Package ‘mcmcsae’

February 27, 2024

Version 0.7.7

License GPL-3

Title Markov Chain Monte Carlo Small Area Estimation

Type Package

LazyLoad yes

Encoding UTF-8

Description Fit multi-level models with possibly correlated
random effects using Markov Chain Monte Carlo simulation.
Such models allow smoothing over space and time and are useful in,
for example, small area estimation.

Date 2024-02-26

Depends R (>= 4.1.0)

Imports Matrix (>= 1.5.0), Rcpp (>= 0.11.0), methods, GIGrvg (>= 0.7),
loo (>= 2.0.0), matrixStats

Suggests dbarts, BayesLogit, lintools, splines, spdep, sf, bayesplot,
coda, posterior, parallel, testthat, roxygen2, knitr,
rmarkdown, survey

LinkingTo Rcpp, RcppEigen, Matrix, GIGrvg

VignetteBuilder knitr

RoxygenNote 7.3.1

Collate 'MCMCsim.R' 'MH.R' 'tabMatrix.R' 'MatrixUtils.R'
'RcppExports.R' 'TMVN_methods.R' 'TMVN_sampler.R'
'aux_closures.R' 'cMVN_sampler.R' 'cholesky.R'
'conjugate_gradients.R' 'family.R' 'formulas.R' 'kronprod.R'
'mc_bart.R' 'mc_block.R' 'mc_gen.R' 'mc_gl.R' 'mc_mec.R'
'mc_reg.R' 'mc_vfac.R' 'mc_vreg.R' 'mcmcsae.R' 'model_eval.R'
'modelmatrix.R' 'models.R' 'opts.R' 'parallel.R' 'plots.R'
'prediction.R' 'priors.R' 'random.R' 'samplers.R' 'sbc.R'
'sparse_template.R' 'utils.R'

NeedsCompilation yes

Author Harm Jan Boonstra [aut, cre],
Grzegorz Baltissen [ctb]

Maintainer Harm Jan Boonstra <hjboonstra@gmail.com>

Repository CRAN

Date/Publication 2024-02-27 03:40:02 UTC

R topics documented:

`mcmcsae-package` *Markov Chain Monte Carlo Small Area Estimation*

Description

Fit multi-level models with possibly correlated random effects using MCMC.

Details

Functions to fit multi-level models with Gaussian, binomial, multinomial, negative binomial or Poisson likelihoods using MCMC. Models with a linear predictor consisting of various possibly correlated random effects are supported, allowing flexible modeling of temporal, spatial or other kinds of dependence structures. For Gaussian models the variance can be modeled too. By modeling variances at the unit level the marginal distribution can be changed to a Student-t or Laplace distribution, which may account better for outliers. The package has been developed with applications to small area estimation in official statistics in mind. The posterior samples for the model parameters can be passed to a prediction function to generate samples from the posterior predictive distribution for user-defined quantities such as finite population domain means. For model assessment, posterior predictive checks and DIC/WAIC criteria can easily be computed.

`acceptance_rates` *Return Metropolis-Hastings acceptance rates*

Description

Return Metropolis-Hastings acceptance rates

Usage

```
acceptance_rates(obj, aggregate.chains = FALSE)
```

Arguments

`obj`	an mcdraws object, i.e. the output of function `MCMCsim`.
`aggregate.chains`	whether to return averages over chains or results per chain.

Value

A list of acceptance rates.

Examples

```
ex <- mcmcsae_example()
# specify a model that requires MH sampling (in this case for a modeled
#   degrees of freedom parameter in the variance part of the model)
sampler <- create_sampler(ex$model, data=ex$dat, formula.V=~vfac(factor="fA",
  prior=pr_invchisq(df="modeled")))
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4, store.all=TRUE)
(summary(sim))
acceptance_rates(sim)
```

`aggrMatrix` *Utility function to construct a sparse aggregation matrix from a factor*

Description

Utility function to construct a sparse aggregation matrix from a factor

Usage

```
aggrMatrix(fac, w = 1, mean = FALSE, facnames = FALSE)
```

Arguments

`fac`	factor variable.
`w`	vector of weights associated with the levels of `fac`.
`mean`	if `TRUE`, aggregation will produce (weighted) means instead of sums.
`facnames`	whether the factor levels should be used as column names for the aggregation matrix.

Value

A sparse aggregation matrix of class `tabMatrix`.

Examples

```
n <- 1000
f <- sample(1:100, n, replace=TRUE)
x <- runif(n)
M <- aggrMatrix(f)
all.equal(crossprod_mv(M, x), as.vector(tapply(x, f, sum)))
```

`brt` *Create a model component object for a BART (Bayesian Additive Regression Trees) component in the linear predictor*

Description

This function is intended to be used on the right hand side of the `formula` argument to `create_sampler` or `generate_data`. It creates a BART term in the model's linear predictor. To use this model component one needs to have R package **dbarts** installed.

Usage

```
brt(
  formula,
  X = NULL,
  n.trees = 75L,
  name = "",
  debug = FALSE,
  keepTrees = FALSE,
  ...
)
```

Arguments

formula	a formula specifying the predictors to be used in the BART model component. Variable names are looked up in the data frame passed as data argument to create_sampler or generate_data, or in environment(formula).
X	a design matrix can be specified directly, as an alternative to the creation of one based on formula. If X is specified formula is ignored.
n.trees	number of trees used in the BART ensemble.
name	the name of the model component. This name is used in the output of the MCMC simulation function MCMCsim. By default the name will be 'bart' with the number of the model term attached.
debug	if TRUE a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.
keepTrees	whether to store the trees ensemble for each Monte Carlo draw. This is required for prediction based on new data. The default is FALSE to save memory.
...	parameters passed to dbarts.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

References

H.A. Chipman, E.I. Georgea and R.E. McCulloch (2010). BART: Bayesian additive regression trees. The Annals of Applied Statistics 4(1), 266-298.

J.H. Friedman (1991). Multivariate adaptive regression splines. The Annals of Statistics 19, 1-67.

Examples

```
# generate data, based on an example in Friedman (1991)
gendat <- function(n=200L, p=10L, sigma=1) {
  x <- matrix(runif(n * p), n, p)
  mu <- 10*sin(pi*x[, 1] * x[, 2]) + 20*(x[, 3] - 0.5)^2 + 10*x[, 4] + 5*x[, 5]
  y <- mu + sigma * rnorm(n)
  data.frame(x=x, mu=mu, y=y)
}

train <- gendat()
test <- gendat(n=25)

# keep trees for later prediction based on new data
sampler <- create_sampler(
  y ~ brt(~ . - y, name="bart", keepTrees=TRUE),
  sigma.mod=pr_invchisq(df=3,  scale=var(train$y)),
  data = train
)
sim <- MCMCsim(sampler, n.chain=2, n.iter=700, thin=2,
  store.all=TRUE, verbose=FALSE)
```

```
(summ <- summary(sim))
plot(train$mu, summ$bart[, "Mean"]); abline(0, 1)
# NB prediction is currently slow

pred <- predict(sim, newdata=test,
  iters=sample(seq_len(ndraws(sim)), 100),
  show.progress=FALSE
)
(summpred <- summary(pred))
plot(test$mu, summpred[, "Mean"]); abline(0, 1)
```

CG_control   *Set options for the conjugate gradient (CG) sampler*

Description

Set options for the conjugate gradient (CG) sampler

Usage

```
CG_control(
  max.it = NULL,
  stop.criterion = NULL,
  preconditioner = c("GMRF", "GMRF2", "GMRF3", "identity"),
  scale = 1,
  chol.control = chol_control(),
  verbose = FALSE
)
```

Arguments

`max.it`	maximum number of CG iterations.
`stop.criterion`	total squared error stop criterion for the CG algorithm.
`preconditioner`	one of "GMRF", "GMRF2", "GMRF3" and "identity".
`scale`	scale parameter; only used by the "GMRF3" preconditioner.
`chol.control`	options for Cholesky decomposition, see `chol_control`.
`verbose`	whether diagnostic information about the CG sampler is shown.

Value

A list of options used by the conjugate gradients algorithm.

chol_control — *Set options for Cholesky decomposition*

Description

These options are only effective in case the matrix to be decomposed is sparse, i.p. of class `dsCMatrix-class`.

Usage

```
chol_control(perm = NULL, super = NA, ordering = 0L, inplace = TRUE)
```

Arguments

`perm`	logical scalar, see `Cholesky`. If `NULL`, the default, the choice is left to a simple heuristic.
`super`	logical scalar, see `Cholesky`.
`ordering`	an integer scalar passed to CHOLMOD routines determining which reordering schemes are tried to limit sparse Cholesky fill-in.
`inplace`	whether sparse Cholesky updates should re-use the same memory location.

Value

A list with specified options used for Cholesky decomposition.

combine_chains — *Combine multiple mcdraws objects into a single one by combining their chains*

Description

This function can be used to combine the results of parallel simulations.

Usage

```
combine_chains(...)
```

Arguments

`...`	objects of class `mcdraws`.

Value

A combined object of class `mcdraws` where the number of stored chains equals the sum of the numbers of chains in the input objects.

`combine_iters` *Combine multiple mcdraws objects into a single one by combining their draws*

Description

This function is used to combine the results of parallel posterior predictive simulations.

Usage

```
combine_iters(...)
```

Arguments

`...`	objects of class `mcdraws`

Value

A combined object of class `mcdraws` where the number of stored draws equals the sum of the numbers of draws in the input objects.

`computeDesignMatrix` *Compute a list of design matrices for all terms in a model formula, or based on a sampler environment*

Description

If `sampler` is provided instead of `formula`, the design matrices are based on the model used to create the sampler environment. In that case, if `data` is `NULL`, the design matrices stored in `sampler` are returned, otherwise the design matrices are computed for the provided data based on the sampler's model. The output is a list of dense or sparse design matrices for the model components with respect to `data`.

Usage

```
computeDesignMatrix(formula = NULL, data = NULL, labels = TRUE)
```

Arguments

`formula`	model formula.
`data`	data frame to be used in deriving the design matrices.
`labels`	if `TRUE`, column names are assigned.

Value

A list of design matrices.

Examples

```
n <- 1000
dat <- data.frame(
  x = rnorm(n),
  f = factor(sample(1:50, n, replace=TRUE))
)
str(computeDesignMatrix(~ x, dat)[[1]])
model <- ~ reg(~x, name="beta") + gen(~x, factor=~f, name="v")
X <- computeDesignMatrix(model, dat)
str(X)
```

`compute_GMRF_matrices` *Compute (I)GMRF incidence, precision and restriction matrices corresponding to a generic model component*

Description

This function computes incidence, precision and restriction matrices, or a subset thereof, for a Gaussian Markov Random Field (GMRF). A GMRF is specified by a formula passed to the `factor` argument, in the same way as for the `factor` argument of `gen`.

Usage

```
compute_GMRF_matrices(
  factor,
  data,
  D = TRUE,
  Q = TRUE,
  R = TRUE,
  cols2remove = NULL,
  remove.redundant.R.cols = TRUE,
  enclos = .GlobalEnv,
  n.parent = 1L,
  ...
)
```

Arguments

`factor`	factor formula of a generic model component, see `gen`.
`data`	data frame to be used in deriving the matrices.
`D`	if `TRUE` compute the incidence matrix.
`Q`	if `TRUE` compute the precision matrix.
`R`	if `TRUE` compute the restriction matrix.
`cols2remove`	if an integer vector is passed, the dimensions (columns of D, rows and columns of Q and rows of R) that are removed. This can be useful in the case of empty domains.

`remove.redundant.R.cols`	whether to test for and remove redundant restrictions from restriction matrix R
`enclos`	enclosure to look for objects not found in `data`.
`n.parent`	for internal use; in case of custom factor, the number of frames up the calling stack in which to evaluate any custom matrices
`...`	further arguments passed to `economizeMatrix`.

Value

A list containing some or all of the components `D` (incidence matrix), `Q` (precision matrix) and `R` (restriction matrix).

Examples

```
n <- 1000
dat <- data.frame(
  x = rnorm(n),
  f1 = factor(sample(1:50, n, replace=TRUE)),
  f2 = factor(sample(1:10, n, replace=TRUE))
)
mats <- compute_GMRF_matrices(~ f1 * RW1(f2), dat)
str(mats)
```

`correlation` *Correlation factor structures in generic model components*

Description

Element 'factor' of a model component created using function `gen` is a formula composed of several possible terms described below. It is used to derive a (typically sparse) precision matrix for a set of coefficients, and possibly a matrix representing a set of linear constraints to be imposed on the coefficient vector.

iid(f) Independent effects corresponding to the levels of factor `f`.

RW1(f, circular=FALSE, w=NULL) First-order random walk over the levels of factor `f`. The random walk can be made circular and different (fixed) weights can be attached to the innovations. If specified, `w` must be a positive numeric vector of length one less than the number of factor levels. For example, if the levels correspond to different times, it would often be reasonable to choose `w` proportional to the reciprocal time differences. For equidistant times there is generally no need to specify `w`.

RW2(f) Second-order random walk.

AR1(f, phi, w=NULL) First-order autoregressive correlation structure among the levels of `f`. Required argument is the (fixed) autoregressive parameter `phi`. For irregularly spaced AR(1) processes weights can be specified, in the same way as for `RW1`.

season(f, period) Dummy seasonal with period `period`.

spatial(f, poly.df, snap, queen, derive.constraints=FALSE) CAR spatial correlation. Argument `poly.df` can either be an object of (S4) class `SpatialPolygonsDataFrame` or an object of (S3) class `sf`. The latter can be obtained, e.g., from reading in a shape file using function `st_read`. Arguments `snap` and `queen` are passed to `poly2nb`. If `derive.constraints=TRUE` the constraint matrix for an IGMRF model component is formed by computing the singular vectors of the precision matrix.

spline(f, knots, degree) P-splines, i.e. penalized B-splines structure over the domain of a quantitative variable f. Arguments knots and degree are passed to `splineDesign`. If `knots` is a single value it is interpreted as the number of knots, otherwise as a vector of knot positions. By default 40 equally spaced knots are used, and a degree of 3.

custom(f, D=NULL, Q=NULL, R=NULL, derive.constraints=NULL) Either a custom precision or incidence matrix associated with factor f can be passed to argument `Q` or `D`. Optionally a constraint matrix can be supplied as `R`, or constraints can be derived from the null space of the precision matrix by setting `derive.constraints=TRUE`.

