

Package ‘mcmcsae’

July 20, 2026

Version 0.8.1

License GPL-3

Title Markov Chain Monte Carlo Small Area Estimation

Type Package

LazyLoad yes

Encoding UTF-8

Language en-GB

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 2026-07-19

Depends R (>= 4.1.0)

Imports Matrix (>= 1.6.2), Rcpp (>= 0.11.0), methods, GIGrvg (>= 0.7), loo (>= 2.0.0), collapse

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

LinkingTo Rcpp, RcppEigen, Matrix, GIGrvg

VignetteBuilder knitr

Collate 'AR1_sampler.R' 'GMRF_extension.R' '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' 'f_binomial.R' 'f_gamma.R' 'f_gaussian.R' 'f_multi.R' 'f_multinomial.R' 'f_negbinomial.R' 'f_poisson.R' 'f_student_t.R' 'family.R' 'formulas.R' 'kronprod.R' 'mc_bart.R' 'mc_block.R' 'mc_gen.R' 'mc_gl.R' 'mc_mec.R' 'mc_offset.R' 'mc_reg.R' 'mc_s.R' 'mc_vfac.R' 'mc_vreg.R' 'mcmcsae.R' 'model_eval.R' 'modelmatrix.R' 'models.R' 'parallel.R' 'plots.R' 'prediction.R' 'priors.R' 'projection.R' 'random.R' 'samplers.R' 'sbc.R' 'set_constraints.R' 'sparse_template.R' 'utils.R'

Config/roxygen2/version 8.0.0

NeedsCompilation yes

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

Maintainer Harm Jan Boonstra <hjboonstra@gmail.com>

Repository CRAN

Date/Publication 2026-07-20 18:40:02 UTC

Contents

mcmcsae-package	4
acceptance_rates	4
aggrMatrix	5
as.data.frame.dc_summary	6
as.data.frame.mcdraws_summary	6
bayesR2	7
binomial_control	7
brt	8
CG_control	10
chol_control	11
cMVN_control	11
combine_chains	12
combine_iters	12
computeDesignMatrix	13
correlation	14
create_cMVN_sampler	17
create_sampler	18
create_TMVN_sampler	21
f_binomial	24
f_gamma	26
f_gaussian	27
f_gaussian_gamma	30
f_multi	31
f_multinomial	31
f_negbinomial	32
f_poisson	35
f_student_t	36
gaussian_control	37
gen	38
generate_data	41
gen_control	42
get_draw	43
glreg	44
GMRF_structure	45
labels	46
matrix-vector	47

maximize_log_lh_p	47
MCMC-diagnostics	49
MCMC-object-conversion	50
mcmcsae_example	51
MCMCsim	52
mc_offset	55
mec	56
model-information-criteria	58
model_matrix	60
multinomial_control	61
negbinomial_control	62
n_chains-n_draws-n_vars	62
par_names	63
plot.dc	64
plot.mcdraws	65
plot_coef	65
poisson_control	67
posterior-moments	68
predict.mcdraws	69
print.dc_summary	71
print.mcdraws_summary	72
pr_beta	73
pr_exp	73
pr_fixed	74
pr_gamma	74
pr_gig	75
pr_invchisq	75
pr_invwishart	76
pr_MLiG	77
pr_normal	78
pr_truncnormal	78
pr_unif	79
read_draws	79
reg	80
residuals-fitted-values	82
s	84
sampler_control	85
SBC_test	87
setup_cluster	89
set_constraints	89
set_MH	90
sim_marg_var	91
stop_cluster	92
student_t_control	92
subset.dc	93
summary.dc	94
summary.mcdraws	95
TMVN-methods	96

transform_dc 97

vfac 98

vreg 99

weights.mcdraws 100

Index 102

mcmcstae-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 modelling of temporal, spatial or other kinds of dependence structures. For Gaussian models the variance can be modelled too. By modelling 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 modelled
# degrees of freedom parameter in the variance part of the model)
sampler <- create_sampler(ex$model, data=ex$dat,
  family = f_gaussian(var.model = ~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)))

```

`as.data.frame.dc_summary`*Convert a dc_summary object to a data.frame*

Description

Convert a dc_summary object to a data.frame

Usage

```
## S3 method for class 'dc_summary'  
as.data.frame(x, row.names, optional, ...)
```

Arguments

x	a summary of a draws component object.
row.names	not used.
optional	not used.
...	not used.

Value

The dc_summary object as a data.frame.

`as.data.frame.mcdraws_summary`*Convert a mcdraws_summary object to a data.frame*

Description

Convert a mcdraws_summary object to a data.frame

Usage

