for newdata
###-------------------------------------------------------------------------

probs <- clustering_probabilities_and_deviance(mcmc, id, newdata,
                                               dynamic_prob = FALSE,
                                               start = 1,
                                               end = 100,
                                               thin = 2,
                                               chains = 1:nchains,
                                               tolerance = 1e-7,
                                               maxiter = 100,
                                               maxnrep = 20,
                                               NGQ = 1)

###-------------------------------------------------------------------------
###     Final clustering
###-------------------------------------------------------------------------
certainty <- probs$certainty
clustering <- probs$clustering
# To be classified in the cluster with maximal clustering probability,
# it has to overcome the given threshold, otherwise remains unclassified.
threshold <- 0.8
clustering[certainty < threshold] <- 0

ch <- 1
newdata$clustering <- clustering[,ch]

newdata1 <- newdata[newdata$j==1,]
table(newdata1$clustering, newdata1$g)
# row 0: unclassified
```

clustGLMM *Sample MCMC for a Mixture of GLMMs of Mixed-type*

Description

First, initial values are set based on separate univariate GLMMs for each individual outcome. Then, an inner C function is called to do the hard work of MCMC sampling (Metropolis within Gibbs sampling). Outputs are processed and returned in required format (either in nicely organized lists or one huge data.frame).

Usage

```
clustGLMM(formula, id, family, data, G,
          varying, save, param, tuning, inits,
          iter = 1000, nchains = 1, standardize = TRUE,
          howsave = c("data.frame", "list", "cmcmc"))
```

Arguments

formula	a structured list containing formulae for all modelled outcomes. Let y be a modelled outcome then formula[[y]] should be a list containing $fixed formula for effects common to all clusters, $group formula for cluster-specific effects, $random formula for id-specific effects, $offset a column name from data to be used as an offset; by default it is set to an empty string, i.e., "", which means no offset.
id	a string, the name of the column containing the unit ID.
family	a vector of distributional families named by the outcome name in data; see below for implemented options.
data	a data.frame containing the outcomes, regressors, and the unit ID variable indicated in id.
G	an integer, the number of mixture components (maximal number of clusters). Can be a vector of integers (of length nchains) to provide a different value for each chain.
varying	a named vector of Boolean values; declares which parameters should be group-specific (TRUE) with the following names: "prec_num", "c_ord", "InvSigma", "InvQ", "naY".
save	a named vector of Boolean values; declares which model parameters should be returned (TRUE) and which should be dropped (FALSE). See also 'Details'.
param	a named list of hyperparameters for prior distributions. See also 'Details'.
tuning	a list of integer valued and double-valued tuning parameters. See also 'Details'.
inits	a structured list of initial values; ideally the last generated values from a previous run. If missing, units are randomly assigned to components and parameters are initiated using values obtained based on separately estimated univariate models. See also 'Details'.

`iter` length of each sampled chain.

`nchains` number of chains to be sampled.

`standardize` a Boolean whether the numeric outcome and regressors should be scaled.

`howsave` a string declaring how the sampled chains should be returned as `draws`. One of:

- `"list"` `draws` is a list indexed by chain numbers, parameters will be saved in a structured list in which it is easy to orient,
- `"data.frame"` `draws` is a large data.frame, where one row corresponds to one iteration of one specific chain and columns are all model parameters,
- `"cmcmc"` `draws` is a list indexed by chain numbers and contains the condensed output of the inner C function (not recommended!).

Details

Column `id` in the `data` data.frame contains different IDs. For simplicity, these IDs are tied with numbers from 1 to total number of unique IDs. Then, unit-specific quantities are referred to by these integers instead of full IDs.

Outcomes in `data` are expected to belong to one of the following families:

numeric `"num"` or `"gaussian"`; double values without any restriction,

count `"poi"` or `"poisson"`; integer values 0, 1, 2, ...,

binary `"bin"` or `"bernoulli"`; factor of two levels, first will be baseline (0), other (1),

ordinal `"ord"` or `"cumulative"`; (ordered) factor of at least 3 levels, if not apriori ordered, ordering will correspond to the one by `factor(, ordered = TRUE)`,

categorical `"cat"` or `"categorical"`; factor of at least 3 levels, first category will be baseline.

Only these 5 types of outcome families are supported. The link functions cannot be changed. `family` is internally relabeled to contain only `"num"`, `"poi"`, `"bin"`, `"ord"`, `"cat"` and reordered to follow this order of types. The same ordering of outcomes is applied to the covariance matrix `Sigma` of random effects.

A named vector `varying` contains a Boolean for each model parameter that may be considered cluster-specific. The defaults are set by `default_varying`.

A named vector `save` contains a Boolean for each model parameter including some transformed parameters. For an overview on all model parameters including the default values used, see `default_save`.

A named list `param` contains the values of the hyperparameters needed to set up the priors. The default values are set up by `default_param`. By adjusting the values of `ae` and `be` different levels of sparsity can be achieved. The lower `ae/be` is, the sparser the results expected.

The list of initial values `inits` is convenient to supply in case one is intereseted in continuing the sampling using the last sampled values as initialization, e.g., if the chains have not yet converged. If `inits` is missing, then a completely new set of initial values for each of the sampled chains is generated in the following way:

1. Units are randomly (uniformly) distributed to `G` components and `w` is initiated with `1/G` for all components.

2. For each numeric outcome, a regression model with combined `fixed` and `group` formula is fit to set up initial values for fixed parameters `beta_num_fixed`. Then, the model is fitted again (if

possible) for each component separately to determine the initial values for group-specific effects `beta_num`. Depending on the choice of group-specificity for `prec_num` the initial values are chosen as the inverse of the error variance estimate of the corresponding model.

3. Effects for count outcomes are initialized in a similar way using `glm(, family = poisson(link = "log"))` instead.

4. Effects for binary outcomes are initialized in a similar way using `glm(, family = binomial(link = "logit"))` instead.

5. Effects and ordered intercepts for ordinal outcomes are initialized in a similar way using `polr(, Hess = TRUE)` from **MASS** instead.

6. Effects for general categorical outcomes are initialized in a similar way using `multinom()` from **nnet** instead.

7. Matrix `InvSigma` is initialized by diagonal matrices with gamma distributed (`rate = 1/init_sd_b^2`) diagonal.

8. Matrix `InvQ` is initialized as inverse of (averaged and scaled) `InvSigma`.

9. Random effects are sampled randomly from a centered normal distribution with `sd = param$init_sd_b`.

The algorithm also requires a set of tuning parameters in `tuning` that tune the sampling. For clarity, the parameters are split into two sublists depending on the expected type (`integer` or `double`). If `tuning` is missing, then the default values are set by `default_tuning`.

Value

An object of class `"clustglmm"` where the most important elements contained are:

`draws`	a list indexed by chain with generated samples saved either in a large data.frame (`howsave = "data.frame"`) or a carefully structured list (`howsave = "list"`).
`param_names`	individual parameter (column) names in a structured list by general parameter name.

In addition the object contains the inputs given to the function: `iter`, `nchains`, `family` (sorted by type), `formula`, `G` (a vector of Gs for all chains), `varying`, `save`, `howsave`, `param`, `tuning`, `standardize`.

Furthermore elements consisting of newly defined quantities are:

`iterations`	a list per each chain containing iterations available,
`modeGplus`	a vector of the most frequent number of non-empty components per chain,
`sameclusters`	a Boolean declaring whether the cluster labels in all sampled chains are the same (useful for plotting),
`clusters`	a list of ordered non-empty components for each sampled chain (`modeGplus` most frequently appearing cluster labels),
`clustering`	a matrix containing estimated cluster labels based on all sampled allocation indicators for each unit (row) and sampled chain (column),
`certainty`	a matrix containing the probability of assignment into the cluster specified in `clustering` based on all sampled allocation indicators for each unit (row) and sampled chain (column),
`inits`	a structured list of parameter values used as initial values,

`last` a structured list in the same format as `inits` to be used as initial values for additional sampling,

`InitType` string describing whether the initial values were given or created,

`call` summary of what has been fitted,

`iter` length of each sampled chain,

`n` total number of units (unique IDs),

`numbered_unique_ids` numbers `1:n` named by unique values from column `id`,

`Nums` names of numerical outcomes,

`Pois` names of count outcomes,

`Bins` names of binary outcomes,

`Ords` names of ordinal outcomes,

`Cats` names of categorical outcomes,

`nfix` a vector named by outcomes that declares the dimension size of fixed effects common to all clusters,

`ngrp` a vector named by outcomes that declares the dimension size of group-specific effects,

`nran` a vector named by outcomes that declares the dimension size of random effects specific to each unit,

`noff` a vector named by outcomes that declares whether an offset is included in formula,

`totnran` total dimension of random effects, `totnran = sum(nran)`,

`lfixnames` a list of names of fixed effects common to all clusters for each modelled outcome,

`lgrpnames` a list of names of group-specific effects for each modelled outcome,

`lrannames` a list of names of random effects specific to each unit for each modelled outcome,

`loffnames` a list of names of offset variables for each modelled outcomes,

`fixnames` unlisted `lfixnames`,

`grpnames` unlisted `lgrpnames`,

`rannames` unlisted `lrannames`,

`offnames` unlisted `loffnames`,

`nY` number of outcomes per type,

`Kord` number of categories for each ordinal outcome,

`Kcat` number of categories for each general categorical outcome,

`settings` a list (for each chain) of matrices where each row belongs to one model parameter and describes its properties such as: 1. is it saved?, 2. is it group-specific?, 3. is any dimension size dependent on outcome?, 4. on which type of outcome it depends?, 5. is it point, vector or matrix? 5. what are the dimension sizes?, 6. is the matrix symmetric?, 7. does the matrix contain diagonal?, 8. what is the total size of the parameter (including group-specificity)?; very useful matrix for organizing the samples and writing automated plotting functions,

yspecd1	a list of model parameters, which have their first dimension size dependent on outcomes and how large it is, e,g, fixed-effects beta_num_fix are vectors that can be of different size among outcomes under different formulae,
yspecd2	a list of matrix-valued model parameters, which have their second dimension size dependent on outcomes and how large it is, e.g. fixed-effects beta_cat_fix for categorical outcome y in Cats are matrices (Kcat[y] * nfix[y]) with both dimensions dependent on the outcome y,
isYna	a vector of Booleans that determine which outcome values are not observed, outcomes are ordered by Ys = c(Nums, Pois, Bins, Ords, Cats).
centers	a named vector of centers of numeric outcomes and regressors.
scales	a named vector of scales of numeric outcomes and regressors.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

generate_longitudinal_mixed_type_data, plot.clustglmm, default_varying, default_save, default_param, default_tuning

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###------------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###------------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
```

```
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###--------------------------------------------------------------------------
###     Calling clustGLMM()
###--------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

## Extract the last state
inits <- mcmc$last

## Initialize with inits = Continue in sampling from the last state
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  inits = inits,
                  G = Gmax, iter = 100, nchains = nchains)

## Note that sampling takes time, here we generate too short chains.
## Sample longer chains for more faithful posterior approximation.
## Parallelization for different chains is recommended.
## See the vignette for more examples.
```

`default_varying` *Setting Default Values in* `clustGLMM`

Description

Sets default values for the hyperparameters of the priors `default_param`, indicators of group-specificity `default_varying`, integer and double valued tuning parameters `default_tuning` and indicators of what variables should be saved `default_save`.