Usage

```
iid(name)

RW1(name, circular = FALSE, w = NULL)

RW2(name)

AR1(name, phi, w = NULL)

season(name, period)

spatial(name, poly.df, snap = sqrt(.Machine$double.eps), queen = TRUE)

spline(name, knots, degree)

custom(name, D = NULL, Q = NULL, R = NULL, derive.constraints = NULL)
```

Arguments

`name`	name of a variable, unquoted.
`circular`	whether the random walk is circular.
`w`	a vector of weights.
`phi`	value of an autoregressive parameter.
`period`	a positive integer specifying the seasonal period.
`poly.df`	a spatial data frame.
`snap`	passed to `poly2nb`.
`queen`	passed to `poly2nb`.
`knots`	passed to `splineDesign`.
`degree`	passed to `splineDesign`.

`D`	custom incidence matrix.
`Q`	custom precision matrix.
`R`	custom restriction matrix.
`derive.constraints`	whether to derive the constraint matrix for an IGMRF model component numerically from the precision matrix.

References

B. Allevius (2018). On the precision matrix of an irregularly sampled AR(1) process. arXiv:1801.03791.

H. Rue and L. Held (2005). Gaussian Markov Random Fields. Chapman & Hall/CRC.

Examples

```
# example of CAR spatial random effects
if (requireNamespace("sf")) {
  # 1. load a shape file of counties in North Carolina
  nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
  # 2. generate some data according to a model with a few regression
  # effects, as well as spatial random effects
  gd <- generate_data(
    ~ reg(~ AREA + BIR74, prior=pr_normal(precision=1), name="beta") +
      gen(factor = ~ spatial(NAME, poly.df=nc), name="vs"),
    sigma.mod = pr_invchisq(df=10, scale=1),
    data = nc
  )
  # add the generated target variable and the spatial random effects to the
  # spatial dataframe object
  nc$y <- gd$y
  nc$vs_true <- gd$pars$vs
  # 3. fit a model to the generated data, and see to what extent the
  #    parameters used to generate the data, gd$pars, are reproduced
  sampler <- create_sampler(
    y ~ reg(~ AREA + BIR74, prior=pr_normal(precision=1), name="beta") +
    gen(factor = ~ spatial(NAME, poly.df=nc), name="vs"),
    block=TRUE, data=nc
  )
  sim <- MCMCsim(sampler, store.all=TRUE, n.iter=600, n.chain=2, verbose=FALSE)
  (summ <- summary(sim))
  nc$vs <- summ$vs[, "Mean"]
  plot(nc[c("vs_true", "vs")])
  plot(gd$pars$vs, summ$vs[, "Mean"]); abline(0, 1, col="red")
}
```

`create_sampler` *Create a sampler object*

Description

This function sets up a sampler object, based on the specification of a model. The object contains functions to draw a set of model parameters from their prior and conditional posterior distributions, and to generate starting values for the MCMC simulation. The functions share a common environment containing precomputed quantities such as design matrices based on the model and the data. The sampler object is the main input for the MCMC simulation function `MCMCsim`.

Usage

```
create_sampler(
  formula,
  data = NULL,
  family = "gaussian",
  ny = NULL,
  ry = NULL,
  r.mod,
  sigma.fixed = NULL,
  sigma.mod = NULL,
  Q0 = NULL,
  formula.V = NULL,
  logJacobian = NULL,
  linpred = NULL,
  compute.weights = FALSE,
  block = NULL,
  prior.only = FALSE,
  control = sampler_control()
)
```

Arguments

`formula`	formula to specify the response variable and additive model components. The model components form the linear predictor part of the model. A model component on the right hand side can be either a regression term specified by `reg(...)`, a covariates subject to error term specified by `mec(...)`, or a generic random effect term specified by `gen(...)`. See for details the help pages for these model component creation functions. An offset can be specified as `offset(...)`. Other terms in the formula are collectively interpreted as ordinary regression effects, treated in the same way as a `reg(...)` term, but without the option to change the prior.
`data`	data frame with n rows in which the variables specified in model components can be found.

`family`	character string describing the data distribution. The default is 'gaussian'. Other options are 'binomial', 'multinomial', 'negbinomial' for the negative binomial distribution, 'poisson', and 'gamma'. See mcmcsae-family for the related functions that can be used to specify `family` and associated parameters and controls. For the binomial distribution logistic and probit link functions are supported, the latter only for binary data. For the negative binomial, Poisson and gamma sampling distributions a log link function is assumed. Note that currently `family = 'poisson'` is implemented using the negative binomial distribution with its (reciprocal) overdispersion parameter set to a very large value. For categorical or multinomial data, `family = "multinomial"` can be used. The implementation is based on a stick-breaking representation of the multinomial distribution, and the logistic link function relates each category except the last to a linear predictor. The categories can be referenced in the model specification formula by 'cat_'.
`ny`	in case `family="binomial"` the (vector of) numbers of trials. It can be either a numeric vector or the name of a variable in `data`. Defaults to a vector of 1s.
`ry`	in case `family="negbinomial"` the known, i.e. fixed part of the (reciprocal) dispersion parameter. It can be specified either as a numeric vector or the name of a numeric variable in `data`. The overall dispersion parameter is the product of `ry` with a positive scalar factor modelled as specified by argument `r.mod`. By default `ry` is taken to be 1. For `family = "poisson"` a single value can be specified, determining how well the Poisson distribution is approximated by the negative binomial distribution. The value should be large enough such that the negative binomial's overdispersion becomes negligible, but not too large as this might result in slow MCMC mixing. The default is `ry=100` in this case.
`r.mod`	prior specification for a scalar (reciprocal) dispersion parameter of the negative binomial distribution. The prior can be specified by a call to a prior specification function. Currently `pr_invchisq`, `pr_gig` and `pr_fixed` are supported. The default is a chi-squared prior with 1 degree of freedom. To set the overall dispersion parameter to the value(s) specified by `ry`, use `r.mod = pr_fixed(value=1)`.
`sigma.fixed`	for Gaussian models, if `TRUE` the residual standard deviation parameter 'sigma_' is fixed at 1. In that case argument `sigma.mod` is ignored. This is convenient for Fay-Herriot type models with (sampling) variances assumed to be known. Default is `FALSE`.
`sigma.mod`	prior for the variance parameter of a gaussian sampling distribution. This can be specified by a call to one of the prior specification functions `pr_invchisq`, `pr_exp`, `pr_gig` or `pr_fixed` for inverse chi-squared, exponential, generalized inverse gaussian or degenerate prior distribution, respectively. The default is an improper prior `pr_invchisq(df=0, scale=1)`. A half-t prior on the standard deviation can be specified using `pr_invchisq` with a chi-squared distributed scale parameter.
`Q0`	n x n data-level precision matrix for a Gaussian model. It defaults to the unit matrix. If an n-vector is provided it will be expanded to a (sparse) diagonal matrix with Q0 on its diagonal. If a name is supplied it will be looked up in `data` and subsequently expanded to a diagonal matrix.

`formula.V`	a formula specifying the terms of a variance model in the case of a Gaussian likelihood. Currently two types of terms are supported: a regression term for the log-variance specified with `vreg(...)`, and a term `vfac(...)` for multiplicative modeled factors at a certain level specified by a factor variable. By using unit-level inverse-chi-squared factors the marginal sampling distribution becomes a Student-t distribution, and by using unit-level exponential factors it becomes a Laplace or double exponential distribution.
`logJacobian`	if the data are transformed the logarithm of the Jacobian can be supplied so that it is incorporated in all log-likelihood computations. This can be useful for comparing information criteria for different transformations. It should be supplied as a vector of the same size as the response variable, and is currently only supported if `family="gaussian"`. For example, when a log-transformation is used on response vector `y`, the vector `-log(y)` should be supplied.
`linpred`	a list of matrices defining (possibly out-of-sample) linear predictors to be simulated. This allows inference on e.g. (sub)population totals or means. The list must be of the form `list(name_1=X_1, ...)` where the names refer to the model component names and predictions are computed by summing `X_i %*% p[[name_i]]`. Alternatively, `linpred="fitted"` can be used as a short-cut for simulations of the full in-sample linear predictor.
`compute.weights`	if `TRUE` weights are computed for each element of `linpred`. Note that for a large dataset in combination with vector-valued linear predictors the weights can take up a lot of memory. By default only means are stored in the simulation carried out using `MCMCsim`.
`block`	DEPRECATED, please use argument `control` instead, see also `sampler_control`. Note that this parameter is now by default set to `TRUE`.
`prior.only`	whether a sampler is set up only for sampling from the prior or for sampling from both prior and posterior distributions. Default `FALSE`. If `TRUE` there is no need to specify a response in `formula`. This is used by `generate_data`, which samples from the prior predictive distribution.
`control`	a list with further computational options. These options can be specified using function `sampler_control`.

Details

The right hand side of the `formula` argument to `create_sampler` can be used to specify additive model components. Currently four model components are supported: `reg(...)` for regression or 'fixed' effects, `gen(...)` for generic random effects, `mec(...)` for measurement in covariates effects, and `brt(...)` for a Bayesian additive regression trees component. Note that an offset can be added separately, in the usual way using `offset(...)`.

For gaussian models, `formula.V` can be used to specify the variance structure of the model. Currently two specialized variance model components are supported, `vreg(...)` for regression effects predicting the log-variance and `vfac(...)` for modeled variance factors.

Value

A sampler object, which is the main input for the MCMC simulation function `MCMCsim`. The sampler object is an environment with precomputed quantities and functions. The main functions are rprior, which returns a sample from the prior distributions, draw, which returns a sample from the full conditional posterior distributions, and start, which returns a list with starting values for the Gibbs sampler. If prior.only is TRUE, functions draw and start are not created.

References

J.H. Albert and S. Chib (1993). Bayesian analysis of binary and polychotomous response data. Journal of the American statistical Association 88(422), 669-679.

D. Bates, M. Maechler, B. Bolker and S.C. Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software 67(1), 1-48.

S.W. Linderman, M.J. Johnson and R.P. Adams (2015). Dependent multinomial models made easy: Stick-breaking with the Polya-Gamma augmentation. Advances in Neural Information Processing Systems, 3456-3464.

P.A. Parker, S.H. Holan and R. Janicki (2023). Conjugate Modeling Approaches for Small Area Estimation with Heteroscedastic Structure. Journal of Survey Statistics and Methodology, smad002.

N. Polson, J.G. Scott and J. Windle (2013). Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. Journal of the American Statistical Association 108(504), 1339-1349.

H. Rue and L. Held (2005). Gaussian Markov Random Fields. Chapman & Hall/CRC.

M. Zhou and L. Carin (2015). Negative Binomial Process Count and Mixture Modeling. IEEE Transactions on Pattern Analysis and Machine Intelligence 37(2), 307-320.

Examples

```
# first generate some data
n <- 200
x <- rnorm(n)
y <- 0.5 + 2*x + 0.3*rnorm(n)
# create a sampler for a simple linear regression model
sampler <- create_sampler(y ~ x)
sim <- MCMCsim(sampler)
(summary(sim))

y <- rbinom(n, 1, 1 / (1 + exp(-(0.5 + 2*x))))
# create a sampler for a binary logistic regression model
sampler <- create_sampler(y ~ x, family="binomial")
sim <- MCMCsim(sampler)
(summary(sim))
```

---

create_TMVN_sampler *Set up a sampler object for sampling from a possibly truncated and degenerate multivariate normal distribution*

Description

This function sets up an object for multivariate normal sampling based on a specified precision matrix. Linear equality and inequality restrictions are supported. For sampling under inequality restrictions four algorithms are available. The default in that case is an exact Hamiltonian Monte Carlo algorithm (Pakman and Paninski, 2014). A related algorithm is the zig-zag Hamiltonian Monte Carlo method (Nishimura et al., 2021) in which momentum is sampled from a Laplace instead of normal distribution. Alternatively, a Gibbs sampling algorithm can be used (Rodriguez-Yam et al., 2004). The fourth option is a data augmentation method that samples from a smooth approximation to the truncated multivariate normal distribution (Souris et al., 2018).

Usage

```
create_TMVN_sampler(
  Q,
  mu = NULL,
  Xy = NULL,
  update.Q = FALSE,
  update.mu = update.Q,
  name = "x",
  coef.names = NULL,
  R = NULL,
  r = NULL,
  S = NULL,
  s = NULL,
  lower = NULL,
  upper = NULL,
  check.constraints = FALSE,
  method = NULL,
  reduce = NULL,
  chol.control = chol_control()
)
```

Arguments

`Q`	precision matrix of the (unconstrained) multivariate normal distribution.
`mu`	mean of the (unconstrained) multivariate normal distribution.
`Xy`	alternative to specifying mu; in this case `mu` is computed as Q^{-1}`Xy`.
`update.Q`	whether `Q` is updated for each draw.
`update.mu`	whether `mu` is updated for each draw. By default equal to `update.Q`.
`name`	name of the TMVN vector parameter.
`coef.names`	optional labels for the components of the vector parameter.
`R`	equality restriction matrix.
`r`	rhs vector for equality constraints $R'x = r$, where R' denotes the transpose of R.
`S`	inequality restriction matrix.

`s`	rhs vector for inequality constraints $S'x >= s$, where S' denotes the transpose of S.
`lower`	alternative to `s` for two-sided inequality restrictions `lower` $<= S'x <=$ `upper`.
`upper`	alternative to `s` for two-sided inequality restrictions `lower` $<= S'x <=$ `upper`.
`check.constraints`	if `TRUE` check whether the starting values satisfy all constraints.
`method`	sampling method. The options are "direct" for direct sampling from the unconstrained or equality constrained multivariate normal (MVN). For inequality constrained MVN sampling three methods are supported: "HMC" for (exact) Hamiltonian Monte Carlo, "HMCZigZag" for (exact) Hamiltonian Monte Carlo with Laplace momentum, "Gibbs" for a component-wise Gibbs sampling approach, and "softTMVN" for a data augmentation method that samples from a smooth approximation to the truncated MVN. Alternatively, the method setting functions `m_direct`, `m_HMC`, `m_HMC_ZigZag`, `m_Gibbs` or `m_softTMVN` can be used to select the method and possibly set some of its options to non-default values, see `TMVN-methods`.
`reduce`	whether to a priori restrict the simulation to the subspace defined by the equality constraints.
`chol.control`	options for Cholesky decomposition, see `chol_control`.

Details

The componentwise Gibbs sampler uses univariate truncated normal samplers as described in Botev and L'Ecuyer (2016). These samplers are implemented in R package **TruncatedNormal**, but here translated to C++ for an additional speed-up.

Value

An environment for sampling from a possibly degenerate and truncated multivariate normal distribution.

Author(s)

Harm Jan Boonstra, with help from Grzegorz Baltissen

References

Z.I. Botev and P. L'Ecuyer (2016). Simulation from the Normal Distribution Truncated to an Interval in the Tail. in VALUETOOLS.

Y. Cong, B. Chen and M. Zhou (2017). Fast simulation of hyperplane-truncated multivariate normal distributions. Bayesian Analysis 12(4), 1017-1037.

Y. Li and S.K. Ghosh (2015). Efficient sampling methods for truncated multivariate normal and student-t distributions subject to linear inequality constraints. Journal of Statistical Theory and Practice 9(4), 712-732.

A. Nishimura, Z. Zhang and M.A. Suchard (2021). Hamiltonian zigzag sampler got more momentum than its Markovian counterpart: Equivalence of two zigzags under a momentum refreshment limit. arXiv:2104.07694.

A. Pakman and L. Paninski (2014). Exact Hamiltonian Monte Carlo for truncated multivariate gaussians. Journal of Computational and Graphical Statistics 23(2), 518-542.

G. Rodriguez-Yam, R.A. Davis and L.L. Scharf (2004). Efficient Gibbs sampling of truncated multivariate normal with application to constrained linear regression. Unpublished manuscript.

H. Rue and L. Held (2005). Gaussian Markov Random Fields. Chapman & Hall/CRC.

A. Souris, A. Bhattacharya and P. Debdeep (2018). The Soft Multivariate Truncated Normal Distribution. arXiv:1807.09155.

K.A. Valeriano, C.E. Galarza and L.A. Matos (2023). Moments and random number generation for the truncated elliptical family of distributions. Statistics and Computing 33(1), 1-20.

Examples

```
S <- cbind(diag(2), c(-1, 1), c(1.1, -1))  # inequality matrix
# S'x >= 0 represents the wedge x1 <= x2 <= 1.1 x1
# example taken from Pakman and Paninski (2014)
# 1. exact Hamiltonian Monte Carlo (Pakman and Paninski, 2014)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), S=S, method="HMC")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 2. exact Hamiltonian Monte Carlo with Laplace momentum (Nishimura et al., 2021)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), S=S, method="HMCZigZag")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 3. Gibbs sampling approach (Rodriguez-Yam et al., 2004)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), S=S, method="Gibbs")
sim <- MCMCsim(sampler, burnin=500, n.iter=2000, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 4. soft TMVN approximation (Souris et al., 2018)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), S=S, method="softTMVN")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
```

---

gen *Create a model component object for a generic random effects component in the linear predictor*

Description

This function is intended to be used on the right hand side of the `formula` argument to `create_sampler` or `generate_data`.

Usage

```
gen(
  formula = ~1,
  factor = NULL,
  remove.redundant = FALSE,
  drop.empty.levels = FALSE,
  X = NULL,
  var = NULL,
  prior = NULL,
  Q0 = NULL,
  PX = NULL,
  GMRFmats = NULL,
  priorA = NULL,
  Leroux = FALSE,
  R0 = NULL,
  RA = NULL,
  constr = NULL,
  S0 = NULL,
  SA = NULL,
  formula.gl = NULL,
  a = 1000,
  name = "",
  sparse = NULL,
  control = gen_control(),
  debug = FALSE
)
```

Arguments

`formula`	a model formula specifying the effects that vary over the levels of the factor variable(s) specified by argument `factor`. Defaults to `~1`, corresponding to random intercepts. If `X` is specified `formula` is ignored. Variable names are looked up in the data frame passed as `data` argument to `create_sampler` or `generate_data`, or in `environment(formula)`.
`factor`	a formula with factors by which the effects specified in the `formula` argument vary. Often only one such factor is needed but multiple factors are allowed so that interaction terms can be modeled conveniently. The formula must take the form ~ `f1(fac1, ...) * f2(fac2, ...) ...`, where `fac1`, `fac2` are factor variables and `f1`, `f2` determine the correlation structure assumed between levels of each factor, and the `...` indicate that for some correlation types further arguments can be passed. Correlation structures currently supported include `iid` for independent identically distributed effects, `RW1` and `RW2` for random walks of first or second order over the factor levels, `AR1` for first-order autoregressive effects, `season` for seasonal effects, `spatial` for spatial (CAR) effects and `custom` for supplying a custom precision matrix corresponding to the levels of the factor. For further details about the correlation structures, and further arguments that can be passed, see `correlation`. Argument `factor` is ignored if `X` is specified. The factor variables are looked up in the data frame passed as `data` argument to

create_sampler or generate_data, or in environment(formula).

remove.redundant
: whether redundant columns should be removed from the model matrix associated with formula. Default is FALSE.

drop.empty.levels
: whether to remove factor levels without observations.

X
: A (possibly sparse) design matrix. This can be used instead of formula and factor.

var
: the (co)variance structure among the varying effects defined by formula over the levels of the factors defined by factor. The default is "unstructured", meaning that a full covariance matrix parameterization is used. For uncorrelated effects with unequal variances use var="diagonal". For uncorrelated effects with equal variances use var="scalar". In the case of a single varying effect there is no difference between these choices.

prior
: the prior specification for the variance parameters of the random effects. These can currently be specified by a call to pr_invwishart in case var="unstructured" or by a call to pr_invchisq otherwise. See the documentation of those prior specification functions for more details.

Q0
: precision matrix associated with formula. This can only be used in combination with var="scalar".

PX
: whether parameter expansion should be used. Default is TRUE, which applies parameter expansion with default options. The only exception is that for gamma sampling distributions the default is FALSE, i.e. no parameter expansion. Alternative options can be specified by supplying a list with one or more of the following components:

  prior prior for the multiplicative expansion parameter. Defaults to a normal prior with mean 0 and standard deviation 1, unless the sampling distribution is gamma in which case the default is a Multivariate Log inverse Gamma prior. The default parameters can be changed using functions pr_normal or pr_MLiG.

  vector whether a redundant multiplicative expansion parameter is used for each varying effect specified by formula. The default is TRUE except when var="scalar". If FALSE a single redundant multiplicative parameter is used.

  data.scale whether the data level scale is used as a variance factor for the expansion parameters. Default is TRUE.

GMRFmats
: list of incidence/precision/constraint matrices. This can be specified as an alternative to factor. It should be a list such as that returned by compute_GMRF_matrices. Can be used together with argument X as a flexible alternative to formula and factor.

priorA
: prior distribution for scale factors at the variance scale associated with QA. In case of IGMRF models the scale factors correspond to the innovations. The default NULL means not to use any local scale factors. A prior can currently be specified using pr_invchisq or pr_exp.

`Leroux`	this option alters the precision matrix determined by `factor` by taking a weighted average of it with the identity matrix. If `TRUE` the model gains an additional parameter, the 'Leroux' parameter, being the weight of the original, structured, precision matrix in the weighted average. By default a uniform prior for the weight and a uniform Metropolis-Hastings proposal density are employed. This default can be changed by supplying a list with elements a, b, and a.star, b.star, implying a beta(a, b) prior and a beta(a.star, b.star) independence proposal density. A third option is to supply a single number between 0 and 1, which is then used as a fixed value for the Leroux parameter.
`R0`	an optional equality restriction matrix acting on the coefficients defined by `formula`, for each level defined by `factor`. If c is the number of restrictions, `R0` is a q0 x c matrix where q0 is the number of columns of the design matrix derived from `formula`. Together with `RA` it defines the set of equality constraints to be imposed on the vector of coefficients. Only allowed in combination with `var="scalar"`.
`RA`	an optional equality restriction matrix acting on the coefficients defined by `factor`, for each effect defined by `formula`. If c is the number of restrictions, `RA` is a l x c matrix where l is the number of levels defined by `factor`. Together with `R0` this defines the set of equality constraints to be imposed on the vector of coefficients. If `constr=TRUE`, additional constraints are imposed, corresponding to the null-vectors of the singular precision matrix in case of an intrinsic Gaussian Markov Random Field.
`constr`	whether constraints corresponding to the null-vectors of the precision matrix are to be imposed on the vector of coefficients. By default this is `TRUE` for improper or intrinsic GMRF model components, i.e. components with a singular precision matrix such as random walks or CAR spatial components.
`S0`	an optional inequality restriction matrix acting on the coefficients defined by `formula`, for each level defined by `factor`. If c is the number of restrictions, `S0` is a q0 x c matrix where q0 is the number of columns of the design matrix derived from `formula`. Together with `SA` it defines the set of inequality constraints to be imposed on the vector of coefficients.
`SA`	an optional inequality restriction matrix acting on the coefficients defined by `factor`, for each effect defined by `formula`. If c is the number of restrictions, `SA` is a l x c matrix where l is the number of levels defined by `factor`. Together with `S0` this defines the set of constraints to be imposed on the vector of coefficients.
`formula.gl`	a formula of the form ~ `glreg(...)` for group-level predictors around which the random effect component is hierarchically centered. See `glreg` for details.
`a`	only used in case the effects are MLiG distributed, such as is assumed in case of a gamma sampling distribution, or for gaussian variance modelling. In those cases a controls how close the effects' prior is to a normal prior, see `pr_MLiG`.
`name`	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'gen' with the number of the model term attached.
`sparse`	whether the model matrix associated with `formula` should be sparse. The default is based on a simple heuristic based on storage size.

control	a list with further computational options. These options can be specified using function `gen_control`.
debug	if `TRUE` a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

References

J. Besag and C. Kooperberg (1995). On Conditional and Intrinsic Autoregression. Biometrika 82(4), 733-746.

C.M. Carvalho, N.G. Polson and J.G. Scott (2010). The horseshoe estimator for sparse signals. Biometrika 97(2), 465-480.

L. Fahrmeir, T. Kneib and S. Lang (2004). Penalized Structured Additive Regression for Space-Time Data: a Bayesian Perspective. Statistica Sinica 14, 731-761.

A. Gelman (2006). Prior distributions for variance parameters in hierarchical models. Bayesian Analysis 1(3), 515-533.

A. Gelman, D.A. Van Dyk, Z. Huang and W.J. Boscardin (2008). Using Redundant Parameterizations to Fit Hierarchical Models. Journal of Computational and Graphical Statistics 17(1), 95-122.

B. Leroux, X. Lei and N. Breslow (1999). Estimation of Disease Rates in Small Areas: A New Mixed Model for Spatial Dependence. In M. Halloran and D. Berry (Eds.), Statistical Models in Epidemiology, the Environment and Clinical Trials, 135-178.

T. Park and G. Casella (2008). The Bayesian Lasso. Journal of the American Statistical Association 103(482), 681-686.

H. Rue and L. Held (2005). Gaussian Markov Random Fields. Chapman & Hall/CRC.

`generate_data` *Generate a data vector according to a model*

Description

This function generates draws from the prior predictive distribution. Parameter values are drawn from their priors, and consequently data is generated from the sampling distribution given these parameter values.

Usage

```
generate_data(
  formula,
  data = NULL,
  family = "gaussian",
  ny = NULL,
  ry = NULL,
  r.mod,
  sigma.fixed = NULL,
  sigma.mod = NULL,
  Q0 = NULL,
  formula.V = NULL,
  linpred = NULL
)
```

Arguments

`formula`	A model formula, see `create_sampler`. Any left-hand-side of the formula is ignored.
`data`	see `create_sampler`.
`family`	sampling distribution family, see `create_sampler`.
`ny`	see `create_sampler`.
`ry`	see `create_sampler`.
`r.mod`	see `create_sampler`.
`sigma.fixed`	see `create_sampler`.
`sigma.mod`	see `create_sampler`.
`Q0`	see `create_sampler`.
`formula.V`	see `create_sampler`.
`linpred`	see `create_sampler`.

Value

A list with a generated data vector and a list of prior means of the parameters. The parameters are drawn from their priors.

Examples

```
n <- 250
dat <- data.frame(
  x = rnorm(n),
  g = factor(sample(1:10, n, replace=TRUE)),
  ny = 10
)
gd <- generate_data(
  ~ reg(~ 1 + x, Q0=10, b0=c(0, 1), name="beta") + gen(factor = ~ g, name="v"),
  family="binomial", ny="ny", data=dat
)
gd
plot(dat$x, gd$y)
```

`gen_control` *Set computational options for the sampling algorithms used for a 'gen' model component*

Description

Set computational options for the sampling algorithms used for a 'gen' model component

Usage

```
gen_control(MHprop = c("GiG", "LNRW"))
```

Arguments

`MHprop`	MH proposal for the variance component in case of a MLiG prior on the coefficients. The two options are "GiG" for a generalized inverse gamma proposal, and "LNRW" for a log_normal random walk proposal. The former should approximate the conditional posterior quite well provided MLiG parameter `a` is large, such that the coefficients' prior is approximately normal.