Usage

```
default_param(sparse = TRUE, ...)
```

```
default_save(...)
default_tuning(experience = FALSE, ...)
default_varying(...)
```

Arguments

sparse	a logical, if TRUE the rate and shape hyperparameters of e0 are set to values encouraging sparsity in the weights
experience	a logical, indicating if the default values are changed to values proven suitable from empirical experience
...	named scalars to override default values

Details

If arguments param, save, tuning and varying of clustGLMM are not specified, these functions are used to create the default values. They can also be used to create prototypical param, varying, tuning and save argument specifications and then adjust them.

Value

The output of default_varying is a named vector that contains a Boolean for each model parameter that may be considered group-specific. The default values are:

varying["prec_num"] = TRUE group-specific precisions for numeric outcomes,

varying["c_ord"] = TRUE group-specific ordered intercepts for ordinal outcomes,

varying["InvSigma"] = FALSE group-invariant combined precision matrix for all random effects,

varying["InvQ"] = FALSE group-invariant precision matrix used in prior for InvSigma,

varying["naY"] = FALSE group-invariant imputed missing outcome values.

The output of default_save is a named vector that contains a Boolean for each model parameter including some transformed parameters indicating if they are saved and included in the return object. The default values are:

save["beta_num_fix"] = TRUE fixed effects for numeric outcomes,

save["beta_num"] = TRUE group-specific effects for numeric outcomes,

save["prec_num"] = TRUE precisions for numeric outcomes,

save["sd_num"] = TRUE standard deviations for numeric outcomes (transformed from prec_num),

save["var_num"] = TRUE variances for numeric outcomes (transformed from prec_num),

save["beta_poi_fix"] = TRUE fixed effects for count outcomes,

save["beta_poi"] = TRUE group-specific effects for count outcomes,

save["beta_bin_fix"] = TRUE fixed effects for binary outcomes,

save["beta_bin"] = TRUE group-specific effects for binary outcomes,

save["beta_ord_fix"] = TRUE fixed effects for ordinal outcomes,

save["beta_ord"] = TRUE group-specific effects for ordinal outcomes,

save["c_ord"] = TRUE ordered intercepts for ordinal outcomes,

save["a_ord"] = FALSE transformed ordered intercepts for ordinal outcomes into unrestricted variables,

save["pi_ord"] = TRUE transformed a_ord into probabilities,

save["beta_cat_fix"] = TRUE fixed effects for general categorical outcomes,

save["beta_cat"] = TRUE group-specific effects for general categorical outcomes,

save["InvSigma"] = TRUE combined precision matrix for all random effects,

save["Sigma"] = TRUE combined covariance matrix for all random effects (transformed from InvSigma),

save["sdSigma"] = TRUE standard deviations for all random effects (transformed from InvSigma),

save["corSigma"] = TRUE correlations between all random effects (transformed from InvSigma),

save["detInvSigma"] = FALSE determinant of InvSigma matrix,

save["InvQ"] = FALSE precision matrix used in prior for InvSigma,

save["Q"] = FALSE covariance matrix used in prior for InvSigma (transformed from InvQ),

save["detInvQ"] = FALSE determinant of InvQ matrix,

save["b"] = FALSE a matrix of random effects for all outcomes and units,

save["w"] = TRUE mixture weights of length G that sum up to 1,

save["ng"] = TRUE cluster occupancies of length G that sum up to number of units,

save["loglik"] = TRUE NOT a log-likelihood but a very crude approximation to it created as an easy to evaluate byproduct of full-conditional cluster allocation probabilities,

save["pUig"] = FALSE a matrix of full-conditional cluster allocation probabilities,

save["U"] = TRUE a vector of current allocations of units into clusters (values in 1, ..., G),

save["Gplus"] = TRUE a number of non-empty components,

save["e0"] = TRUE precision parameter for prior of w,

save["naY"] = FALSE imputed missing outcome values.

The output of default_param is a named list that contains values of the hyperparameters needed to set up the priors. The default values are:

param[["InvSigma_df"]] = 1 prior degrees of freedom for Wishart prior for InvSigma (dimension of this matrix is automatically added to this value),

param[["InvQ_df"]] = 1 prior degrees of freedom for Wishart prior for InvQ (dimension of this matrix is automatically added to this value),

param[["prec_num_shp"]] = 1 prior shape for precision parameter prec_num for numeric outcomes,

param[["prec_num_rte"]] = 10 prior rate for precision parameter prec_num for numeric outcomes,

param[["beta_num_fix_sd"]] = 1 standard deviation for fixed effects for numeric outcomes,

param[["beta_num_sd"]] = 1 standard deviation for group-specific effects for numeric outcomes,

param[["beta_poi_fix_sd"]] = 1 standard deviation for fixed effects for count outcomes,

param[["beta_poi_sd"]] = 1 standard deviation for group-specific effects for count outcomes,

param[["beta_bin_fix_sd"]] = 1 standard deviation for fixed effects for binary outcomes,

`param[["beta_bin_sd"]] = 1` standard deviation for group-specific effects for binary outcomes,

`param[["beta_ord_fix_sd"]] = 1` standard deviation for fixed effects for ordinal outcomes,

`param[["beta_ord_sd"]] = 1` standard deviation for group-specific effects for ordinal outcomes,

`param[["api_prior"]] = 1` prior parameter for Dirichlet distribution of `pi_ord` (same for all levels),

`param[["beta_cat_fix_sd"]] = 1` standard deviation for fixed effects for general categorical outcomes,

`param[["beta_cat_sd"]] = 1` standard deviation for group-specific effects for general categorical outcomes,

`param[["init_b_sd"]] = 0.01` standard deviation for sampling initial values of random effects `b`,

`param[["e0_shp"]] = 1` prior shape of `e0`,

`param[["e0_rte"]] = 100` prior rate of `e0`,

`param[["InvV"]] = 0.01` diagonal element of the prior inverse scale matrix for `InvQ`.

By adjusting the values of `e0_shp` and `e0_rte` different levels of sparsity can be achieved. The lower `e0_shp/e0_rte` is, the more sparse results are expected. With `sparse = TRUE` the prior distribution is a gamma distribution with `e0_shp` = 1 and `e0_rte` = 100. Otherwise `e0_shp` = 4 and `e0_rte` = 1 will be used, which prefers non-sparse solutions.

The output of `default_tuning` is a named list of parameters that tune the sampling algorithm. For clarity, the parameters are split into two sublists depending on the expected type (`integer` or `double`). The default values are:

`$integer$freq_proposal_update = 10` every `freq_proposal_update` iterations change the proposal distributions for Metropolis steps,

`$integer$times_proposal = 1` how many times a new state is proposed within one Metropolis step,

`$integer$maxiter = 25` maximum iterations allowed during Newton-Raphson update of proposal distribution,

`$integer$maxnrep = 100` maximum number of repetitions of Newton-Raphson with a different start location,

`$integer$kspec_bi_cat = 0` Boolean as integer, if `TRUE` then each categorical outcome has a different set of random effects for each of `Kcat-1` levels,

`$double$const_proposal_beta_poi_fix = 1.0` constant by which the proposal direction for `beta_poi_fix` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_poi = 1.0` constant by which the proposal direction for `beta_poi` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_bin_fix = 1.0` constant by which the proposal direction for `beta_bin_fix` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_bin = 1.0` constant by which the proposal direction for `beta_bin` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_ord_fix = 1.0` constant by which the proposal direction for `beta_ord_fix` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_ord = 1.0` constant by which the proposal direction for `beta_ord` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_cat_fix` = `1.0` constant by which the proposal direction for `beta_cat_fix` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_beta_cat` = `1.0` constant by which the proposal direction for `beta_cat` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_a_ord` = `1.0` constant by which the proposal direction for `a_ord` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_b` = `1.0` constant by which the proposal direction for `b` is multiplied (const_proposal^2 to variance),

`$double$const_proposal_e0` = `1.0` constant by which the proposal direction for `e0` is multiplied (const_proposal^2 to variance),

`$double$tolerance` = `1e-07` tolerance when computing the norm of the shift within Newton-Raphson.

With `experience` = `TRUE` the chosen values will be set to those proven suitable from empirical experience.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

`clustGLMM`

Examples

```
(param <- default_param())
(param <- default_param(sparse = FALSE))
(save <- default_save())
(tuning <- default_tuning())
(tuning <- default_tuning(experience = TRUE))
(varying <- default_varying())
```

`generate_longitudinal_mixed_type_data`

Generate an Artificial Longitudinal Dataset with Mixed-type Outcomes

---

Description

Five different longitudinal outcomes are sampled accoring to the model described by Vávra et al. (2024). Column `i` is the unit id, `j` is the number of observation for `i`th unit. Numeric `ynum`, count `ypoi`, binary `ybin`, ordinal `yord` and categorical `ycat` outcomes are sampled according to predictors `pred_...` which are constructed from the intercept, time covariate `x`, categorical covariate `f` and the random effects `b_...`. The definition of the predictor can be adjusted.