Value

A list with computational options regarding a 'gen' model component.

`get_draw` *Extract a list of parameter values for a single draw*

Description

Extract a list of parameter values for a single draw

Usage

```
get_draw(obj, iter, chain)
```

Arguments

`obj`	an object of class `mcdraws`.
`iter`	iteration number.
`chain`	chain number.

Value

A list with all parameter values of draw `iter` from chain `chain`.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4, store.all=TRUE)
get_draw(sim, iter=20, chain=3)
```

---

glreg — *Create a model object for group-level regression effects within a generic random effects component.*

Description

This function is intended to be used to specify the `formula.gl` argument to the `gen` model component specification function. Group-level predictors and hierarchical centering are not used by default, and they currently cannot be used in a model component that is sampled together with another model component in the same Gibbs block.

Usage

```
glreg(
  formula = NULL,
  remove.redundant = FALSE,
  prior = NULL,
  Q0 = NULL,
  data = NULL,
  name = ""
)
```

Arguments

`formula`	a formula specifying the group-level predictors to be used within a model component. If no `data` is supplied the group-level predictors are derived as group-level means from the unit-level data passed as `data` argument to `create_sampler` or `generate_data`.
`remove.redundant`	whether redundant columns should be removed from the design matrix. Default is `FALSE`.
`prior`	prior specification for the group-level effects. Currently only normal priors with mean 0 can be specified, using function `pr_normal`.
`Q0`	prior precision matrix for the group-level effects. The default is a zero matrix corresponding to a noninformative improper prior. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(precision = Q0.value)`.

`data`	group-level data frame in which the group-level variables specified in `formula` are looked up.
`name`	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default this name will be the name of the corresponding generic random effects component appended by '_gl'.

Value

An object with precomputed quantities for sampling from prior or conditional posterior distributions for this model component. Only intended for internal use by other package functions.

`labels` *Get and set the variable labels of a draws component object for a vector-valued parameter*

Description

Get and set the variable labels of a draws component object for a vector-valued parameter

Usage

```
## S3 method for class 'dc'
labels(object, ...)

labels(object) <- value
```

Arguments

`object`	a draws component object.
`...`	currently not used.
`value`	a vector of labels.

Value

The extractor function returns the variable labels.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=50, n.iter=100, n.chain=1, store.all=TRUE)
labels(sim$beta)
labels(sim$v)
labels(sim$beta) <- c("a", "b")
labels(sim$beta)
```

`matrix-vector` *Fast matrix-vector multiplications*

Description

Functions for matrix-vector multiplies like `%*%` and `crossprod`, but often faster for the matrix types supported. The return value is always a numeric vector.

Usage

```
M %m*v% v

crossprod_mv(M, v)
```

Arguments

`M`	a matrix of class 'matrix', 'dgCMatrix', 'dsCMatrix', 'tabMatrix', or 'ddiMatrix'.
`v`	a numeric vector.

Value

For `%m*v%` the vector Mv and for `crossprod_mv` the vector $M'v$ where M' denotes the transpose of M.

Examples

```
M <- matrix(rnorm(10*10), 10, 10)
x <- rnorm(10)
M %m*v% x
crossprod_mv(M, x)
M <- Matrix::rsparsematrix(100, 100, nnz=100)
x <- rnorm(100)
M %m*v% x
crossprod_mv(M, x)
```

`maximize_log_lh_p` *Maximize the log-likelihood or log-posterior as defined by a sampler closure*

Description

Maximize the log-likelihood or log-posterior as defined by a sampler closure

Usage

```
maximize_log_lh_p(
  sampler,
  type = c("llh", "lpost"),
  method = "BFGS",
  control = list(fnscale = -1),
  ...
)
```

Arguments

`sampler`	sampler function closure, i.e. the return value of a call to `create_sampler`.
`type`	either "llh" (default) or "lpost", for optimization of the log-likelihood, or the log-posterior, respectively.
`method`	optimization method, passed to `optim`.
`control`	control parameters, passed to `optim`.
`...`	other parameters passed to `optim`.

Value

A list of parameter values that, provided the optimization was successful, maximize the (log-)likelihood or (log-)posterior.

Examples

```
n <- 1000
dat <- data.frame(
  x = rnorm(n),
  f = factor(sample(1:50, n, replace=TRUE))
)
df <- generate_data(
  ~ reg(~x, name="beta", prior=pr_normal(precision=1)) + gen(~x, factor=~f, name="v"),
  sigma.fixed=TRUE, data=dat
)
dat$y <- df$y
sampler <- create_sampler(y ~ x + gen(~x, factor=~f, name="v"), data=dat)
opt <- maximize_log_lh_p(sampler)
str(opt)
plot(df$par$v, opt$par$v); abline(0, 1, col="red")
```

MCMC-diagnostics *Compute MCMC diagnostic measures*

Description

R_hat computes Gelman-Rubin convergence diagnostics based on the MCMC output in a model component, and n_eff computes the effective sample sizes, .i.e. estimates for the number of independent samples from the posterior distribution.

Usage

```
R_hat(dc)

n_eff(dc, useFFT = TRUE, lag.max, cl = NULL)
```

Arguments

dc	a draws component (dc) object corresponding to a model parameter.
useFFT	whether to use the Fast Fourier Transform algorithm. Default is TRUE as this is typically faster.
lag.max	the lag up to which autocorrelations are computed in case useFFT=FALSE.
cl	a cluster for parallel computation.

Value

In case of R_hat the split-R-hat convergence diagnostic for each component of the vector parameter, and in case of n_eff the effective number of independent samples for each component of the vector parameter.

References

A. Gelman and D. B. Rubin (1992). Inference from Iterative Simulation Using Multiple Sequences. Statistical Science 7, 457-511.

A. Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari and D.B. Rubin (2013). Bayesian Data Analysis, 3rd edition. Chapman & Hall/CRC.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4, store.all=TRUE)
n_eff(sim$beta)
n_eff(sim$v_sigma)
n_eff(sim$v_rho)
R_hat(sim$beta)
R_hat(sim$llh_)
R_hat(sim$v_sigma)
```

MCMC-object-conversion

Convert a draws component object to another format

Description

Use `to_mcmc` to convert a draws component to class `mcmc.list`, allowing one to use MCMC diagnostic functions provided by package **coda**. Use `as.array` to convert to an array of dimension `(draws, chains, parameters)`. The array format is supported by some packages for analysis or visualisation of MCMC simulation results, e.g. **bayesplot**. Use `as.matrix` to convert to a matrix, concatenating the chains. Finally, use `to_draws_array` to convert either a draws component or (a subset of components of) an mcdraws object to a `draws_array` object as defined in package **posterior**.

Usage

```
to_mcmc(x)

to_draws_array(x, components = NULL)

## S3 method for class 'dc'
as.array(x, ...)

## S3 method for class 'dc'
as.matrix(x, colnames = TRUE, ...)
```

Arguments

`x`	a component of an mcdraws object corresponding to a scalar or vector model parameter.
`components`	optional character vector of names of draws components in an mcdraws object. This can be used to select a subset of components to convert to `draws_array` format.
`...`	currently ignored.
`colnames`	whether column names should be set.

Value

The draws component(s) coerced to an `mcmc.list` object, a `draws_array` object, an array, or a matrix.

Examples

```
data(iris)
sampler <- create_sampler(Sepal.Length ~ reg(~ Petal.Length + Species, name="beta"), data=iris)
sim <- MCMCsim(sampler, burnin=100, n.chain=2, n.iter=400)
summary(sim)
if (require("coda", quietly=TRUE)) {
  mcbeta <- to_mcmc(sim$beta)
  geweke.diag(mcbeta)
}
if (require("posterior", quietly=TRUE)) {
  mcbeta <- to_draws_array(sim$beta)
  mcbeta
  draws <- to_draws_array(sim)
  str(draws)
}
str(as.array(sim$beta))
str(as.matrix(sim$beta))

# generate some example data
n <- 250
dat <- data.frame(x=runif(n), f=as.factor(sample(1:5, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, prior=pr_normal(precision=1), name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
sim <- MCMCsim(sampler, n.chain=2, n.iter=400)
str(sim$beta)
str(as.array(sim$beta))
bayesplot::mcmc_hist(as.array(sim$beta))
bayesplot::mcmc_dens_overlay(as.array(sim$beta))
# fake data simulation check:
bayesplot::mcmc_recover_intervals(as.array(sim$beta), gd$pars$beta)
bayesplot::mcmc_recover_hist(as.array(sim$beta), gd$pars$beta)

ex <- mcmcsae_example()
plot(ex$dat$fT, ex$dat$y)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, n.chain=2, n.iter=400, store.all=TRUE)
str(sim$beta)
str(as.matrix(sim$beta))
# fake data simulation check:
bayesplot::mcmc_recover_intervals(as.matrix(sim$beta), ex$pars$beta)
bayesplot::mcmc_recover_intervals(as.matrix(sim$u), ex$pars$u)
```

`mcmcsae-family` *Functions for specifying a sampling distribution and link function*

Description

These functions are intended for use in the `family` argument of `create_sampler`. In future versions these functions may gain additional arguments, but currently the corresponding functions `gaussian` and `binomial` can be used as well.

Usage

```
f_gaussian(link = "identity")

f_binomial(link = c("logit", "probit"))

f_negbinomial(link = "logit")

f_poisson(link = "log")

f_multinomial(link = "logit", K = NULL)

f_gamma(
  link = "log",
  shape.vec = ~1,
  shape.prior = pr_gamma(0.1, 0.1),
  shape.MH.type = c("RW", "gamma")
)

f_gaussian_gamma(link = "identity", var.data, ...)
```

Arguments

`link`	the name of a link function. Currently the only allowed link functions are: `"identity"` for (log-)Gaussian sampling distributions, `"logit"` (default) and `"probit"` for binomial distributions and `"log"` for negative binomial sampling distributions.
`K`	number of categories for multinomial model; this must be specified for prior predictive sampling.
`shape.vec`	optional formula specification of unequal shape parameter for gamma family
`shape.prior`	prior for gamma shape parameter. Supported prior distributions: `pr_fixed` with a default value of 1, `pr_exp` and `pr_gamma`. The current default is that of a fixed shape equal to 1, i.e. `pr_fixed(value=1)`.
`shape.MH.type`	the type of Metropolis-Hastings algorithm employed in case the shape parameter is to be inferred. The two choices currently supported are "RW" for a random walk proposal on the log-shape scale and "gamma" for an approximating gamma proposal, found using an iterative algorithm. In the latter case, a Metropolis-Hastings accept-reject step is currently omitted, so the sampling algorithm is an approximate one, though one that is usually quite accurate and efficient.
`var.data`	the (variance) data for the gamma part of family `gaussian_gamma`.
`...`	further arguments passed to `f_gamma`.

Value

A family object.

References

J.W. Miller (2019). Fast and Accurate Approximation of the Full Conditional for Gamma Shape Parameters. Journal of Computational and Graphical Statistics 28(2), 476-480.

`mcmcsae_example` *Generate artificial data according to an additive spatio-temporal model*

Description

This function is used to generate data for several examples.

Usage

```
mcmcsae_example(n = 100L, family = "gaussian")
```

Arguments

`n`	the size of the generated dataset.
`family`	sampling distribution family, see `create_sampler`.

Value

A `list` containing the generated dataset, the values of the model parameters, and the model specification as a formula.

Examples

```
ex <- mcmcsae_example()
str(ex)
```

MCMCsim *Run a Markov Chain Monte Carlo simulation*

Description

Given a sampler object this function runs a MCMC simulation and stores the posterior draws. A sampler object for a wide class of multilevel models can be created using `create_sampler`, but users can also define their own sampler functions, see below. `MCMCsim` allows to choose the parameters for which simulation results must be stored. It is possible to define derived quantities that will also be stored. To save memory, it is also possible to only store Monte Carlo means/standard errors for some large vector parameters, say. Another way to use less memory is to save the simulation results of large vector parameters to file. For parameters specified in `plot.trace` trace plots or pair plots of multiple parameters are displayed during the simulation.

Usage