Usage

```
generate_longitudinal_mixed_type_data(n, n_i = 4, G = 2, timepar = "parallel",
  catpar = "no", catsubjfix = TRUE, xin1 = 1, Sigma,
  Kord = 5, Kcat = 4, kspec_bi_cat = FALSE)
```

Arguments

`n`	number of sample units.
`n_i`	number of observations per unit.
`G`	number of clusters, only `G=2` and `G=3` are predefined.
`timepar`	a string declaring what is the parametrization of time covariate `x`. One of `no`, `parallel`, `cross`, `spline`, `mix`.
`catpar`	a string declaring what is the parametrization of binary covariate `f`. One of `no`, `fixed`, `group`.
`catsubjfix`	a logical value; should the values of categorical covariate be fixed for the same unit or alternate among observations?
`xin1`	`1/xin1` is the width of observational window for one unit, so `xin1` declares how many times does the observational period of one unit fit into the overall data gathering period.
`Sigma`	the covariance matrix for the random intercepts (of dimension 5, one for each outcome). Can be a list of covariance matrices, in such a case the model assumes group-specific covariance matrix.
`Kord`	number of categories for ordinal outcome.
`Kcat`	number of categories for categorical outcome (only up to 4 categories).
`kspec_bi_cat`	a logical value; should the random effects be specific to each level of categorical outcome? If so, the dimension of Sigma should be larger.

Details

The function is limitted in the number of clusters `G` since it is quite laborious to define all the true values of model parameters (e.g. effects β).

Time covariate `x` is sampled so that observations for one unit are close to each other. First, half-length of the observational window `xmid = 1/(2xin1)` is defined. Then, values for a unit are sampled by uniform distribution on `[-xmid, xmid]` and randomly shifted by multiples of `xmid`. Time covariate is limitted to interval [0, 1].

The spline parametrization (`library("splines")`) of the time covariate is of `degree = 2` and has `knots = c(0, 0.25, 0.5, 0.75, 1)`

Binary covariate `f` is sampled with probability 0.5 either once per unit (`catsubjfix = TRUE`) or repeatedly for each observation (`catsubjfix = FALSE`).

The group allocation indicator `g` is sampled uniformly on $\{1, \ldots, G\}$ for each unit.

Then the linear predictors for each observation and outcome are created according to selected parametrization of time covariate `x` (`timepar`) and parametrization of binary covariate `f` (`catpar`). The intercepts are always group-specific which ensures at least some difference among clusters.

Choices for `timepar`:

no covariate `x` does not modify the predictor,

parallel clusters have the same linear trend in time `x` but different intercepts,

cross clusters have different linear trends in time `x` (one is inscreasing, another decreasing),

spline each cluster has a different randomly generated spline parametrization,

mix outcomes have different selection of all of the above (currently, numeric is `spline`, count is `cross`, binary is `parallel`, ordinal is `spline`, categorical is `cross`).

Choices for `catpar`:

no covariate `f` does not modify the predictor,

fixed clusters have the same effect of `f`,

group clusters have different effects of `f`.

Numerical outcome is sampled from normal distribution with group-specific standard deviations (from 0.5 to 1.0).

Count outcome is sampled from Poisson distribution with rate made by `exp()` of the predictor.

Binary outcome (0 or 1) is sampled with probability computed by `plogis`.

Ordinal outcome (ordered factor) is sampled by probabilities computed by sequential differences of `plogis` applied to shifted predictor.

Categorical outcome (unordered factor) is sampled by probabilities computed by `softmax` function from several predictors specific for each category; `softmax(x) = exp(x) / sum(exp(x))` where `x` is `Kcat`-dimensional. By default, predictor for the first category is set to 0 for identifiability purposes.

Value

A list containing:

`data`	a data frame with covariates, random effects, predictors and sampled outcomes,
`sd_num`	a list of true values of standard deviations for numeric covariate depending on the total number of clusters `G`,
`beta_num`	a list of true values (named vector) of β parameters for numeric outcome for each cluster,
`beta_poi`	a list of true values (named vector) of β parameters for count outcome for each cluster,
`beta_bin`	a list of true values (named vector) of β parameters for binary outcome for each cluster,

beta_ord	a list of true values (named vector) of β parameters for ordinal outcome for each cluster,
c_ord	a list of true values of intercepts specific to each level of the ordinal outcome for each cluster,
beta_cat	a list of true values (matrix with named columns) of β parameters for categorical outcome for each cluster.

Author(s)

Implemented by Jan Vávra for performing a simulation study in Vávra et al. (2024).

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

Examples

```
n <- 100
n_i <- 4
G <- 2
timepar <- "mix"
catpar <- "no"
catsubjfix <- TRUE
xin1 <- 1
Kord <- 5
Kcat <- 4
kspec_bi_cat <- FALSE
corr <- matrix(c(1, -0.5, -0.5, -0.4, -0.4,
                 -0.5, 1, 0.3, 0.4, 0.5,
                 -0.5, 0.3, 1, 0.2, 0.3,
                 -0.4, 0.4, 0.2, 1, 0.1,
                 -0.4, 0.5, 0.3, 0.1, 1), byrow = TRUE, ncol = 5)
sdSigmas <- c(0.5, 0.5, 0.5, 0.5, 0.5)
Sigma <- diag(sdSigmas)

set.seed(31415)
ldata <- generate_longitudinal_mixed_type_data(n = n, n_i = n_i,
                     G = G, timepar = timepar,
                     catpar = catpar, catsubjfix = catsubjfix,
                     xin1 = xin1, Sigma = Sigma,
                     Kord = Kord, Kcat = Kcat, kspec_bi_cat = kspec_bi_cat)
data <- ldata$data
head(data)

# The same as the "longitudinal_mixed_type_data"
# which will be used in other examples
data("longitudinal_mixed_type_data")
```

`get_scalar_samples` *Extract Samples for Specified Scalar Model Parameter*

Description

This function takes an output of `clustGLMM` and finds all values related to specified model parameter in a given range (possibly thinned). The implementation reacts not only to what is given but also to what is NOT given. To use it appropriately you need to know which model parameters are group-specific, outcome-dependent and many dimensions do they have.

Usage

```
get_scalar_samples(mcmc, what, gspec, dimspec, yspec,
                   burnin = 0, thin = 1,
                   iterations, chains)
```

Arguments

`mcmc`	an output of `clustGLMM` accepted in both formats (list or matrix).
`what`	a name of the parameter to be extracted; see `names(mcmc$save)` for possible options.
`gspec`	a group specifier (values in `1:mcmc$G`); if not group-specific parameter this *should be missing*.
`dimspec`	a dimension specifier; not specifying this is possible only when the parameter is scalar.
`yspec`	an outcome specifier; if the parameter is not outcome-dependent then this *should be missing*.
`burnin`	the length of burn-in period (default 0), i.e. number of initial samples to ignore for inference.
`thin`	thinning rate (default 1); only every `thin`th value is used for inference (must be a positive integer).
`iterations`	a vector of iterations numbers to extract, overrides any given `burnin` or `thin`.
`chains`	a vector of chain numbers to be used.

Details

Function `get_scalar_samples` is used mainly internally in plotting functions to find appropriate values given the name of the parameter `what`, group specification `gspec` (a number of a cluster), outcome specification `yspec` (name of the outcome) and dimension specification `dimspec`. For vector parameters `dimspec` is just a single number, if `what` is a matrix parameter then 2 numbers are expected.

`iterations` declares which `m` values should be gathered. If `iterations` is missing, the returned values are regulated by changing `burnin` and `thin`. First, only iterations with `m > burnin` are selected, then, from the remaining values only `seq(1, total_remaining_count, by=thin)` are selected. Also, `post_processing` could turn some of the cluster-specific variables into NA, hence, if `gspec` is missing, these cases are avoided.

Value

A list containing `whatLAB` (the default label for this scalar parameter) and `draws` a list with the draws.

`draws` is a list (for each element of `chains`) containing `data.frame` of three columns:

m iteration numbers,

chain the chain number,

y the values of the required parameter.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

`clustGLMM`, `plot.clustglmm`

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
```

```
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###-------------------------------------------------------------------------
###     Calling clustGLMM()
###-------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###-------------------------------------------------------------------------
###     The use of get_scalar_parameter_samples
###-------------------------------------------------------------------------
# Finding values for a specific parameter
# beta_num is group-specific, outcome-dependent and is a vector
get_scalar_samples(mcmc, what = "beta_num", gspec = 2, dimspec = 1, yspec = "ynum",
                   burnin = 0, thin = 1, chains = 1:nchains)

# beta_num_fix is not group-specific, but outcome-dependent and is a vector
get_scalar_samples(mcmc, what = "beta_num_fix", dimspec = 1, yspec = "ynum",
                   burnin = 0, thin = 1, chains = 1:nchains)

mcmc$varying["InvSigma"]
# InvSigma is not group-specific, not outcome-dependent, but is a symmetric matrix
# only elements [i,j] where i<=j are allowed
get_scalar_samples(mcmc, what = "InvSigma", dimspec = c(1,1),
                   burnin = 50, thin = 1, chains = 1:nchains)

# corSigma is not group-specific, not outcome-dependent, but is a symmetric matrix without diagonal
# only elements [i,j] where i<j are allowed
get_scalar_samples(mcmc, what = "corSigma", dimspec = c(1,2),
                   burnin = 50, thin = 10, chains = 1:nchains)
```

---

longitudinal_mixed_type_data

A Simulated Dataset of Longitudinal Mixed-Type Data

Description

An artificially created dataset by generate_longitudinal_mixed_type_data to demonstrate the use of clustGLMM. For details about the parameter setting see the example at the bottom of generate_longitudinal_mixed_

Usage

```
data("longitudinal_mixed_type_data")
```

Format

A data frame with 400 rows and 32 columns.

`i` an id variable

`x` continuous regressor

`j` order of observation for given id

`bs1`,`bs2`,`bs3`,`bs4`,`bs5` B-spline basis for `x`

`f` factor regressor

`g` a true group label from 1, 2, 3

`b_num`,`b_poi`,`b_bin`,`b_ord`,`b_cat` random intercept for the outcome of numerical type, analogously for other types

`pred_num`,`pred_poi`,`pred_bin`,`pred_ord`,`pred_cat1`,`pred_cat2`,`pred_cat3`,`pred_cat4` predictor for the outcome of numerical type, analogously for other types, categorical has multiple predictors for different categories

`c_ord1`,`c_ord2`,`c_ord3`,`c_ord4` ordered intercept for level 1, analogously for other levels

`ynum`,`ypoi`,`ybin`,`yord`,`ycat` an outcome of numerical type, analogously for other types

Examples