```
MCMCsim(
  sampler,
  from.prior = FALSE,
  n.iter = 1000L,
  n.chain = 3L,
  thin = 1L,
  burnin = if (from.prior) 0L else 250L,
  start = NULL,
  store,
  store.all = FALSE,
  pred = NULL,
  store.mean,
  store.sds = FALSE,
  to.file = NULL,
  filename = "MCdraws_",
  write.single.prec = FALSE,
  verbose = TRUE,
  n.progress = n.iter%/%10L,
  trace.convergence = NULL,
  stop.on.convergence = FALSE,
  convergence.bound = 1.05,
  plot.trace = NULL,
  add.to.plot = TRUE,
  plot.type = "l",
  n.cores = 1L,
  cl = NULL,
  seed = NULL,
  export = NULL
)
```

Arguments

`sampler`	sampler object created by `create_sampler`.
`from.prior`	whether to sample from the prior. By default `from.prior=FALSE` and samples are taken from the posterior.
`n.iter`	number of draws after burnin.
`n.chain`	number of independent chains.
`thin`	only every `thin`'th draw is kept.
`burnin`	number of draws to discard at the beginning of each chain.
`start`	an optional function to generate starting values or a list containing for each chain a named list of starting values. It may be used to provide starting values for some or all parameters. The sampler object's own start function, if it exists, is called to generate any starting values not provided by the user.
`store`	vector of names of parameters to store MCMC draws for. By default, simulations are stored for all parameters returned by `sampler$store_default`.
`store.all`	if `TRUE` simulation vectors of all parameters returned by the sampling function of `sampler` will be stored. The default is `FALSE`, and in that case only simulations for the parameters named in `store` are stored.
`pred`	list of character strings defining derived quantities to be computed (and stored) for each draw.
`store.mean`	vector of names of parameters for which only the mean (per chain) is to be stored. This may be useful for large vector parameters (e.g. regression residuals) for which storing complete MCMC output would use too much memory. The function `sampler$store_mean_default` exists it provides the default.
`store.sds`	if `TRUE` store for all parameters in `store.mean`, besides the mean, also the standard deviation. Default is `FALSE`.
`to.file`	vector of names of parameters to write to file.
`filename`	name of file to write parameter draws to. Each named parameter is written to a separate file, named `filename_parametername`.
`write.single.prec`	Whether to write to file in single precision. Default is `FALSE`.
`verbose`	if `FALSE` no output is sent to the screen during the simulation. `TRUE` by default.
`n.progress`	update diagnostics and plots after so many iterations.
`trace.convergence`	vector of names of parameters for which Gelman-Rubin R-hat diagnostics are printed to the screen every `n.progress` iterations.
`stop.on.convergence`	if `TRUE` stop the simulation if the R-hat diagnostics for all parameters in `trace.convergence` are less than `convergence.bound`.
`convergence.bound`	threshold used with `stop.on.convergence`.
`plot.trace`	character vector of parameter names for which to plot draws during the simulation. For one or two parameters trace plots will be shown, and if more parameters are specified the results will be displayed in a pairs plot. For vector parameters a specific component can be selected using brackets, e.g. `"beta[2]"`.

`add.to.plot`	if `TRUE` the plot is updated every `n.progress` iterations, otherwise a new plot (with new scales) is created after every `n.progress` iterations.
`plot.type`	default is "l" (lines).
`n.cores`	the number of cpu cores to use. Default is 1, i.e. no parallel computation. If an existing cluster `cl` is provided, `n.cores` will be set to the number of workers in that cluster.
`cl`	an existing cluster can be passed for parallel computation. If `NULL` and `n.cores > 1`, a new cluster is created.
`seed`	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
`export`	a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in `pred` depend on global variables.

Details

A sampler object is an environment containing data and functions to use for sampling. The following elements of the sampler object are used by `MCMCsim`:

start function to generate starting values.

draw function to draw samples, typically from a full conditional posterior distribution.

rprior function to draw from a prior distribution.

coef.names list of vectors of parameter coefficient names, for vector parameters.

MHpars vector of names of parameters that are sampled using a Metropolis-Hastings (MH) sampler; acceptance rates are kept for these parameters.

adapt function of acceptance rates of `MHpars` to adapt MH-kernel, called every 100 iterations during the burn-in period.

Value

An object of class `mcdraws` containing posterior draws as well as some meta information.

Examples

```
# 1. create a sampler function
sampler <- new.env()
sampler$draw <- function(p) list(x=rnorm(1L), y=runif(1L))
# 2. do the simulation
sim <- MCMCsim(sampler, store=c("x", "y"))
str(sim)
summary(sim)

# example that requires start values or a start function
sampler$draw <- function(p) list(x=rnorm(1L), y=p$x * runif(1L))
sampler$start <- function(p) list(x=rnorm(1L), y=runif(1L))
sim <- MCMCsim(sampler, store=c("x", "y"))
summary(sim)
plot(sim, c("x", "y"))
```

```
# example using create_sampler; first generate some data
n <- 100
dat <- data.frame(x=runif(n), f=as.factor(sample(1:4, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, prior=pr_normal(precision=1), name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ x + f, data=dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=400, n.chain=2)
(summary(sim))
gd$pars
```

`mec` — *Create a model component object for a regression (fixed effects) component in the linear predictor with measurement errors in quantitative covariates*

Description

This function is intended to be used on the right hand side of the `formula` argument to `create_sampler` or `generate_data`. It creates an additive regression term in the model's linear predictor. Covariates are assumed to be measured subject to normally distributed errors with zero mean and variance specified using the `formula` or `V` arguments. Note that this means that `formula` should only contain quantitative variables, and no intercept. By default, the prior for the regression coefficients is improper uniform. A proper normal prior can be set up using function `pr_normal`, and passed to argument `prior`. It should be noted that `pr_normal` expects a precision matrix as input for its second argument, and that the prior variance (matrix) is taken to be the inverse of this precision matrix, where in case the model's family is `"gaussian"` this matrix is additionally multiplied by the residual scalar variance parameter `sigma_^2`.

Usage

```
mec(
  formula = ~1,
  sparse = NULL,
  X = NULL,
  V = NULL,
  prior = NULL,
  Q0 = NULL,
  b0 = NULL,
  R = NULL,
  r = NULL,
  S = NULL,
  s = NULL,
  lower = NULL,
  upper = NULL,
  name = "",
  debug = FALSE
)
```

Arguments

`formula`	a formula specifying the predictors subject to measurement error and possibly their variances as well. In the latter case the formula syntax `~ (x1 | V.x1) + (x2 | V.x2) + ...` should be used where `x1, x2, ...` are the names of (quantitative) predictors and `V.x1, V.x2, ...` are the names of the variables holding the corresponding measurement error variances. If only the predictors are specified the formula has the usual form `~ x1 + x2 + ...`. In that case variances should be specified using argument `V`. All variable names are looked up in the data frame passed as `data` argument to `create_sampler` or `generate_data`, or in `environment(formula)`.
`sparse`	whether the model matrix associated with `formula` should be sparse. The default is to base this on a simple heuristic.
`X`	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on `formula`. If `X` is specified `formula` is ignored.
`V`	measurement error variance; can contain zeros
`prior`	prior specification for the regression coefficients. Currently only normal priors are supported, specified using function `pr_normal`.
`Q0`	prior precision matrix for the regression effects. The default is a zero matrix corresponding to a noninformative improper prior. It can be specified as a scalar value, as a numeric vector of appropriate length, or as a matrix object. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.
`b0`	prior mean for the regression effect. Defaults to a zero vector. It can be specified as a scalar value or as a numeric vector of appropriate length. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.
`R`	optional constraint matrix for equality restrictions R'x = r where `x` is the vector of regression effects.
`r`	right hand side for the equality constraints.
`S`	optional constraint matrix for inequality constraints S'x >= s where x is the vector of regression effects.
`s`	right hand side for the inequality constraints.
`lower`	as an alternative to `s`, `lower` and `upper` may be specified for two-sided constraints lower <= S'x <= upper.
`upper`	as an alternative to `s`, `lower` and `upper` may be specified for two-sided constraints lower <= S'x <= upper.
`name`	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'reg' with the number of the model term attached.
`debug`	if `TRUE` a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

References

L.M. Ybarra and S.L. Lohr (2008). Small area estimation when auxiliary information is measured with error. Biometrika 95(4), 919-931.

S. Arima, G.S. Datta and B. Liseo (2015). Bayesian estimators for small area models when auxiliary information is measured with error. Scandinavian Journal of Statistics 42(2), 518-529.

Examples

```
# example of Ybarra and Lohr (2008)
m <- 50
X <- rnorm(m, mean=5, sd=3)  # true covariate values
v <- rnorm(m, sd=2)
theta <- 1 + 3*X + v  # true values
psi <- rgamma(m, shape=4.5, scale=2)
e <- rnorm(m, sd=sqrt(psi))  # sampling error
y <- theta + e  # direct estimates
C <- c(rep(3, 10), rep(0, 40))  # measurement error for first 10 values
W <- X + rnorm(m, sd=sqrt(C))  # covariate subject to measurement error

# fit Ybarra-Lohr model
sampler <- create_sampler(
  y ~ 1 + mec(~ 0 + W, V=C) + gen(factor=~local_),
  Q0=1/psi, sigma.fixed=TRUE, linpred="fitted"
)
sim <- MCMCsim(sampler, n.iter=800, n.chain=2, store.all=TRUE, verbose=FALSE)
(summ <- summary(sim))
plot(X, W, xlab="true X", ylab="inferred X")
points(X, summ$mec2_X[, "Mean"], col="green")
abline(0, 1, col="red")
legend("topleft", legend=c("prior mean", "posterior mean"), col=c("black", "green"), pch=c(1,1))
```

`model-information-criteria` *Compute DIC, WAIC and leave-one-out cross-validation model measures*

Description

Compute the Deviance Information Criterion (DIC) or Watanabe-Akaike Information Criterion (WAIC) from an object of class `mcdraws` output by `MCMCsim`. Method `waic.mcdraws` computes WAIC using package **loo**. Method `loo.mcdraws` also depends on package **loo** to compute a Pareto-smoothed importance sampling (PSIS) approximation to leave-one-out cross-validation.

Usage

```
compute_DIC(x, use.pV = FALSE)

compute_WAIC(
  x,
  diagnostic = FALSE,
  batch.size = NULL,
  show.progress = TRUE,
  cl = NULL,
  n.cores = 1L
)

## S3 method for class 'mcdraws'
waic(x, by.unit = FALSE, ...)

## S3 method for class 'mcdraws'
loo(x, by.unit = FALSE, r_eff = FALSE, n.cores = 1L, ...)
```

Arguments

`x`	an object of class `mcdraws`.
`use.pV`	whether half the posterior variance of the deviance should be used as an alternative estimate of the effective number of model parameters for DIC.
`diagnostic`	whether vectors of log-pointwise-predictive-densities and pointwise contributions to the WAIC effective number of model parameters should be returned.
`batch.size`	number of data units to process per batch.
`show.progress`	whether to show a progress bar.
`cl`	an existing cluster can be passed for parallel computation. If `cl` is provided, `n.cores` will be set to the number of workers in that cluster. If `NULL` and `n.cores` > 1, a new cluster is created.
`n.cores`	the number of cpu cores to use. Default is one, i.e. no parallel computation.
`by.unit`	if `TRUE` the computation is carried out unit-by-unit, which is slower but uses much less memory.
`...`	Other arguments, passed to `loo`. Not currently used by `waic.mcdraws`.
`r_eff`	whether to compute relative effective sample size estimates for the likelihood of each observation. This takes more time, but should result in a better PSIS approximation. See `loo`.

Value

For `compute_DIC` a vector with the deviance information criterion and effective number of model parameters. For `compute_WAIC` a vector with the WAIC model selection criterion and WAIC effective number of model parameters. Method `waic` returns an object of class `waic, loo`, see the documentation for `waic` in package **loo**. Method `loo` returns an object of class `psis_loo`, see `loo`.

References

D. Spiegelhalter, N. Best, B. Carlin and A. van der Linde (2002). Bayesian Measures of Model Complexity and Fit. Journal of the Royal Statistical Society B 64 (4), 583-639.

S. Watanabe (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. Journal of Machine Learning 11, 3571-3594.

A. Gelman, J. Hwang and A. Vehtari (2014). Understanding predictive information criteria for Bayesian models. Statistics and Computing 24, 997-1016.

A. Vehtari, D. Simpson, A. Gelman, Y. Yao and J. Gabry (2015). Pareto smoothed importance sampling. arXiv:1507.02646.

A. Vehtari, A. Gelman and J. Gabry (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing 27, 1413-1432.

P.-C. Buerkner, J. Gabry and A. Vehtari (2021). Efficient leave-one-out cross-validation for Bayesian non-factorized normal and Student-t models. Computational Statistics 36, 1243-1261.

Examples

```
ex <- mcmcsae_example(n=100)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, n.chain=4, store.all=TRUE)
compute_DIC(sim)
compute_WAIC(sim)
if (require(loo)) {
  waic(sim)
  loo(sim, r_eff=TRUE)
}
```

---

model_matrix *Compute possibly sparse model matrix*

Description

Compute possibly sparse model matrix

Usage

```
model_matrix(
  formula,
  data = NULL,
  contrasts.arg = NULL,
  drop.unused.levels = FALSE,
  sparse = NULL,
  drop0 = TRUE,
  catsep = "",
  by = NULL,
  tabM = FALSE,
  enclos = .GlobalEnv
)
```

Arguments

`formula`	model formula.
`data`	data frame containing all variables used in `formula`. These variables should not contain missing values. An error is raised in case any of them does.
`contrasts.arg`	specification of contrasts for factor variables. Currently supported are "contr.none" (no contrasts applied), "contr.treatment" (first level removed) and "contr.SAS" (last level removed). Alternatively, a named list specifying a single level per factor variable can be passed.
`drop.unused.levels`	whether empty levels of individual factor variables should be removed.
`sparse`	if `TRUE` a sparse matrix of class `dgCMatrix` is returned. This can be efficient for large datasets and a model containing categorical variables with many categories. If `sparse=NULL`, the default, whether a sparse or dense model matrix is returned is based on a simple heuristic.
`drop0`	whether to drop any remaining explicit zeros in resulting sparse matrix.
`catsep`	separator for concatenating factor variable names and level names. By default it is the empty string, reproducing the labels of `model.matrix`.
`by`	a vector by which to aggregate the result.
`tabM`	if `TRUE` return a list of tabMatrix objects.
`enclos`	enclosure to look for objects not found in `data`.

Value

Design matrix X, either an ordinary matrix or a sparse `dgCMatrix`.

`nchains-ndraws-nvars` *Get the number of chains, samples per chain or the number of variables in a simulation object*

Description

Get the number of chains, samples per chain or the number of variables in a simulation object

Usage

```
nchains(obj)

ndraws(obj)

nvars(dc)
```

Arguments

`obj`	an mcdraws object or a draws component (dc) object.
`dc`	a draws component object.

Value

The number of chains or retained samples per chain or the number of variables.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=5, store.all=TRUE)
# resolve possible conflict with posterior package:
nchains <- mcmcsae::nchains; ndraws <- mcmcsae::ndraws
nchains(sim); nchains(sim$beta)
ndraws(sim); ndraws(sim$beta)
nvars(sim$beta); nvars(sim$sigma_); nvars(sim$llh_); nvars(sim$v)
plot(sim, "beta")
nchains(subset(sim$beta, chains=1:2))
ndraws(subset(sim$beta, draws=sample(1:ndraws(sim), 100)))
nvars(subset(sim$u, vars=1:2))
```

`par_names` *Get the parameter names from an mcdraws object*

Description

Get the parameter names from an mcdraws object

Usage

```
par_names(obj)
```

Arguments

`obj`	an mcdraws object.