```
data(longitudinal_mixed_type_data)
summary(longitudinal_mixed_type_data)
```

---

nice_nrow_ncol *Determine Optimal Matrix Dimensions Given the Number of Cells*

Description

Did you ever want to arrange *n* small plots to one big plot of matrix grid? To do so, the easiest way is to set `par(mfrow = c(nrow, ncol))`. Once, you can set it manually. But what if you need to do this over and over for different *n* values? The following function proposes `nrow` and `ncol` such that `nrow * ncol >= n` and the remainder `nrow * ncol - n` is the smallest possible.

Usage

```
nice_nrow_ncol(n, increasing = TRUE, maxsteps = 20, maxratio = 3)
```

Arguments

n	number of cells to be covered.
increasing	a Boolean, if TRUE then the values are returned in increasing (non-decreasing) order, if FALSE then nrow >= ncol.
maxsteps	how many integers away from sqrt{n} to try.
maxratio	what is the maximal ratio between nrow and ncol?

Details

Function starts at integer ceiling{sqrt{n}} and for each integer x between ceiling{sqrt{n}} and ceiling{sqrt{n}} + maxsteps finds the lowest integer y such that x*y >= n. Among all found pairs of x and y that fulfill the condition x/y <= maxratio it chooses the one that has the smallest remainder x*y - n. If more pairs achieve the optimum, the pair with the smallest ratio x/y is chosen. Depending on increasing the optimal pair c(y,x) or c(x,y) is returned.

Value

Two integers c(nrow, ncol).

Author(s)

Jan Vávra

Examples

```
nice_nrow_ncol(10)
nice_nrow_ncol(11)
nice_nrow_ncol(15)
nice_nrow_ncol(15, increasing = FALSE)

# (2, 17) does not fulfill the condition on maxratio
nice_nrow_ncol(34, maxratio = 3)

# (2, 17) fulfills the condition on maxratio
nice_nrow_ncol(34, maxratio = 10)

# ceiling(sqrt(34)) = 6 but 6+5=11 < 17 and pair (2, 17) is not even considered
nice_nrow_ncol(34, maxsteps = 5, maxratio = 10)
nice_nrow_ncol(34, maxsteps = 11, maxratio = 10)
nice_nrow_ncol(34, maxsteps = 12, maxratio = 10)
```

---

plot.clustglmm *Plot the Output of* clustGLMM

Description

Triggers plotting functions for selected parameter depending on the choice of the plot.

Usage

```
## S3 method for class 'clustglmm'
plot(x, what="ng", which="traceplots", ng_trace_chain_split=TRUE, ...)
```

Arguments

x	an object of class "clustglmm", an output of function clustGLMM.
what	a string containing the name of the parameter.
which	a string declaring the type of the plot to be created.
ng_trace_chain_split	a logical indicating ...
...	other arguments.

Details

Depending on which the following functions are called:

ACF creates auto-correlation function plot, calls plot_ACF or plot_ACF_param.

ECDF creates empirical cumulative distribution function plot, calls plot_ECDF or plot_ECDF_param.

kerneldensity creates kernel density estimator plot, calls plot_kerneldensity or plot_kerneldensity_param.

traceplots creates traceplots (samples vs iteration), calls plot_traceplots or plot_traceplots_param.

clusters_ECDF creates empirical cumulative distribution function plot where different clusters are compared (what needs to be a cluster-specific parameter), calls plot_clusters or plot_clusters_ECDF_param.

clusters_kerneldensity creates kernel density estimator plot where different clusters are compared (what needs to be a cluster-specific parameter), calls plot_clusters or plot_clusters_kerneldensity_param.

If the user tries to specify one of: gspec or dimspec, it is treated as an attempt to plot only a single scalar parameter (functions without ..._param). Be careful to specify the dimensions appropriately (including yspec). Inner function get_scalar_samples triggers a warning or even an error if misspecified.

Otherwise if neither gspec nor dimspec are given, then ..._param function is triggered. It is designed to create plots for all chains and all dimensions related to the given what model parameter. The output is organized into multiple carefully arranged layout. The only way to plot a smaller subset of parameters is via yspec argument, which accepts even multiple outcome values. However, only plots for an intersection of given and plausible outcomes for the given parameter are returned.

These functions except also burnin (initial part of the chain to be ignored), thin (plot only every thinth sample) and chains (a vector of chain numbers to be plotted).

Value

No return value, only creates plots.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
```

```
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###---------------------------------------------------------------------------
###     Using plot() function on clustglmm object
###---------------------------------------------------------------------------

# traceplots by default
plot(mcmc)
plot(mcmc, "Gplus")
plot(mcmc, "beta_num", burnin=10, thin=2, chains=c(1))

# change the plot type
plot(mcmc, "w", "ACF")
plot(mcmc, "sdSigma", "ECDF")
plot(mcmc, "corSigma", "kerneldensity")
plot(mcmc, "beta_bin", "clusters_kerneldensity")
plot(mcmc, "beta_ord", "clusters_ECDF")
```

`plot_cat_vs_x_grouped` *Plot Clustered Longitudinal (Panel) Data*

Description

Functions that work with longitudinal data and plot the outcomes vs a covariate with respect to some grouping, e.g. clustering found by `clustGLMM`.

Usage

```
plot_num_vs_x_grouped(data, y = "y", x = "x", group, id = "i", units,
                      xbreaks, add_lowess = TRUE, alpha = 0.7,
                      legend_placement = "topright")
plot_cat_vs_x_grouped(data, y = "y", x = "x", group,
                      xbreaks, main = NA, margin = 0.05, reverse = FALSE)
```

Arguments

`data`	a `data.frame` containing `y`, `x`, `group`, `id`.
`y`	a name of the outcome to be plotted.
`x`	a name of numeric or factor regressor.
`group`	a name of the clustering variable that divides different ids into groups. If missing, all observations are given the label "0" plotted with `slategray.colors` and `legend_placement` is set by default to `NULL`.
`id`	a name of the id variable.
`units`	a vector of unit ids to be plotted, if missing all ids are plotted.

xbreaks	a set of breaks along which to cut regressor x, if missing x is divided equidistantly into 10 categories.
add_lowess	a Boolean whether to add lowess curves for each group label.
alpha	a color transparency factor; applied only for overlapping boxplots if covariate x is factor.
main	plot title to be printed above the plot.
margin	how far from each other should different groups be displayed.
reverse	a Boolean whether to reverse the order of ordinal categories; if TRUE 0 on top, if FALSE 0 at the bottom.
legend_placement	a string describing the position of legend (see legend) or NULL to omit the legend.

Details

Colors are created automatically by rainbow_hcl from package colorspace. slategray.colors are used for group 0 (unclassified).

Output of plot_num_vs_x_grouped depends on the type of covariate x:

factor boxplots of the outcome y wrt levels of covariate x with a transparent color determined by group. Boxplots for the same group may overlap, hence, transparent colors.

numeric spaghetti-plots the outcome y wrt covariate x coloured by group. Only units included in units are plotted.

If x is not already a factor, plot_cat_vs_x_grouped divides the data by a numeric covariate x into groups divided by xbreaks. Then, for each group label in group (including group 0 = unclassified) a spineplot capturing the evolution of proportions of a categorical outcome y wrt the covariate x.

Value

The functions do not return anything but produce a plot.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###---------------------------------------------------------------------------
###     Clustering based on not post-processed data (using sampled indicators)
###---------------------------------------------------------------------------
### Take the sampled U indicators
threshold <- 0.5 # if not exceeded, unit remains unclassified
```

```
# posterior mode of Gplus=number of nonemtpy components
mcmc$modeGplus

# labels of non-empty clusters in each chain
mcmc$clusters

# clustering based on sampled indicators
myclustering <- mcmc$clustering

# the highest estimated clustering probability among all clusters
mcmc$certainty

# units with frequency ratio smaller than chosen threshold remain unclassified (group 0)
myclustering[mcmc$certainty < threshold] <- 0
clusters <- list()

# proposed clustering by each chain separately
for(ch in 1:mcmc$nchains){
  longitudinal_mixed_type_data[,paste0("clustering",ch)] <-
    myclustering[longitudinal_mixed_type_data[,id],ch]
  longitudinal_mixed_type_data[,paste0("certainty",ch)] <-
    mcmc$certainty[longitudinal_mixed_type_data[,id],ch]
}

data1 <- longitudinal_mixed_type_data[longitudinal_mixed_type_data$j==1,]
# Confusion matrices for each chain separately:
table(data1$clustering1, data1$g)
# We can also see the suitable permutation for labels

###---------------------------------------------------------------------------
###     Plotting the resulting clustering
###---------------------------------------------------------------------------

# Plotting numeric outcomes
plot_num_vs_x_grouped(longitudinal_mixed_type_data, y = "ynum", x = "x",
                      group = "g", id = "i", units = 1:mcmc$n)

plot_num_vs_x_grouped(longitudinal_mixed_type_data, y = "ynum", x = "x",
                      group = "clustering1", id = "i", units = 1:mcmc$n)

# Plotting binary, ordinal, categorical outcomes
plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "ybin", x = "x",
                      group = "g")

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "ybin", x = "x",
                      group = "g", reverse = TRUE)

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "ybin", x = "x",
                      group = "clustering1")

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "yord", x = "x",
                      group = "g", reverse = TRUE)
```

```
plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "yord", x = "x",
                      group = "g", reverse = FALSE)

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "yord", x = "x",
                      group = "clustering1")

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "ycat", x = "x",
                      group = "g")

plot_cat_vs_x_grouped(longitudinal_mixed_type_data, y = "ycat", x = "x",
                      group = "clustering1")
```

`plot_diagnostics` *Plot MCMC Samples from* `clustGLMM`

Description

There are 4 main plots to be created:

`ACF` auto-correlation function,

`ECDF` empirical cumulative distribution function,

`kerneldensity` one-dimensional kernel density estimate,

`traceplots` parameter samples against the iteration number.

All of these for one specific parameter can be plotted by `plot_diagnostics`. These functions above work only for a univariate parameter. Plots for all elements of a multivariate parameter can be created using `..._param` functions. A special function is `plot_clusters` which allows to compare densities or ECDFs of univariate group-specific parameter across non-empty clusters.

Usage