Value

The names of the parameters whose MCMC simulations are stored in `obj`.

Examples

```
data(iris)
sampler <- create_sampler(Sepal.Length ~
    reg(~ Petal.Length + Species, name="beta"), data=iris)
sim <- MCMCsim(sampler, burnin=100, n.iter=400)
(summary(sim))
par_names(sim)
```

`plot.dc` *Trace, density and autocorrelation plots for (parameters of a) draws component (dc) object*

Description

Trace, density and autocorrelation plots for (parameters of a) draws component (dc) object

Usage

```
## S3 method for class 'dc'
plot(x, nrows, ncols, ask = FALSE, ...)
```

Arguments

`x`	a draws component object.
`nrows`	number of rows in plot layout.
`ncols`	number of columns in plot layout.
`ask`	ask before plotting the next page; default is `FALSE`.
`...`	arguments passed to `density`.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
plot(sim$u)
```

`plot.mcdraws` *Trace, density and autocorrelation plots*

Description

Trace, density and autocorrelation plots for selected components of an `mcdraws` object.

Usage

```
## S3 method for class 'mcdraws'
plot(x, vnames, nrows, ncols, ask = FALSE, ...)
```

Arguments

`x`	an object of class `mcdraws`.
`vnames`	optional character vector to select a subset of parameters.
`nrows`	number of rows in plot layout.
`ncols`	number of columns in plot layout.
`ask`	ask before plotting the next page; default is `FALSE`.
`...`	arguments passed to `density`.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
plot(sim, c("beta", "u", "u_sigma", "v_sigma"), ask=TRUE)
```

`plot_coef` *Plot a set of model coefficients or predictions with uncertainty intervals based on summaries of simulation results or other objects.*

Description

This function plots estimates with error bars. Multiple sets of estimates can be compared. The error bars can either be based on standard errors or on explicitly specified lower and upper bounds. The function is adapted from function `plot.sae` in package **hbsae**, which in turn was adapted from function `coefplot.default` from package **arm**.

Usage

```
plot_coef(
  ...,
  n.se = 1,
  est.names,
  sort.by = NULL,
  decreasing = FALSE,
  index = NULL,
  maxrows = 50L,
  maxcols = 6L,
  offset = 0.1,
  cex.var = 0.8,
  mar = c(0.1, 2.1, 5.1, 0.1)
)
```

Arguments

`...`	`dc_summary` objects (output by the `summary` method for simulation objects of class `dc`), `sae` objects (output by the functions of package **hbsae**), or lists. In case of a list the components used are those with name `est` for point estimates, `se` for standard error based intervals or `lower` and `upper` for custom intervals. Instead of `dc_summary` objects matrix objects are also supported as long as they contain columns named "Mean" and "SD" as do `dc_summary` objects. Named parameters of other types that do not match any other argument names are passed to lower-level plot functions.
`n.se`	number of standard errors below and above the point estimates to use for error bars. By default equal to 1. This only refers to the objects of class `dc_summary` and `sae`.
`est.names`	labels to use in the legend for the components of the `...` argument
`sort.by`	vector by which to sort the coefficients, referring to the first object passed.
`decreasing`	if `TRUE`, sort in decreasing order (default).
`index`	vector of names or indices of the selected areas to be plotted.
`maxrows`	maximum number of rows in a column.
`maxcols`	maximum number of columns of estimates on a page.
`offset`	space used between plots of multiple estimates for the same area.
`cex.var`	the font size for the variable names, default=0.8.
`mar`	a numerical vector of the form `c(bottom, left, top, right)`, specifying the number of lines of margin on each of the four sides of the plot.

Examples

```
# create artificial data
set.seed(21)
n <- 100
dat <- data.frame(
  x=runif(n),
  f=factor(sample(1:20, n, replace=TRUE))
)
model <- ~ reg(~ x, prior=pr_normal(precision=1), name="beta") + gen(factor=~f, name="v")
gd <- generate_data(model, data=dat)
dat$y <- gd$y
# fit a base model
model0 <- y ~ reg(~ 1, name="beta") + gen(factor=~f, name="v")
sampler <- create_sampler(model0, data=dat, block=TRUE)
sim <- MCMCsim(sampler, store.all=TRUE)
(summ0 <- summary(sim))
# fit 'true' model
model <- y ~ reg(~ x, name="beta") + gen(factor=~f, name="v")
sampler <- create_sampler(model, data=dat, block=TRUE)
sim <- MCMCsim(sampler, store.all=TRUE)
(summ <- summary(sim))
# compare random effect estimates against true parameter values
plot_coef(summ0$v, summ$v, list(est=gd$pars$v), n.se=2, offset=0.2,
  maxrows=10, est.names=c("base model", "true model", "true"))
```

`posterior-moments` *Get means or standard deviations of parameters from the MCMC output in an mcdraws object*

Description

Get means or standard deviations of parameters from the MCMC output in an mcdraws object

Usage

```
get_means(obj, vnames = NULL)

get_sds(obj, vnames = NULL)
```

Arguments

`obj`	an object of class `mcdraws`.
`vnames`	optional character vector to select a subset of parameters.

Value

A list with simulation means or standard deviations.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4)
get_means(sim)
get_means(sim, "e_")
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4,
  store.mean=c("beta", "u"), store.sds=TRUE)
summary(sim, "beta")
get_means(sim, "beta")
get_sds(sim, "beta")
get_means(sim, "u")
get_sds(sim, "u")
```

`predict.mcdraws` *Generate draws from the predictive distribution*

Description

Generate draws from the predictive distribution

Usage

```
## S3 method for class 'mcdraws'
predict(
  object,
  newdata = NULL,
  X. = if (is.null(newdata)) "in-sample" else NULL,
  type = c("data", "link", "response", "data_cat"),
  var = NULL,
  ny = NULL,
  ry = NULL,
  fun. = identity,
  labels = NULL,
  ppcheck = FALSE,
  iters = NULL,
  to.file = FALSE,
  filename,
  write.single.prec = FALSE,
  show.progress = TRUE,
  verbose = TRUE,
  n.cores = 1L,
  cl = NULL,
  seed = NULL,
  export = NULL,
  ...
)
```

Arguments

`object`	an object of class `mcdraws`, as output by `MCMCsim`.
`newdata`	data frame with auxiliary information to be used for prediction.
`X.`	a list of design matrices; alternatively, `X.` equals 'in-sample' or 'linpred'. If 'in-sample' (the default if newdata is not supplied), the design matrices for in-sample prediction are used. If 'linpred' the 'linpred_' component of `object` is used.
`type`	the type of predictions. The default is `"data"`, meaning that new data is generated according to the predictive distribution. If `type="link"` only the linear predictor for the mean is generated, and in case `type="response"` the linear predictor is transformed to the response scale. For Gaussian models `type="link"` and `type="response"` are equivalent. For binomial and negative binomial models `type="response"` returns the simulations of the latent probabilities. For multinomial models `type="link"` generates the linear predictor for all categories except the last, and `type="response"` transforms this vector to the probability scale, and `type="data"` generates the multinomial data, all in long vector format, where the output for all categories (except the last) are stacked. For multinomial models and single trials, a further option is `type="data_cat"`, which generates the data as a categorical vector, with integer coded levels.
`var`	variance(s) used for out-of-sample prediction. By default 1.
`ny`	number of trials used for out-of-sample prediction in case of a binomial model. By default 1.
`ry`	fixed part of the (reciprocal) dispersion parameter in case of a negative binomial model.
`fun.`	function applied to the vector of posterior predictions to compute one or multiple summaries or test statistics. The function can have one or two arguments. The first argument is always the vector of posterior predictions. The optional second argument represents a list of model parameters, needed only when a test statistic depends on them. The function must return an integer or numeric vector.
`labels`	optional names for the output object. Must be a vector of the same length as the result of `fun.`.
`ppcheck`	if `TRUE`, function `fun.` is also applied to the observed data and an MCMC approximation is computed of the posterior predictive probability that the test statistic for predicted data is greater than the test statistic for the observed data.
`iters`	iterations in `object` to use for prediction. Default `NULL` means that all draws from `object` are used.
`to.file`	if `TRUE` the predictions are streamed to file.
`filename`	name of the file to write predictions to in case `to.file=TRUE`.
`write.single.prec`	Whether to write to file in single precision. Default is `FALSE`.
`show.progress`	whether to show a progress bar.

`verbose`	whether to show informative messages.
`n.cores`	the number of cpu cores to use. Default is one, i.e. no parallel computation. If an existing cluster `cl` is provided, `n.cores` will be set to the number of workers in that cluster.
`cl`	an existing cluster can be passed for parallel computation. If `NULL` and `n.cores > 1`, a new cluster is created.
`seed`	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
`export`	a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in `fun.` depend on global variables.
`...`	currently not used.

Value

An object of class `dc`, containing draws from the posterior (or prior) predictive distribution. If `ppcheck=TRUE` posterior predictive p-values are returned as an additional attribute. In case `to.file=TRUE` the file name used is returned.

Examples

```
n <- 250
dat <- data.frame(x=runif(n))
dat$y <- 1 + dat$x + rnorm(n)
sampler <- create_sampler(y ~ x, data=dat)
sim <- MCMCsim(sampler)
summary(sim)
# in-sample prediction
pred <- predict(sim, ppcheck=TRUE)
hist(attr(pred, "ppp"))
# out-of-sample prediction
pred <- predict(sim, newdata=data.frame(x=seq(0, 1, by=0.1)))
summary(pred)
```

`print.dc_summary` *Display a summary of a* `dc` *object*

Description

Display a summary of a `dc` object

Usage

```
## S3 method for class 'dc_summary'
print(
  x,
  digits = 3L,
  max.lines = 1000L,
  tail = FALSE,
  sort = NULL,
  max.label.length = NULL,
  ...
)
```

Arguments

`x`	an object of class `dc_summary`.
`digits`	number of digits to use, defaults to 3.
`max.lines`	maximum number of lines to display. If `NULL`, all elements are displayed.
`tail`	if `TRUE` the last instead of first at most `max.lines` are displayed.
`sort`	column name on which to sort the output.
`max.label.length`	if specified, printed row labels will be abbreviated to at most this length.
`...`	passed on to `print.default`.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
print(summary(sim$u), sort="n_eff")
```

`print.mcdraws_summary` *Print a summary of MCMC simulation results*

Description

Display a summary of an `mcdraws` object, as output by `MCMCsim`.

Usage

```
## S3 method for class 'mcdraws_summary'
print(x, digits = 3L, max.lines = 10L, tail = FALSE, sort = NULL, ...)
```

Arguments

`x`	an object of class `mcdraws_summary` as output by `summary.mcdraws`.
`digits`	number of digits to use, defaults to 3.
`max.lines`	maximum number of elements per vector parameter to display. If `NULL`, all elements are displayed.
`tail`	if `TRUE` the last instead of first `max.lines` of each component are displayed.
`sort`	column name on which to sort the output.
`...`	passed on to `print.default`.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
print(summary(sim), sort="n_eff")
```

`pr_exp` *Create an object representing exponential prior distributions*

Description

Create an object representing exponential prior distributions

Usage

```
pr_exp(scale = 1)
```

Arguments

`scale`	scalar or vector scale parameter.

Value

An environment representing the specified prior, for internal use.

`pr_fixed` *Create an object representing a degenerate prior fixing a parameter (vector) to a fixed value*

Description

Create an object representing a degenerate prior fixing a parameter (vector) to a fixed value

Usage

```
pr_fixed(value = 1)
```

Arguments

`value`	scalar or vector value parameter.

Value

An environment representing the specified prior, for internal use.

`pr_gamma` *Create an object representing gamma prior distributions*

Description

Create an object representing gamma prior distributions

Usage

```
pr_gamma(shape = 1, rate = 1)
```

Arguments

`shape`	scalar or vector shape parameter.
`rate`	scalar or vector rate, i.e. inverse scale, parameter.

Value

An environment representing the specified prior, for internal use.

pr_gig — *Create an object representing Generalized Inverse Gaussian (GIG) prior distributions*

Description

Create an object representing Generalized Inverse Gaussian (GIG) prior distributions

Usage

```
pr_gig(a, b, p)
```

Arguments

`a`	scalar or vector parameter.
`b`	scalar or vector parameter.
`p`	scalar or vector parameter.

Value

An environment representing the specified prior, for internal use.

pr_invchisq — *Create an object representing inverse chi-squared priors with possibly modeled degrees of freedom and scale parameters*

Description

Create an object representing inverse chi-squared priors with possibly modeled degrees of freedom and scale parameters

Usage

```
pr_invchisq(df = 1, scale = 1)
```

Arguments

`df` degrees of freedom parameter. This can be a numeric scalar or vector of length `n`, the dimension of the parameter vector. Alternatively, for a scalar degrees of freedom parameter, `df="modeled"` or `df="modelled"` assign a default (gamma) prior to the degrees of freedom parameter. For more control of this gamma prior a list can be passed with some of the following components:

alpha0 shape parameter of the gamma distribution

beta0 rate parameter of the gamma distribution

- **proposal** "RW" for random walk Metropolis-Hastings or "mala" for Metropolis-adjusted Langevin
- **tau** (starting) scale of Metropolis-Hastings update
- **adapt** whether to adapt the scale of the proposal distribution during burnin to achieve better acceptance rates.

`scale`	scalar or vector scale parameter. Alternatively, `scale="modeled"` or `scale="modelled"` puts a default chi-squared prior on the scale parameter. For more control on this chi-squared prior a list can be passed with some of the following components:

- **df** degrees of freedom (scalar or vector)
- **scale** scale (scalar or vector)
- **common** whether the modeled scale parameter of the inverse chi-squared distribution is (a scalar parameter) common to all `n` parameters.

Value

An environment representing the specified prior, for internal use.

`pr_invwishart` *Create an object representing an inverse Wishart prior, possibly with modeled scale matrix*

Description

Create an object representing an inverse Wishart prior, possibly with modeled scale matrix

Usage

```
pr_invwishart(df = NULL, scale = NULL)
```

Arguments

`df`	Degrees of freedom parameter. This should be a scalar numeric value. Default value is the dimension plus one.
`scale`	Either a (known) scale matrix, or `scale="modeled"` or `scale="modelled"`, which puts default chi-squared priors on the diagonal elements of the inverse Wishart scale matrix. For more control on these chi-squared priors a list can be passed with some of the following components:

- **df** degrees of freedom (scalar or vector) of the chi-squared distribution(s)
- **scale** scale parameter(s) of the chi-squared distribution(s)
- **common** whether the modeled scale parameter of the inverse chi-squared distribution is (a scalar parameter) common to all `n` diagonal elements.

Value

An environment representing the specified prior, for internal use.

References

A. Huang and M.P. Wand (2013). Simple marginally noninformative prior distributions for covariance matrices. Bayesian Analysis 8, 439-452.

`pr_MLiG` *Create an object representing a Multivariate Log inverse Gamma (MLiG) prior distribution*

Description

Create an object representing a Multivariate Log inverse Gamma (MLiG) prior distribution

Usage

```
pr_MLiG(mean = 0, precision = 0, labels = NULL, a = 1000)
```

Arguments

`mean`	scalar or vector parameter for the mean in the large `a` limit, when the distribution approaches a normal distribution.
`precision`	scalar or vector parameter for the precision in the large `a` limit, when the distribution approaches a normal distribution.
`labels`	optional character vector with coefficient labels. If specified, it should have the same length as at least one of `mean` and `precision`, and in that case the MLiG prior with these parameters is assigned to these coefficients, while any coefficients not present in labels will be assigned a non-informative prior with mean 0 and precision 0.
`a`	scalar parameter that controls how close the prior is to independent normal priors with `mean` and `precision` parameters. The larger this value (default is 1000), the closer.

Value

An environment representing the specified prior, for internal use.

References

J.R. Bradley, S.H. Holan and C.K. Wikle (2018). Computationally efficient multivariate spatio-temporal models for high-dimensional count-valued data (with discussion). Bayesian Analysis 13(1), 253-310.

pr_normal *Create an object representing a possibly multivariate normal prior distribution*

Description

Create an object representing a possibly multivariate normal prior distribution

Usage

```
pr_normal(mean = 0, precision = 0, labels = NULL)
```

Arguments

`mean`	scalar or vector mean parameter.
`precision`	scalar, vector or matrix precision parameter.
`labels`	optional character vector with coefficient labels. If specified, it should have the same length as at least one of `mean` and `precision`, and in that case the normal prior with these parameters is assigned to these coefficients, while any coefficients not present in labels will be assigned a non-informative prior with mean 0 and precision 0.

Value

An environment representing the specified prior, for internal use.

read_draws *Read MCMC draws from a file*

Description

Read draws written to file by `MCMCsim` used with argument `to.file`.

Usage

```
read_draws(name, filename = paste0("MCdraws_", name, ".dat"))
```

Arguments

`name`	name of the parameter to load the corresponding file with posterior draws for.
`filename`	name of the file in which the draws are stored.

Value

An object of class `dc` containing MCMC draws for a (vector) parameter.

Examples

```
## Not run:
# NB this example creates a file "MCdraws_e_.dat" in the working directory
n <- 100
dat <- data.frame(x=runif(n), f=as.factor(sample(1:5, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, prior=pr_normal(precision=1), name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
# run the MCMC simulation and write draws of residuals to file:
sim <- MCMCsim(sampler, n.iter=500, to.file="e_")
summary(sim)
mcres <- read_draws("e_")
summary(mcres)

## End(Not run)
```

---

reg *Create a model component object for a regression (fixed effects) component in the linear predictor*

Description

This function is intended to be used on the right hand side of the `formula` argument to `create_sampler` or `generate_data`. It creates an additive regression term in the model's linear predictor. By default, the prior for the regression coefficients is improper uniform. A proper normal prior can be set up using function `pr_normal`, and passed to argument `prior`. It should be noted that `pr_normal` expects a precision matrix as input for its second argument, and that the prior variance (matrix) is taken to be the inverse of this precision matrix, where in case the model's family is `"gaussian"` this matrix is additionally multiplied by the residual scalar variance parameter `sigma_^2`.

Usage

```
reg(
  formula = ~1,
  remove.redundant = FALSE,
  sparse = NULL,
  X = NULL,
  prior = NULL,
  Q0 = NULL,
  b0 = NULL,
  R = NULL,
  r = NULL,
  S = NULL,
  s = NULL,
  lower = NULL,
  upper = NULL,
  name = "",
  debug = FALSE
)
```

Arguments

`formula`	a formula specifying the predictors to be used in the model, in the same way as the right hand side of the `formula` argument of R's `lm` function. Variable names are looked up in the data frame passed as `data` argument to `create_sampler` or `generate_data`, or in `environment(formula)`.
`remove.redundant`	whether redundant columns should be removed from the design matrix. Default is `FALSE`. But note that treatment contrasts are automatically applied to all factor variables in `formula`.
`sparse`	whether the model matrix associated with `formula` should be sparse. The default is to base this on a simple heuristic.
`X`	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on `formula`. If `X` is specified `formula` is ignored.
`prior`	prior specification for the regression coefficients. Supported priors can be specified using functions `pr_normal`, `pr_fixed`, or `pr_MLiG`. The latter prior is only available in conjunction with a gamma family sampling distribution.
`Q0`	prior precision matrix for the regression effects. The default is a zero matrix corresponding to a noninformative improper prior. It can be specified as a scalar value, as a numeric vector of appropriate length, or as a matrix object. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.
`b0`	prior mean for the regression effect. Defaults to a zero vector. It can be specified as a scalar value or as a numeric vector of appropriate length. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.
`R`	optional constraint matrix for equality restrictions R'x = r where `x` is the vector of regression effects.
`r`	right hand side for the equality constraints.
`S`	optional constraint matrix for inequality constraints S'x >= s where `x` is the vector of regression effects.
`s`	right hand side for the inequality constraints.
`lower`	as an alternative to `s`, `lower` and `upper` may be specified for two-sided constraints lower <= S'x <= upper.
`upper`	as an alternative to `s`, `lower` and `upper` may be specified for two-sided constraints lower <= S'x <= upper.
`name`	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'reg' with the number of the model term attached.
`debug`	if `TRUE` a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

Examples

```
data(iris)
# default: flat priors on regression coefficients
sampler <- create_sampler(Sepal.Length ~
    reg(~ Petal.Length + Species, name="beta"),
  data=iris
)
sim <- MCMCsim(sampler, burnin=100, n.iter=400)
summary(sim)
# (weakly) informative normal priors on regression coefficients
sampler <- create_sampler(Sepal.Length ~
    reg(~ Petal.Length + Species, prior=pr_normal(precision=1e-2), name="beta"),
  data=iris
)
sim <- MCMCsim(sampler, burnin=100, n.iter=400)
summary(sim)
# binary regression
sampler <- create_sampler(Species == "setosa" ~
    reg(~ Sepal.Length, prior=pr_normal(precision=0.1), name="beta"),
  family="binomial", data=iris)
sim <- MCMCsim(sampler, burnin=100, n.iter=400)
summary(sim)
pred <- predict(sim)
str(pred)
# example with equality constrained regression effects
n <- 500
df <- data.frame(x=runif(n))
df$y <- rnorm(n, 1 + 2*df$x)
R <- matrix(1, 2, 1)
r <- 3
sampler <- create_sampler(y ~ reg(~ 1 + x, R=R, r=r, name="beta"), data=df)
sim <- MCMCsim(sampler)
summary(sim)
plot(sim, "beta")
summary(transform_dc(sim$beta, fun=function(x) crossprod_mv(R, x) - r))
```

`residuals-fitted-values`

Extract draws of fitted values or residuals from an mcdraws object

---

Description

For a model created with `create_sampler` and estimated using `MCMCsim`, these functions return the posterior draws of fitted values or residuals. In the current implementation the fitted values correspond to the linear predictor and the residuals are computed as the data vector minus the fitted values, regardless of the model's distribution family. For large datasets the returned object can become very large. One may therefore select a subset of draws or chains or use `mean.only=TRUE` to return a vector of posterior means only.

Usage

```
## S3 method for class 'mcdraws'
fitted(
  object,
  mean.only = FALSE,
  units = NULL,
  chains = seq_len(nchains(object)),
  draws = seq_len(ndraws(object)),
  matrix = FALSE,
  type = c("link", "response"),
  ...
)

## S3 method for class 'mcdraws'
residuals(
  object,
  mean.only = FALSE,
  units = NULL,
  chains = seq_len(nchains(object)),
  draws = seq_len(ndraws(object)),
  matrix = FALSE,
  ...
)
```

Arguments

`object`	an object of class `mcdraws`.
`mean.only`	if `TRUE` only the vector of posterior means is returned. In that case the subsequent arguments are ignored. Default is `FALSE`.
`units`	the data units (by default all) for which fitted values or residuals should be computed.
`chains`	optionally, a selection of chains.
`draws`	optionally, a selection of draws per chain.
`matrix`	whether a matrix should be returned instead of a dc object.
`type`	the type of fitted values: "link" for fitted values on the linear predictor scale (the default), and "response" for fitted values on the response scale. Returned residuals are always on the response scale.
`...`	currently not used.

Value

Either a draws component object or a matrix with draws of fitted values or residuals. The residuals are always on the response scale, whereas fitted values can be on the scale of the linear predictor or the response depending on `type`. If `mean.only=TRUE`, a vector of posterior means.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, store.all=TRUE)
fitted(sim, mean.only=TRUE)
summary(fitted(sim))
residuals(sim, mean.only=TRUE)
summary(residuals(sim))
bayesplot::mcmc_intervals(as.matrix(subset(residuals(sim), vars=1:20)))
```

`sampler_control` *Set computational options for the sampling algorithms*

Description

Set computational options for the sampling algorithms

Usage

```
sampler_control(
  add.outer.R = TRUE,
  recompute.e = TRUE,
  expanded.cMVN.sampler = FALSE,
  CG = NULL,
  block = TRUE,
  block.V = TRUE,
  auto.order.block = TRUE,
  chol.control = chol_control(),
  max.size.cps.template = 100,
  PG.approx = TRUE,
  PG.approx.m = -2L,
  CRT.approx.m = 20L
)
```

Arguments

`add.outer.R`	whether to add the outer product of a constraint matrix for a better conditioned linear system of equations, typically for coefficients sampled in a Gibbs-block. Default is `TRUE`. If `NULL`, a simple heuristic is used to decide whether to add the outer product of possibly a submatrix of the constraint matrix.

`recompute.e`	when `FALSE`, residuals or linear predictors are only computed at the start of the simulation. This may give a modest speedup but in some cases may be less accurate due to round-off error accumulation. Default is `TRUE`.
`expanded.cMVN.sampler`	whether an expanded linear system including dual variables is used for equality constrained multivariate normal sampling. If set to `TRUE` this may improve the performance of the blocked Gibbs sampler in case of a large number of equality constraints, typically identifiability constraints for GMRFs.
`CG`	use a conjugate gradient iterative algorithm instead of Cholesky updates for sampling the model's coefficients. This must be a list with possible components `max.it`, `stop.criterion`, `verbose`, `preconditioner` and `scale`. See the help for function `CG_control`, which can be used to specify these options. Conjugate gradient sampling is currently an experimental feature that can be used for blocked Gibbs sampling but with some limitations.
`block`	if `TRUE`, the default, all coefficients are sampled in a single block. Alternatively, a list of character vectors with names of model components whose coefficients should be sampled together in blocks.
`block.V`	if `TRUE`, the default, all coefficients of `reg` and `gen` components in a variance model formula are sampled in a single block. Alternatively, a list of character vectors with names of model components whose coefficients should be sampled together in blocks.
`auto.order.block`	whether Gibbs blocks should be ordered automatically in such a way that those with the most sparse design matrices come first. This way of ordering can make Cholesky updates more efficient.
`chol.control`	options for Cholesky decomposition, see `chol_control`.
`max.size.cps.template`	maximum allowed size in MB of the sparse matrix serving as a template for the sparse symmetric crossproduct X'QX of a dgCMatrix X, where Q is a diagonal matrix subject to change.
`PG.approx`	whether Polya-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, `BayesLogit::rpg` is used, which is exact for some values of the shape parameter.
`PG.approx.m`	if `PG.approx=TRUE`, the number of explicit gamma draws in the sum-of-gammas representation of the Polya-Gamma distribution. The remainder (infinite) convolution is approximated by a single moment-matching gamma draw. Special values are: `-2L` for a default choice depending on the value of the shape parameter balancing performance and accuracy, `-1L` for a moment-matching normal approximation, and `0L` for a moment-matching gamma approximation.
`CRT.approx.m`	scalar integer specifying the degree of approximation to sampling from a Chinese Restaurant Table distribution. The approximation is based on Le Cam's theorem. Larger values yield a slower but more accurate sampler.

Value

A list with specified computational options used by various sampling functions.

References

D. Bates, M. Maechler, B. Bolker and S.C. Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software 67(1), 1-48.

Y. Chen, T.A. Davis, W.W. Hager and S. Rajamanickam (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Transactions on Mathematical Software 35(3), 1-14.