```
plot_ACF(mcmc, scalar_samples, what, gspec, dimspec, yspec,
         burnin = 0, thin = 1, lag.max = 30, iterations, chains,
         COL, move.width = 0.3, labcex = 1, whatLAB)
plot_ECDF(mcmc, scalar_samples, what, gspec, dimspec, yspec,
          burnin = 0, thin = 1, iterations, chains,
          COL, labcex = 1, whatLAB)
plot_kerneldensity(mcmc, scalar_samples, what, gspec, dimspec, yspec,
                   burnin = 0, thin = 1, iterations, chains,
                   dolegend = FALSE, COL, labcex = 1, whatLAB)
plot_traceplots(mcmc, scalar_samples, what, gspec, dimspec, yspec,
                burnin = 0, thin = 1, iterations, chains,
                COL, labcex = 1, whatLAB)
plot_ng_trace_chain_split(mcmc,
                          burnin = 0, thin = 1, iterations, chains,
                          COL, labcex = 1, setparmfrow = TRUE)
plot_diagnostics(mcmc, scalar_samples, what, gspec, dimspec, yspec,
```

```
                  burnin = 0, thin = 1, lag.max = 30, iterations, chains,
                  trueval, setparmfrow = TRUE, COL, move.width = 0.3, whatLAB)
plot_ACF_param(mcmc, what = "ng", yspec, burnin = 0, thin = 1, iterations,
               lag.max = 30, chains, setparmfrow = TRUE, allclusters = FALSE)
plot_ECDF_param(mcmc, what, yspec, burnin = 0, thin = 1, iterations, chains,
                setparmfrow = TRUE, allclusters = FALSE)
plot_kerneldensity_param(mcmc, what, yspec, burnin = 0, thin = 1, iterations,
                         chains, setparmfrow = TRUE, allclusters = FALSE)
plot_traceplots_param(mcmc, what, yspec, burnin = 0, thin = 1, iterations,
                      chains, setparmfrow = TRUE, allclusters = FALSE)
plot_clusters(mcmc, what, dimspec, yspec, whichg,
              burnin = 0, thin = 1, iterations, chains,
              setparmfrow = TRUE, doKern = TRUE, doECDF = TRUE,
              COL, labcex = 0.8, whatLAB, wherelegend = "topright")
plot_clusters_kerneldensity_param(mcmc, what, whichg, yspec, burnin = 0,
                                  thin = 1, iterations, chains,
                                  setparmfrow = TRUE)
plot_clusters_ECDF_param(mcmc, what, whichg, yspec, burnin = 0, thin = 1,
                         iterations, chains, setparmfrow = TRUE)
```

Arguments

mcmc	an output of clustGLMM accepted in both formats (list or matrix).
scalar_samples	an output of get_scalar_samples, overrides any given burnin or thin.
what	a name of the parameter to be extracted; see names(mcmc$save) for possible options.
gspec	a group specifier (values in 1:mcmc$G); if not group-specific parameter this *should be missing*.
dimspec	a dimension specifier; not specifying this is possible only when the parameter is scalar.
yspec	an outcome specifier; if the parameter is not outcome-dependent then this *should be missing*.
burnin	the length of burn-in period (default 0), i.e. number of initial samples to ignore for inference.
thin	thinning rate (default 1); only every thinth value is used for inference (must be a positive integer).
lag.max	maximal lag value to be plotted in ACF.
iterations	a vector of iterations numbers to extract, overrides any given burnin or thin.
chains	a vector of chain numbers to be plotted.
trueval	a true value of the parameter to be plotted by vertical/horizontal line.
setparmfrow	whether par(mfrow = ..) should be set automatically before creating all plots.
allclusters	whether for group-specific parameters plot all the clusters (TRUE) or just those in mcmc$clusters (FALSE).
COL	colors for different chains; default values are set by diverge_hcl(mcmc$nchains, c = 80, l = 70).

move.width	ACF for different chains are jittered around the actual lag by seq(-move.width, move.width, length.out = length(chains)+1)
whatLAB	the label for the parameter; default corresponds to the label for that parameter that would be used in *matrix* format of the output.
labcex	cex value for label of plots.
dolegend	a Boolean declaring whether legend should be added to the plot.
whichg	a set of cluster labels to be compared in plot_clusters. By default mcmc$clusters is taken under consistency among selected chains. Can be a list in plot_clusters_kerneldensity_para and plot_clusters_ECDF_param.
doKern	a Boolean declaring whether kernel density estimates should be plotted for comparing different clusters.
doECDF	a Boolean declaring whether ECDFs should be plotted for comparing different clusters.
wherelegend	a string that declares where the legend should be placed.

Details

scalar_samples is a list (for each element of chains) containing data.frame of three columns: chain declaring the chain number, m declaring the iterations numbers and y containing chosen samples. This could be created using get_scalar_samples, see its help file for more details. If missing, get_scalar_samples is called anyway with the given specifications. If iterations is missing, first, only iterations with m > burnin are selected, then, from the remaining values only seq(1, total_remaining_count, by=thin) are selected. Parameter is selected by filling/missing what, gspec, yspec, dimspec. More details here: get_scalar_samples.

Description of the plots:

ACF Autocorrelation function plots have the lag on x-axis [0, lag.max] and auto-correlations in interval (-1, 1) on y-axis. By default, lag.max = min(30, mcmc$iter-1). Different chains have different colors and are slightly jittered for better visuality, see move.width.

ECDF Cumulative distribution function plots have the range of sampled values on x-axis and the cumulative probabilities in [0, 1] on y-axis. Different colors depict different chains of non-decreasing ECDFs.

kerneldensity Kernel density estimator plots have the range of sampled values on x-axis and the density value in [0, max(density)] on y-axis. Different colors depict different chains.

traceplot Traceplots have the range of the considered iteration numbers on x-axis and the sampled values of the parameter on y-axis. Different colors depict different chains. Stationarity of these plots is desired for reliable posterior approximation, if the plots are not stationary then consider continuation in sampling.

You can plot all 4 of these by plot_diagnostics. By default they are produced in 2 by 2 layout, but if you would prefer some different arrangement you can do that before calling the function and supressing setparmfrow = FALSE.

Using the functions above you can only plot a scalar model parameter which is specified by filling/missing what, gspec, yspec, dimspec. If you would like to plot results for all parameters that are associated with parameter what then ..._param functions can do this for you automatically. By default, only cluster-specific parameters for the (non-empty) clusters saved in mcmc$clusters

are arranged. If the labels across chains do not coincide, then chains are plotted separately. The exception is made for w and ng for which always all clusters are plotted. Moreover, there is special function plot_ng_trace_chain_split for plotting traces of ng for all clusters into one plot per chain (default by plot.clustglmm). Using function nice_nrow_ncol plots are arranged into a most effective grid. Moreover, parameters which are symmetric matrices (with or without diagonal) are arranged as expected and only the upper triangle is filled with plots. See examples below.

Special function in this collection is plot_clusters which allows to plot kernel density estimates or empirical cumulative distribution function of the same parameters for different cluster labels (whichg) into one plot. There are also plot_clusters_kerneldensity_param and plot_clusters_ECDF_param functions that create the plots automatically for a set of given parameter what, however, the plotted cluster labels only include those in mcmc$clusters of corresponding chain.

Value

The functions do not return anything but produce one or more plots.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM, get_scalar_samples, nice_nrow_ncol, post_processing.

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###-------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###-------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
```

```
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)
plot(mcmc)
plot(mcmc, ng_trace_chain_split=FALSE)

###---------------------------------------------------------------------------
###     Creating plots
###---------------------------------------------------------------------------

## Traceplots of number of non-empty clusters
# Plotting with defaults values:
# thin = 1, burnin = 0, chains = 1:mcmc$nchains
plot_traceplots(mcmc = mcmc, what = "Gplus")

## Traceplots of number of observations in cluster 1
plot_traceplots(mcmc = mcmc, what = "ng", gspec = 1)

# Omitting the burn-in-period
plot_traceplots(mcmc = mcmc, what = "ng", gspec = 1, burnin = 20)

# Thinned plots
plot_traceplots(mcmc = mcmc, what = "ng", gspec = 1, burnin = 20, thin = 5)

## First fixed beta coefficient for the first numerical outcome
plot_ACF(mcmc = mcmc, what = "beta_num_fix", yspec = mcmc$Nums[1], dimspec = 1)
plot_ECDF(mcmc = mcmc, what = "beta_num_fix", yspec = mcmc$Nums[1], dimspec = 1)
plot_kerneldensity(mcmc = mcmc, what = "beta_num_fix", yspec = mcmc$Nums[1],
                   dimspec = 1, dolegend = TRUE)
plot_traceplots(mcmc = mcmc, what = "beta_num_fix", yspec = mcmc$Nums[1],
```

```
                dimspec = 1)
plot_diagnostics(mcmc = mcmc, what = "beta_num_fix", yspec = mcmc$Nums[1], dimspec = 1)

## Correlation matrix between the random effects
# True correlation matrix used for generating the data
corr <- matrix(c(1, -0.5, -0.5, -0.4, -0.4,
                 -0.5, 1, 0.3, 0.4, 0.5,
                 -0.5, 0.3, 1, 0.2, 0.3,
                 -0.4, 0.4, 0.2, 1, 0.1,
                 -0.4, 0.5, 0.3, 0.1, 1), byrow = TRUE, ncol = 5)

par(mfrow = c(mcmc$totnran-1, mcmc$totnran-1), mar = c(4, 4, 1, 1))
for(i in 1:(mcmc$totnran-1)){
  for(j in 2:(mcmc$totnran)){
    if(i < j){
      plot_traceplots(mcmc = mcmc, what = "corSigma",
                      dimspec = c(i, j),
                      labcex = 0.6)
      abline(h = corr[i,j], col = "black", lty = 2)
    }else{
      plot(x = c(0, 1), y = c(0, 1), type = "n", xlab = "", ylab = "",
           xaxt = "n", yaxt = "n", bty = "n")
    }
  }
}
# or simply use
plot_traceplots_param(mcmc, "corSigma")
# but no true value displayed...

## Possible what arguments:
# "_param" - plots everything tied to the given what parameter
plot_traceplots_param(mcmc, "Gplus", thin = 5)
plot_traceplots_param(mcmc, "e0", thin = 5)
plot_traceplots_param(mcmc, "ng", thin = 5)
plot_traceplots_param(mcmc, "w", thin = 5)
plot_traceplots_param(mcmc, "sdSigma", thin = 5)
plot_traceplots_param(mcmc, "corSigma", thin = 5)
plot_traceplots_param(mcmc, "Sigma", thin = 5)
plot_traceplots_param(mcmc, "InvSigma", thin = 5)
plot_traceplots_param(mcmc, "prec_num", thin = 5)
plot_traceplots_param(mcmc, "beta_num_fix", thin = 5)
plot_traceplots_param(mcmc, "beta_num", thin = 5)
plot_traceplots_param(mcmc, "beta_poi_fix", thin = 5)
plot_traceplots_param(mcmc, "beta_poi", thin = 5)
plot_traceplots_param(mcmc, "beta_bin_fix", thin = 5)
plot_traceplots_param(mcmc, "beta_bin", thin = 5)
plot_traceplots_param(mcmc, "beta_ord_fix", thin = 5)
plot_traceplots_param(mcmc, "beta_ord", thin = 5)
plot_traceplots_param(mcmc, "c_ord", thin = 5)
plot_traceplots_param(mcmc, "beta_cat_fix", thin = 5)
plot_traceplots_param(mcmc, "beta_cat", thin = 5)
plot_traceplots_param(mcmc, "loglik", thin = 5)
```

```
## Other "_param" versions for different types of plots
plot_kerneldensity_param(mcmc, "beta_num")
plot_ECDF_param(mcmc, "prec_num")
plot_ACF_param(mcmc, "w")

## Comparing different cluster-specific parameters
## We recommend to do it for chains separately
ch <- 1
TAB <- table(mcmc$last[[ch]]$U)
nonempty <- as.numeric(names(TAB)[which(TAB>0)])
# non-empty clusters decided based on last sampled values
# by default, mcmc$clusters (based on all iterations) is taken

neff <- mcmc$ngrp[[mcmc$Nums[1]]]
oldpar <- par(mfrow = c(length(mcmc$Nums), neff), mar = c(4, 4, 1, 1))

for(y in mcmc$Nums){
  for(i in 1:neff){
    plot_clusters(mcmc = mcmc, what = "beta_num",
                  yspec = y, dimspec = i, whichg = nonempty,
                  setparmfrow = FALSE, chains = ch,
                  doECDF = FALSE)
    # Now you can add anything to the plot
    # e.g. true values, etc.
  }
}
par(oldpar)

# or simply use
plot_clusters_kerneldensity_param(mcmc = mcmc, what = "beta_num", chains = c(ch))
# where cluster labels are taken from mcmc$clusters
```

`post_processing` *Post-process the Chains Sampled with* `clustGLMM`

Description

Sampling MCMC chains via `clustGLMM` assumes underlying `Gmax` clusters and allows for learning the true number of underlying clusters. Many clusters are eventually empty and only some cluster labels survive. Given random initialization, different chains may end up with different labels of non-empty clusters including their total count. The function `post_processing` performs the post-processing suggested by Frühwirth-Schnatter (2011) and Malsiner-Walli et al (2016).

Usage

```
post_processing(mcmc, cols, iter.max = 200, nstart = 30)
permute_cluster_labels(mcmc, perm)
```

Arguments

mcmc	an output of clustGLMM accepted in both formats (list or data.frame).
cols	a vector of column names of cluster-specific variables to be used for k-means clustering. If missing, all cluster-specific parameters are used.
iter.max	the maximum number of iterations of kmeans algorithm allowed.
nstart	how many random sets for kmeans should be chosen?
perm	a list of permutation of cluster labels, one permutation for each chain. If missing, the chains are aligned to chain 1 based on the Hungarian algorithm, see package **RcppHungarian**.

Details

post_processing follows the post-processing suggested by Frühwirth-Schnatter (2011) and Malsiner-Walli et al. (2016) to identify the number of underlying cluster and to resolve possible issues with label-switching. All post-processing is done separately for each chain, keep in mind that the results among chains may differ due to different initial conditions. The procedure can be divided into several steps:

0) The procedure works with mcmc$draws only under howsave = "data.frame", so if howsave = "list" then mcmc is converted to the "matrix" format, post-processed and converted back to "list".

1) Take the most frequent Gplus value - modeGplus - the number of non-empty clusters across the sampled chain and find the posterior draws with Gplus = modeGplus. The indices are stored in inds and their length is iters.

2) Target all cluster-specific parameters. To clarify these are parameters with (g) in their column name, parameters w, U, pUig do not count. If you prefer to use just a subset of those parameters, you have to specify the column names by cols input variable.

3) Create submatrices of sampled cluster-specific parameters. Index by index in inds gather the cluster-specific parameters from only non-empty clusters. Due to label-switching it may happen that different cluster labels correspond to the non-empty clusters at each index. After these operations it may happen that values in one column correspond to different clusters.

4) Prepare the data for k-means clustering to resolve the label-switching. Gather values for different clusters below each other. That is, if there are L different cluster-specific parameters and iters samples, then the matrix is rearranged from iters * (L * modeGplus) to (iters * modeGplus) * L.

5) K-means clustering kmeans using centers=modeGplus, iter.max = iter.max, nstart = nstart is applied to matrix from 5). Ideally, rows corresponding to iteration m would be each clustered to a different mode. In such case, we would obtain a permutation for each row that aligns the meaning of the cluster labels across all iterations. Unfortunately, it may happen that some iterations do not result into clear permutation. In such case, this iteration has to be eliminated (replaced with NA), so the length of the chain used for inference is reduced from iters to possibly even lower number Mrhos.

6) Regroup the matrix of cluster-specific data back to the format in 3) while also correcting for the permutations found in 5). This should result in a matrix where all the values in one column correspond to the same cluster. This output matrix is called clusterspec.

7) Create the matrix of posterior draws by relabelling and permuting clustering-related parameters.

8) Update all other items saved in mcmc, see clustGLMM. Especially, clustering, certainty, G (now modeGplus, possibly different among chains!), iterations (omit iterations with NA values created by not leading to permutation in step 5)), settings (changed dimensions for cluster-specific parameters), param_names, last (list of last sampled values - now from the last iterations with modeGplus clusters, properly permuted) and call (summary of the post-processing). Many items remain the same as in mcmc, e.g., even the initial values.

permute_cluster_labels simply takes the list of permutations perm and reshuffles the cluster labels as requested. Useful when you want to align the meaning of cluster labels across different chains. For example, an element of the perm list should be a vector c(2, 3, 4, 1), if

cluster 2 is to become cluster 1,

cluster 3 is to become cluster 2,

cluster 4 is to become cluster 3,

cluster 1 is to become cluster 4.

If perm is missing, then labels within the first chain are fixed. For remaining chains we take the posterior means of group-specific parameters and compute the root mean square errors among all couples of cluster labels (between the first and currect chain). The resulting cost matrix is sent to HungarianSolver and second column of $pairs output is used as the permuation for the current chain. This automatic choice of perm is slowed down by the call of summary to mcmc.

Value

Both functions return back list of class clustGLMM, the same structure as mcmc, but with modified items.

Author(s)

Jan Vávra

References

Frühwirth-Schnatter, S. (2011) Dealing with Label Switching under Model Uncertainty, John Wiley & Sons, chap 10, pp 213–239

Malsiner-Walli, G., Frühwirth-Schnatter, S., Grün, B.(2016) Model-based clustering based on sparse finite Gaussian mixtures. Statistics and Computing 26:303–324. https://doi.org/10.1007/s11222-014-9500-2

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM, plot.clustglmm, nice_nrow_ncol, post_processing, kmeans.

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
```

```
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###-------------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###-------------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###-------------------------------------------------------------------------------
###     Calling clustGLMM()
###-------------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 200, nchains = nchains)

###-------------------------------------------------------------------------------
###     Post-processing the sampled chains
###-------------------------------------------------------------------------------

### First run the function post_processing
# takes some time with iter large....
set.seed(12345)
mcmc <- post_processing(mcmc)
mcmc
```

```
summary(mcmc)

# All plotting functions should work even with these post-processed draws
plot(mcmc, "beta_num", "clusters_kerneldensity", chains=1:nchains)

### Additionally permute so that the meaning of the number of clusters
### is kept the same across all chains

## first we have to recognize which chain has which meaning
neff <- mcmc$ngrp[["ynum"]]
oldpar <- par(mfrow = c(nchains,neff), mar = c(4,4,1,1))
for(ch in 1:nchains){
  TAB <- table(mcmc$last[[ch]]$U)
  nonempty <- as.numeric(names(TAB)[which(TAB>0)])

  for(i in 1:neff){
    plot_clusters(mcmc = mcmc, what = "beta_num",
                  yspec = "ynum", dimspec = i, whichg = nonempty,
                  setparmfrow = FALSE, chains = ch,
                  doECDF = FALSE)
  }
}
par(oldpar)

## confusion matrices with the true labelling
for(ch in 1:mcmc$nchains){
  longitudinal_mixed_type_data[,paste0("clustering",ch)] <-
    mcmc$clustering[longitudinal_mixed_type_data[,id],ch]
}

data1 <- longitudinal_mixed_type_data[longitudinal_mixed_type_data$j==1,]
table(data1$clustering1, data1$g)
table(data1$clustering2, data1$g)


### Example how to change the permutation so that results across chains align
### + align also with the true labelling within the given data
perm <- list()
perm[[1]] <- c(2, 1) # change if needed
perm[[2]] <- c(1, 2) # change if needed

# function to permute meaning of cluster labels in mcmc according to perm
mcmc1 <- permute_cluster_labels(mcmc, perm)

# If perm is not specified,
# Hungarian algorithm matches the labelling of other chains to chain number 1
mcmc2 <- permute_cluster_labels(mcmc)
# which in this case is different from the true labelling in data



###-------------------------------------------------------------------------
###     Plotting post-processed chains
```

```
###-------------------------------------------------------------------------
# Notice that now there are only modeGplus clusters and not original G

plot(mcmc1, "w", "traceplots", thin = 10)
plot(mcmc1, "w", "kerneldensity")
plot(mcmc1, "beta_num", "clusters_kerneldensity")
plot(mcmc1, "beta_poi", "clusters_kerneldensity")
plot(mcmc1, "beta_bin", "clusters_kerneldensity")
plot(mcmc1, "beta_ord", "clusters_kerneldensity")
plot(mcmc1, "beta_cat", "clusters_kerneldensity")