`SBC_test` *Simulation based calibration*

Description

Simulation based calibration

Usage

```
SBC_test(
  ...,
  pars,
  n.draws = 25L,
  n.sim = 20L * n.draws,
  burnin = 25L,
  thin = 2L,
  show.progress = TRUE,
  verbose = TRUE,
  n.cores = 1L,
  cl = NULL,
  seed = NULL,
  export = NULL
)
```

Arguments

`...`	passed to `create_sampler` (can be all parameters except `prior.only`)
`pars`	named list with univariate functions of the parameters to use in test. This list is passed to argument `pred` of `MCMCsim`.
`n.draws`	number of posterior draws to retain in posterior simulations.
`n.sim`	number of simulation iterations.
`burnin`	burnin to use in posterior simulations, passed to `MCMCsim`.
`thin`	thinning to use in posterior simulations, passed to `MCMCsim`.
`show.progress`	whether a progress bar should be shown.
`verbose`	set to `FALSE` to suppress messages.
`n.cores`	the number of cpu cores to use. Default is one, i.e. no parallel computation. If an existing cluster `cl` is provided, `n.cores` will be set to the number of workers in that cluster.

`cl`	an existing cluster can be passed for parallel computation. If `NULL` and `n.cores > 1`, a new cluster is created.
`seed`	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
`export`	a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in the model formulae depend on global variables.

Value

A matrix with ranks.

References

M. Modrak, A.H. Moon, S. Kim, P. Buerkner, N. Huurre, K. Faltejskova, A. Gelman and A. Vehtari (2023). Simulation-based calibration checking for Bayesian computation: The choice of test quantities shapes sensitivity. Bayesian Analysis, 1(1), 1-28.

Examples

```
## Not run:
# this example may take a long time
n <- 10L
dat <- data.frame(x=runif(n))
ranks <- SBC_test(~ reg(~ 1 + x, prior=pr_normal(mean=c(0.25, 1), precision=1), name="beta"),
  sigma.mod=pr_invchisq(df=1, scale=list(df=1, scale=1)), data=dat,
  pars=list(mu="beta[1]", beta_x="beta[2]", sigma="sigma_"),
  n.draws=9L, n.sim=10L*20L, thin=2L, burnin=20L
)
ranks

## End(Not run)
```

`setup_cluster` *Set up a cluster for parallel computing*

Description

The cluster is set up for a number of workers by loading the **mcmcsae** package and setting up independent RNG streams.

Usage

```
setup_cluster(n.cores = NULL, seed = NULL, export = NULL)
```

Arguments

`n.cores`	the number of cpu cores to use.
`seed`	optional random seed for reproducibility.
`export`	a character vector with names of objects to export to the workers.

Value

An object representing the cluster.

`stop_cluster` *Stop a cluster*

Description

Stop a cluster set up by `setup_cluster`.

Usage

```
stop_cluster(cl)
```

Arguments

`cl`	the cluster object.

Value

`NULL`.

`subset.dc` *Select a subset of chains, samples and parameters from a draws component (dc) object*

Description

Select a subset of chains, samples and parameters from a draws component (dc) object

Usage

```
## S3 method for class 'dc'
subset(
  x,
  chains = seq_len(nchains(x)),
  draws = seq_len(ndraws(x)),
  vars = seq_len(nvars(x)),
  ...
)
```

Arguments

`x`	a draws component (dc) object.
`chains`	an integer vector indicating which chains to select.
`draws`	an integer vector indicating which samples to select.
`vars`	an integer vector indicating which parameters to select.
`...`	not used.

Value

The selected part of the draws component as an object of class dc.

Examples

```
n <- 300
dat <- data.frame(x=runif(n), f=as.factor(sample(1:7, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, prior=pr_normal(precision=1), name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
sim <- MCMCsim(sampler)
(summary(sim$beta))
(summary(subset(sim$beta, chains=1)))
(summary(subset(sim$beta, chains=1, draws=sample(1:ndraws(sim), 100))))
(summary(subset(sim$beta, vars=1:2)))
```

`summary.dc` *Summarize a draws component (dc) object*

Description

Summarize a draws component (dc) object

Usage

```
## S3 method for class 'dc'
summary(
  object,
  probs = c(0.05, 0.5, 0.95),
  na.rm = FALSE,
  time = NULL,
  abbr = FALSE,
  batch.size = 100L,
  ...
)
```

Arguments

`object`	an object of class `dc`.
`probs`	vector of probabilities at which to evaluate quantiles.
`na.rm`	whether to remove NA/NaN draws in computing the summaries.
`time`	MCMC computation time; if specified the effective sample size per unit of time is returned in an extra column labeled 'efficiency'.
`abbr`	if `TRUE` abbreviate the labels in the output.
`batch.size`	number of parameter columns to process simultaneously. A larger batch size may speed things up a little, but if an out of memory error occurs it may be a good idea to use a smaller number and try again. The default is 100.
`...`	arguments passed to `n_eff`.

Value

A matrix with summaries of class `dc_summary`.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
summary(sim$u)
```

`summary.mcdraws` *Summarize an mcdraws object*

Description

Summarize an mcdraws object

Usage

```
## S3 method for class 'mcdraws'
summary(
  object,
  vnames = NULL,
  probs = c(0.05, 0.5, 0.95),
  na.rm = FALSE,
  efficiency = FALSE,
  abbr = FALSE,
  batch.size = 100L,
  ...
)
```

Arguments

`object`	an object of class `mcdraws`, typically generated by function `MCMCsim`.
`vnames`	optional character vector to select a subset of parameters.
`probs`	vector of probabilities at which to evaluate quantiles.
`na.rm`	whether to remove NA/NaN draws in computing the summaries.
`efficiency`	if `TRUE` the effective sample size per second of computation time is returned as well.
`abbr`	if `TRUE` abbreviate the labels in the output.
`batch.size`	number of parameter columns to process simultaneously for vector parameters. A larger batch size may speed things up a little, but if an out of memory error occurs it may be a good idea to use a smaller number and try again. The default is 100.
`...`	arguments passed to `n_eff`.

Value

A list of class `mcdraws_summary` summarizing `object`.

Examples

```
ex <- mcmcsae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE)
summary(sim)
par_names(sim)
summary(sim, c("beta", "v_sigma", "u_sigma"))
```

`TMVN-methods` *Functions for specifying the method and corresponding options for sampling from a possibly truncated and degenerate multivariate normal distribution*

Description

These functions are intended for use in the `method` argument of `create_TMVN_sampler`.

Usage

```
m_direct()

m_Gibbs(slice = FALSE, diagnostic = FALSE, debug = FALSE)

m_HMC(
  Tsim = pi/2,
  max.events = .Machine$integer.max,
  diagnostic = FALSE,
  debug = FALSE
)

m_HMCZigZag(
  Tsim = 1,
  rate = 1,
  prec.eq = NULL,
  diagnostic = FALSE,
  max.events = .Machine$integer.max,
  adapt = FALSE,
  debug = FALSE
)

m_softTMVN(
  sharpness = 100,
  useV = FALSE,
  CG = NULL,
  PG.approx = TRUE,
  PG.approx.m = -2L,
  debug = FALSE
)
```

Arguments

`slice`	if `TRUE`, a Gibbs within slice sampler is used.
`diagnostic`	whether information about violations of inequalities, bounces off inequality walls (for 'HMC' and 'HMCZigZag' methods) or gradient events (for 'HMCZigZag') is printed to the screen.
`debug`	if `TRUE` a breakpoint is set at the beginning of the TMVN sampling function. Mainly intended for developers.
`Tsim`	the duration of a Hamiltonian Monte Carlo simulated particle trajectory. This can be specified as either a single positive numeric value for a fixed simulation time, or as a function that is applied in each MCMC iteration to generates a simulation time.
`max.events`	maximum number of events (reflections off inequality walls and for method 'HMCZigZag' also gradient events). Default is unlimited. Specifying a finite number may speed up the sampling but may also result in a biased sampling algorithm.
`rate`	vector of Laplace rate parameters for method 'HMCZigZag'. It must be a positive numeric vector of length one or the number of variables.
`prec.eq`	positive numeric vector of length 1 or the number of equality restrictions, to control the precision with which the equality restrictions are imposed; the larger `prec.eq` the more precisely they will be imposed.

`adapt`	experimental feature: if `TRUE` the rate parameter will be adapted in an attempt to make the sampling algorithm more efficient.
`sharpness`	for method 'softTMVN', the sharpness of the soft inequalities; the larger the better the approximation of exact inequalities. It must a positive numeric vector of length one or the number of inequality restrictions.
`useV`	for method 'softTMVN' whether to base computations on variance instead of precision matrices.
`CG`	use a conjugate gradient iterative algorithm instead of Cholesky updates for sampling the model's coefficients. This must be a list with possible components `max.it`, `stop.criterion`, `verbose`. See the help for function `CG_control`, which can be used to specify these options. Currently the preconditioner and scale options cannot be set for this use case.
`PG.approx`	see `sampler_control`.
`PG.approx.m`	see `sampler_control`.

Value

A method object, for internal use only.

`transform_dc` *Transform one or more draws component objects into a new one by applying a function*

Description

Transform one or more draws component objects into a new one by applying a function

Usage

```
transform_dc(..., fun, to.matrix = FALSE, labels = NULL)
```

Arguments

`...`	draws component object(s) of class `dc`.
`fun`	a function to apply. This function should take as many arguments as there are input objects. The arguments can be arbitrarily named, but they are assumed to be in the same order as the input objects. The function should return a vector.
`to.matrix`	if `TRUE` the output is in matrix format; otherwise it is a draws component object.
`labels`	optional labels for the output object.

Value

Either a matrix or a draws component object.

Examples

```
ex <- mcmcsae_example(n=50)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=300, thin=2, n.chain=4, store.all=TRUE)
summary(sim$v_sigma)
summary(transform_dc(sim$v_sigma, fun=function(x) x^2))
summary(transform_dc(sim$u, sim$u_sigma, fun=function(x1, x2) abs(x1)/x2))
```

---

vfac — *Create a model component object for a variance factor component in the variance function of a gaussian sampling distribution*

Description

This function is intended to be used on the right hand side of the `formula.V` argument to `create_sampler` or `generate_data`.

Usage

```
vfac(
  factor = "local_",
  prior = pr_invchisq(df = 1, scale = 1),
  name = "",
  debug = FALSE
)
```

Arguments

`factor`	The name of a factor variable. The name `"local_"` has a special meaning, and assigns a different variance scale parameter to each data unit. In case of inverse chi-squared priors this implies that the marginal sampling distribution is a t distribution. In case of exponential priors the marginal sampling distribution is a Laplace or double exponential distribution.
`prior`	the prior assigned to the variance factors. Currently the prior can be inverse chi-squared or exponential, specified by a call to `pr_invchisq` or `pr_exp`, respectively. The default priors are inverse chi-squared with 1 degree of freedom. See the help pages of the prior specification functions for details on how to set non-default priors.
`name`	The name of the variance model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'vfac' with the number of the variance model term attached.
`debug`	If `TRUE` a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

`vreg` *Create a model component object for a regression component in the variance function of a gaussian sampling distribution*

Description

This function is intended to be used on the right hand side of the `formula.V` argument to `create_sampler` or `generate_data`.

Usage

```
vreg(
  formula = NULL,
  remove.redundant = FALSE,
  sparse = NULL,
  X = NULL,
  prior = NULL,
  Q0 = NULL,
  b0 = NULL,
  name = ""
)
```

Arguments

`formula`	a formula for the regression effects explaining the log-variance. Variable names are looked up in the data frame passed as `data` argument to `create_sampler` or `generate_data`, or in `environment(formula)`.
`remove.redundant`	whether redundant columns should be removed from the design matrix. Default is `FALSE`.
`sparse`	whether the model matrix associated with `formula` should be sparse. The default is determined by a simple heuristic based on storage size.
`X`	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on `formula`. If `X` is specified `formula` is ignored.
`prior`	prior specification for the coefficients. Currently only normal priors are supported, specified using function `pr_normal`.
`Q0`	prior precision matrix for the regression effects. The default is a zero matrix corresponding to a noninformative improper prior. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.

`b0`	prior mean for the regression effect. Defaults to a zero vector. DEPRECATED, please use argument `prior` instead, i.e. `prior = pr_normal(mean = b0.value, precision = Q0.value)`.
`name`	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'vreg' with the number of the variance model term attached.

Value

An object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Intended for internal use by other package functions.

References

E. Cepeda and D. Gamerman (2000). Bayesian modeling of variance heterogeneity in normal regression models. Brazilian Journal of Probability and Statistics, 207-221.

T.I. Lin and W.L. Wang (2011). Bayesian inference in joint modelling of location and scale parameters of the t distribution for longitudinal data. Journal of Statistical Planning and Inference 141(4), 1543-1553.

`weights.mcdraws` *Extract weights from an mcdraws object*

Description

Extract weights from an mcdraws object

Usage

```
## S3 method for class 'mcdraws'
weights(object, ...)
```

Arguments

`object`	an object of class `mcdraws`.
`...`	currently not used.

Value

A vector with (simulation means of) weights.

Examples

```
# first create a population data frame
N <- 1000  # population size
pop <- data.frame(x=rnorm(N), area=factor(sample(1:10, N, replace=TRUE)))
pop$y <- 1 + 2*pop$x + seq(-1, to=1, length.out=10)[pop$area] + 0.5*rnorm(N)
pop$sample <- FALSE
pop$sample[sample(seq_len(N), 100)] <- TRUE
# a simple linear regression model:
sampler <- create_sampler(
  y ~ reg(~ x, name="beta"),
  linpred=list(beta=rowsum(model.matrix(~ x, pop), pop$area)), compute.weights=TRUE,
  data=pop[pop$sample, ]
)
sim <- MCMCsim(sampler)
(summary(sim))
str(weights(sim))
crossprod_mv(weights(sim), pop$y[pop$sample])
summary(sim$linpred_)
# a multilevel model:
sampler <- create_sampler(
  y ~ reg(~ x, name="beta") + gen(factor = ~ area, name="v"),
 linpred=list(beta=rowsum(model.matrix(~ x, pop), pop$area), v=diag(10)), compute.weights=TRUE,
  data=pop[pop$sample, ]
)
sim <- MCMCsim(sampler)
(summary(sim))
str(weights(sim))
crossprod_mv(weights(sim), pop$y[pop$sample])
summary(sim$linpred_)
```

Index