# Compare
plot(mcmc1, "beta_num", "clusters_kerneldensity")
plot(mcmc2, "beta_num", "clusters_kerneldensity")
```

`predict.clustglmm` *Predict Method for Class* `clustglmm`

Description

Predicts the linear combinations of fixed and group part of the regression of `clustGLMM` output. Transforms linear predictors to response level. Explores the posterior predictive distribution of the response.

Usage

```
## S3 method for class 'clustglmm'
predict(object, newdata, y = names(object$family)[1],
    type = c("link", "response", "posterior_predictive_response"),
    method = c("samples", "plugin"),
    what = c("fg", "f", "g"),
    posterior_predictive_by_cluster=TRUE,
    fun_posterior = mean, level = 0.95, interval = c("ET", "HPD"),
    chain = 1, burnin = 0, thin = 1,
    gs = object$clusters[[chain]], ...)
```

Arguments

`object`	an object of class `"clustglmm"`, an output of `clustGLMM` function.
`newdata`	a `data.frame` containing all columns needed to construct model matrices.
`y`	a string; outcome to be predicted.
`type`	what should be predicted; one of `"link"` (default), `"response"` or `"posterior_predictive_response"` see Details below.
`method`	the method of predicting; one of `"samples"` (default) or `"plugin"`, see Details below.
`what`	more detailed clarification to the selected `type = "link"`.

posterior_predictive_by_cluster
: should under the posterior predictive distribution for response be explored for each cluster separately?

fun_posterior
: a function summarizing samples into an estimator of location parameter of posterior distribution; e.g. mean or median.

level
: the credibility level of credible intervals; 0.95 by default.

interval
: the method of constructing credible intervals; one of "ET" (equal-tailed by default) or "HPD" (highest-posterior density), see Details below.

chain
: chain number to be used to do the prediction, the default is the first chain = 1.

burnin
: the length of burn-in period (default 0), i.e., number of initial samples to ignore for inference.

thin
: thinning rate (default 1); only every thinth value is used for inference (must be a positive integer).

gs
: cluster labels, for which prediction of cluster-specific should be made; all non-empty clusters by default.

...
: other arguments.

Details

Depending on type the following is computed:

link reconstructs the linear predictor for given outcome y. You can specify by what which part of the predictor should be reconstructed: "f", "g", "fg", which part of the predictor should be reconstructed: the fixed part invariant to all clusters, the group-specific part for chosen clusters gs or sum of both (default).

response linear predictor "fg" transformed depending on the outcome type. Not transformed for num. Transformed by exp for poi. Category probabilities for bin, ord and cat.

posterior_predictive_response explores the posterior predictive distribution. Depending on posterior_predictive_by_c either for each cluster separately or you learn a mixture over given clusters gs. For each iteration in object$draws[[chain]], samples latent parameters (allocation and random effects) and then an outcome value. It is important to know that only clusters in gs are considered here, other clusters are ignored.

Argument what only specifies option type="link":

f the fixed part of linear predictor invariant to all clusters,

g the group-specific part of linear predictor for chosen clusters gs,

fg sum of the two above.

As we approximate the posterior distribution with samples, there may be several ways on obtaining the posterior estimate of desired quantity (type). Depending on method we proceed in the following way:

samples treat the desired quantity as a parametric function, evaluate it with all (closely specified) samples in object$draws and estimate it by fun_posterior. This option allows for construction of credible intervals.

plugin estimate the model parameters with `fun_posterior` first and then plug them into the function defining the quantity. Does not provide credible intervals and `type="posterior_predictive_response"`.

The computation of credible intervals under `method="samples"` is given by `interval`:

ET equal-tailed credible interval determined out of samples from posterior distribution with `quantile` function with `probs = c((1-level)/2, 1 - (1-level)/2`.

HPD highest-posterior density credible interval determined out of samples from posterior distribution with `hdi` function from **HDInterval**.

For `method="plugin"` the credible intervals are not implemented.

Value

For given outcome `y` returns a list with

$fit the estimate of the posterior location obtained via function `fun_posterior` over samples,

$lwr the lower bound of credible interval, if `method="samples"`,

$upr the upper bound of credible interval, if `method="samples"`.

Moreover, `type="posterior_predictive_response"` provides

$samples the non-aggregated samples of the response,

$aux the auxiliary quantities used to each generate outcome. Linear predictor for numeric, exponential of linear predictor for count, probabilities for binary, ordinal and categorical outcomes.

The output is an array of up to 4 dimensions (dropped if only size 1) in this exact order:

1. the size of `newdata`,
2. the number of clusters `length(gs)`, unless just 1 (then dropped) if not group-specific (`what="f"`),
3. the dimension of samples, if not reduced by `fun_posterior` (only `$samples` and `$aux`).
4. the number of different outcome levels: 1 for numeric and count (then dropped), K (probabilities) or K-1 (predictors) for categorical outcome of K levels

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

`clustGLMM`

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###---------------------------------------------------------------------------
###     predict.clustglmm()
###---------------------------------------------------------------------------

# By default, predicts the first outcome in mcmc$family
```

```
predict(mcmc, newdata = subset(longitudinal_mixed_type_data, i %in% 1:3), type = "link")
# result is list of fit, lwr, upr matrices

# Predict the evolution in time for covariate f=0
newdata <- data.frame("x" = seq(0, 1, length.out=100), "f"=factor(0, levels=c(0,1)))

## Numeric outcome
# both fixed and group-specific part of the predictor
p1 <- predict(mcmc, newdata, y="ynum", type="link", what="fg")
# posterior predictive distribution of the response
p2 <- predict(mcmc, newdata, y="ynum", type="posterior_predictive_response")

matplot(newdata$x, p1$fit, type = "l", lty = 1, lwd = 2)
matplot(newdata$x, p1$lwr, type = "l", lty = 2, add = TRUE)
matplot(newdata$x, p1$upr, type = "l", lty = 2, add = TRUE)
matplot(newdata$x, p2$lwr, type = "l", lty = 3, add = TRUE) # needs longer chains
matplot(newdata$x, p2$upr, type = "l", lty = 3, add = TRUE) # needs longer chains

## Count outcome - conditional mean
# predictor
pl <- predict(mcmc, newdata, y="ypoi", type="link")
# conditional mean
p1 <- predict(mcmc, newdata, y="ypoi", type="response")
# posterior predictive distribution
p2 <- predict(mcmc, newdata, y="ypoi", type="posterior_predictive_response")

matplot(newdata$x, p1$fit, type = "l", lty = 1, lwd = 2)
matplot(newdata$x, p1$lwr, type = "l", lty = 2, add = TRUE)
matplot(newdata$x, p1$upr, type = "l", lty = 2, add = TRUE)
matplot(newdata$x, p2$lwr, type = "l", lty = 3, add = TRUE) # needs longer chains
matplot(newdata$x, p2$upr, type = "l", lty = 3, add = TRUE) # needs longer chains

## Categorical outcome - list of fit, lwr, upr 3D arrays (category dimension added)
predict(mcmc, newdata[1:5,], y="ycat", type="link", what="fg")
```

`print.clustglmm` *Print the Output of* `clustGLMM`

Description

A method for printing objects of class `clustglmm`, the outputs of `clustGLMM`.

Usage

```
## S3 method for class 'clustglmm'
print(x, which = "call", ...)
```

Arguments

x	an object of class "clustglmm", an output of clustGLMM function.
which	a vector containing the following strings "call" to print the overview of the estimated model, "clustering" to print the brief summary of the clustering.
...	other arguments.

Details

First, if "call" is in which, it prints x$call that summarizes what has been fitted including sample sizes, number of units, observations per unit, chosen maximal number of clusters, overview of outcomes and selected types, fixed-effects common to all groups, random-effects with the structure of Sigma matrix, group-specific formulae for outcomes, list of other group-specific parameters and summary of the MCMC sampling.

Next, if "clustering" is in which, it summarizes the clustering based on sampled indicators U for each chain separately. First, modeGplus, the most frequent value of Gplus - number of non-empty components, is determined. Then, all samples of cluster indicators U are gathered from all units to determine modeGplus most frequent cluster labels, stored in clusters. To each unit i we assign the cluster label from clusters that maximizes the frequency of samples of U[i], stored in clustering. We also keep the relative frequency to the assigned cluster, stored in certainty. By choosing reasonable threshold, e.g. 0.5, you can make a unit unclassified (0) if the certainty is not high enough: clustering[certainty < threshold] <- 0. If G = 1, then clustering summary is irrelevant.

Value

No returned value, only prints.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
```

```
# For more thorough analyses see the vignettes.


###-------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###-------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###-------------------------------------------------------------------------
###     Calling clustGLMM()
###-------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###-------------------------------------------------------------------------
###     Printing the results
###-------------------------------------------------------------------------

mcmc
print(mcmc, which = "clustering")
print(mcmc, which = c("call", "clustering"))
```

---

psurvey *A Simulated Panel Survey Data*

Description

An artificially created dataset to demonstrate the use of clustGLMM. It contains data about 500 people (i) each observed random number of times in a span of 20 years. The data were generated by assuming 3 latent groups (g). People in group 1 have lower income (did not finish university) than other two groups. Groups 2 and 3 are similar in income but differ in the attitude, while people in group 2 have more relaxing approach and spend money on trips, people in group 3 work hard on their career and enjoy life less.

Usage

```
data("psurvey")
data("psurvey_latent")
```

Format

psurvey is a data frame with 4512 observations on the following 6 variables

i a numeric vector - id of a person

j a numeric vector - order of observation for a person

age a numeric vector - age of a person when observation taken

income a numeric vector - income of a person in artificial units

ntrips a numeric vector - number of trips taken in the last year

satisfaction an ordered factor with levels 0 < 1 < 2 - level of self-evaluated satisfaction with their current lifestyle (0 not satisfied, 1 neither-nor, 2 satisfied)

Additionally, data frame psurvey_latent contains columns that are known from the creation process, but should not be used for analysis:

i a numeric vector - id of a person

j a numeric vector - order of observation for a person

g a numeric vector - a true group label from 1, 2, 3

b_num a numeric vector - true random intercept for income

b_poi a numeric vector - true random intercept for ntrips

b_ord a numeric vector - true random intercept for satisfaction

Examples

```
data(psurvey)
head(psurvey)
summary(psurvey)

data(psurvey_latent)
head(psurvey_latent)
```

`slategray.colors` *Slategray Color Palette*

Description

Creates a vector of n slategray colors.

Usage

```
slategray.colors(n, start = 0.3, end = 0.9)
slategrey.colors(n, start = 0.3, end = 0.9)
```

Arguments

n	the number of slategray colors to be in the palette.
start	starting slategray level in the palette (should be between 0 and 1 where zero indicates "slategray4" and one indicates "slategray1").
end	ending gray level in the palette.

Details

The function `slategray.colors` chooses a series of n slategray levels between `start` and end by interpolating 0=`slategray4`, 1/3=`slategray3`, 2/3=`slategray2`, `slategray1` = 1. The returned palette contains the corresponding gray colors. This palette is used in `plot_cat_vs_x_grouped` for levels of group 0.

`slategrey.colors` is an alias for `slategray.colors`.

Value

A vector of n slategray colors.

Author(s)

Jan Vávra

Examples

```
slategray.colors(5, start = 0.3, end = 0.9)
slategrey.colors(5, start = 0.1, end = 0.7)
```

summary.clustglmm *Compute the Summary Statistics of* clustGLMM *Output*

Description

Computes and prints the summary statistics for objects of class "clustglmm", the output from function clustGLMM.

Usage

```
## S3 method for class 'clustglmm'
summary(object, ...)
## S3 method for class 'summary.clustglmm'
print(x, which = c("mcmc", "inv_param", "clust_param"), chains = 1L,
      clusters, pattern = "", ...)
```

Arguments

object	an object of class "clustglmm" (as returned by function clustGLMM).
x	an object of class "summary.clustglmm" (as returned by the summary method for "clustglmm" objects) .
which	a character vector that indicates which summaries should be printed.
chains	a vector of chain numbers to print summary results for.
clusters	a list of cluster labels for which group-specific parameters are printed for each chain; if missing x$clusters is used.
pattern	a pattern to be used for matching with grep among parameter names to include in the summary, an empty string by default (i.e., matches all names from the selected category).
...	other arguments (such as burnin or thin for as_coda).

Details

print.summary prints a summary for each chain selected by argument chains. Depending on what is selected with the which argument, the following summaries are printed:

mcmc an overview of the sampled chains,

clustering a summary of the clustering,

inv_param cluster-invariant parameters,

clust_param clustering-related parameters (Gplus, e0, w),

latent_param latent parameters (random effects b, allocation indicators U, missing outcome values naY),

group_param group-specific parameters.

Regarding the clustering option it has to be noted that the summary is based on the sampled allocation indicators when using the raw output of clustGLMM, while it is based on the approximated clustering probabilities for the output of clustering_probabilities_and_deviance.

When printing the summary of the parameters, the argument pattern allows to reduce the list of printed parameters. Group-specific parameters are printed only for the cluster labels in clusters. By default, only parameters for non-empty clusters are printed. The output of summary contains both coda-like tables ("statistics" and "quantiles"), but only a combination of these two is printed (all quantiles and the mean).

Value

summary returns a list of:

call	a string describing the model defined to be estimated,
G	maximal number of clusters,
modeGplus	the most frequent number of non-empty clusters per each chain,
clustering	matrix of assigned clusters for each unit based on sampled indicators,
certainty	matrix of certainties (the highest clustering probability among G other) for each unit based on sampled indicators,
clusters	labels of non-empty clusters for each chain separately,
coda_summary	output of coda summary for each chain separately,
output	the type of output that is summarized: mcmc samples for clustGLMM, probs approximation of clustering probabilities and deviance for clustering_probabilities_and_deviance,
fnames	vector of parameter names invariant to clustering,
gnames	cluster-specific parameter names only for cluster 1,
lnames	latent variable names (allocation indicators, random effects, missing values),
cnames	clustering related parameter names.

Author(s)

Jan Vávra

References

Vávra, J., Komárek, A., Grün, B., Malsiner-Walli, G. Clusterwise multivariate regression of mixed-type panel data. Stat Comput 34, 46 (2024). https://doi.org/10.1007/s11222-023-10304-5

See Also

clustGLMM, clustering_probabilities_and_deviance, print.summary.clustglmm, summary.mcmc

Examples

```
### Get some longitudinal mixed type data
### here generated by generate_longitudinal_mixed_type_data()
data("longitudinal_mixed_type_data")

# Examples here demonstrate the basic use of the available functions on a toy dataset.
# For more thorough analyses see the vignettes.


###---------------------------------------------------------------------------
###     Prepare inputs for clustGLMM()
###---------------------------------------------------------------------------
id <- "i"

family <- c("num", "poi", "bin", "ord", "cat")
names(family) <- Ys <- c("ynum", "ypoi", "ybin", "yord", "ycat")

## Create list of formulae for each outcome
formula <- list()
for(y in Ys){
  formula[[y]] <- list()
  formula[[y]]$fixed <- ~ f
  formula[[y]]$group <- ~ x
  formula[[y]]$random <- ~ 1
  formula[[y]]$offset <- ""
}
formula
# intercept term will be group-specific
# since it appears in $group which has a preference over $fixed

## All other inputs need to be prepared,
## unless the default values are desired (varying, save, howsave, param, tuning).
## See the vignette for how these can be adjusted.

###---------------------------------------------------------------------------
###     Calling clustGLMM()
###---------------------------------------------------------------------------

Gmax <- 5     # maximal number of mixture components to be considered
nchains <- 2  # number of chains to be sampled

## No inits specified
set.seed(123456789)
mcmc <- clustGLMM(formula = formula, id = id, family = family,
                  data = longitudinal_mixed_type_data,
                  G = Gmax, iter = 100, nchains = nchains)

###---------------------------------------------------------------------------
###     summary.clustglmm()
###---------------------------------------------------------------------------
```

```
mcmc
s <- summary(mcmc)

# Choose chains - summary is done separately for each of them
print(s)
print(s, chains = 2)
print(s, chains = c(1, 2))

# Different types of summary
print(s, which = c("mcmc", "clustering"))
print(s, which = "inv_param")
print(s, which = "clust_param")
print(s, which = "latent_param")
print(s, which = "group_param")

# Restricting the rows by pattern
print(s, which = "group_param", pattern = "^beta_num")
print(s, which = "group_param", pattern = "num")
```

Index

∗ **datasets**
psurvey, 54
∗ **longitudinal_mixed_type_data**
longitudinal_mixed_type_data, 27
∗ **package**
clustGLMM-package, 2

as_coda, 4, *4*, *56*

clustering_probabilities_and_deviance, *5*, 7, *57*
clustGLMM, *3*, *5*, *7*, *9*, 12, *18*, *21*, *25, 26*, *28*, *31–33*, *37*, *39*, *42–44*, *49*, *52*, *54*, *56, 57*
clustGLMM-package, 2

default_param, *13*, *16*
default_param *(*default_varying*)*, 17
default_save, *13*, *16*
default_save *(*default_varying*)*, 17
default_tuning, *14*, *16*
default_tuning *(*default_varying*)*, 17
default_varying, *13*, *16*, 17

from_C_to_list *(*as_coda*)*, 4
from_C_to_matrix *(*as_coda*)*, 4
from_list_to_C *(*as_coda*)*, 4
from_list_to_coda *(*as_coda*)*, 4
from_list_to_matrix *(*as_coda*)*, 4
from_matrix_to_C *(*as_coda*)*, 4
from_matrix_to_coda *(*as_coda*)*, 4
from_matrix_to_list *(*as_coda*)*, 4

generate_longitudinal_mixed_type_data, *3*, *16*, 21, *28*
get_scalar_samples, 25, *30*, *38, 39*
grep, *56*

hdi, *49*
HungarianSolver, *44*

kmeans, *43, 44*

legend, *33*
longitudinal_mixed_type_data, 27

mcmc.list, *4*

nice_nrow_ncol, 28, *39*, *44*

permute_cluster_labels, *3*
permute_cluster_labels *(*post_processing*)*, 42
plogis, *23*
plot.clustglmm, *3*, *5*, *16*, *26*, 29, *39*, *44*
plot_ACF, *3*, *30*
plot_ACF *(*plot_diagnostics*)*, 36
plot_ACF_param, *3*, *30*
plot_ACF_param *(*plot_diagnostics*)*, 36
plot_cat_vs_x_grouped, *3*, 32, *55*
plot_clusters, *3*, *30*
plot_clusters *(*plot_diagnostics*)*, 36
plot_clusters_ECDF_param, *3*, *30*
plot_clusters_ECDF_param *(*plot_diagnostics*)*, 36
plot_clusters_kerneldensity_param, *3*, *30*
plot_clusters_kerneldensity_param *(*plot_diagnostics*)*, 36
plot_diagnostics, 36
plot_ECDF, *3*, *30*
plot_ECDF *(*plot_diagnostics*)*, 36
plot_ECDF_param, *3*, *30*
plot_ECDF_param *(*plot_diagnostics*)*, 36
plot_kerneldensity, *3*, *30*
plot_kerneldensity *(*plot_diagnostics*)*, 36
plot_kerneldensity_param, *3*, *30*
plot_kerneldensity_param *(*plot_diagnostics*)*, 36
plot_ng_trace_chain_split *(*plot_diagnostics*)*, 36

plot_num_vs_x_grouped, *3*
plot_num_vs_x_grouped
(plot_cat_vs_x_grouped), 32
plot_traceplots, *3*, *30*
plot_traceplots (plot_diagnostics), 36
plot_traceplots_param, *3*, *30*
plot_traceplots_param
(plot_diagnostics), 36
post_processing, *3*, *25*, *39*, 42, *44*
predict.clustglmm, 47
print.clustglmm, 51
print.summary.clustglmm, *57*
print.summary.clustglmm
(summary.clustglmm), 56
psurvey, 53
psurvey_latent (psurvey), 54

slategray.colors, *32*, *33*, 55
slategrey.colors (slategray.colors), 55
summary, *44*
summary.clustglmm, 56
summary.mcmc, *57*