```
## S3 method for class 'mcdraws_summary'  
as.data.frame(x, row.names, optional, ...)
```

Arguments

x	a summary of an object of class mcdraws.
row.names	not used.
optional	not used.
...	not used.

Value

The `mcdraws_summary` object as a `data.frame`.

bayesR2

Compute a Bayesian measure of percentage of explained variance

Description

Compute a Bayesian measure of percentage of explained variance

Usage

```
bayesR2(x, summary = TRUE)
```

Arguments

`x` object of class `mcdraws` as created by [MCMCsim](#).
`summary` if TRUE only a posterior summary is returned, otherwise a simulation (dc) object.

Value

Posterior summary or full simulation draws of a Bayesian version of R-squared modelfit measure.

References

A. Gelman, B. Goodrich, J. Gabry and A. Vehtari (2019). R-squared for Bayesian Regression Models. *The American Statistician* 73(3), 307-309.

binomial_control

Set computational options for the sampling algorithms

Description

Set computational options for the sampling algorithms

Usage

```
binomial_control(PG.approx = TRUE, PG.approx.m = -2L, probit.HaarPXDA = TRUE)
```

Arguments

<code>PG.approx</code>	whether Poly-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, <code>BayesLogit::rpg</code> is used, which is exact for some values of the shape parameter.
<code>PG.approx.m</code>	if <code>PG.approx=TRUE</code> , the number of explicit gamma draws in the sum-of-gammas representation of the Poly-Gamma distribution. The remainder (infinite) convolution is approximated by a single moment-matching gamma draw. Special values are: <code>-2L</code> for a default choice depending on the value of the shape parameter balancing performance and accuracy, <code>-1L</code> for a moment-matching normal approximation, and <code>0L</code> for a moment-matching gamma approximation.
<code>probit.HaarPXDA</code>	only used for binomial models with probit link. If supported, <code>probit.HaarPXDA=TRUE</code> and the Haar PX-DA sandwich step is added to the Albert-Chib data augmentation scheme for probit multilevel models. This will usually result in a faster mixing MCMC algorithm. Currently supported when all coefficients are sampled in a single Gibbs block and all coefficients' prior means equal zero.

Value

A list with computational options.

<code>brt</code>	<i>Create a model component object for a BART (Bayesian Additive Regression Trees) component in the linear predictor</i>
------------------	--

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,
  fastPredict = TRUE,
  ...
)
```


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 <code>environment(formula)</code> .
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.
fastPredict	if TRUE (default) a fast version of prediction based on the stored BART trees is used. Option FALSE is available mainly for testing purposes.
...	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. George 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

```
## Not run:
if (requireNamespace("dbarts", quietly=TRUE)) {
# generate data, based on an example in Friedman (1991)
gendat <- function(n=150L, 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),
  family = f_gaussian(var.prior=pr_invchisq(df=3, scale=var(train$y))),
```

```

    data = train
  )
  sim <- MCMCsim(sampler, n.chain=2, burnin=200, n.iter=400, thin=2,
    store.all=TRUE, verbose=FALSE)
  (summ <- summary(sim))
  plot(train$mu, summ$bart[, "Mean"]); abline(0, 1)
  pred <- predict(sim, newdata=test,
    iters=sample(seq_len(n_draws(sim)), 50),
    show.progress=FALSE
  )
  (summpred <- summary(pred))
  plot(test$mu, summpred[, "Mean"]); abline(0, 1)
}

## End(Not run)

```

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	<i>Set options for Cholesky decomposition</i>
--------------	---

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.

cMVN_control	<i>Set options for fast sampling from a degenerate multivariate normal distribution</i>
--------------	---

Description

Set options for fast sampling from a degenerate multivariate normal distribution

Usage

```
cMVN_control(
  eps1 = sqrt(.Machine$double.eps),
  eps2 = sqrt(.Machine$double.eps),
  chol.control = chol_control()
)
```

Arguments

eps1	scalar parameter to control numerical robustness against singularity of Q .
eps2	scalar parameter associated with the constraint part to control numerical robustness.
chol.control	options for Cholesky decomposition, see chol_control .

Value

A list of options used by the constrained MVN sampler.

combine_chains	<i>Combine multiple mcdraws objects into a single one by combining their chains</i>
----------------	---

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	<i>Combine multiple mcdraws objects into a single one by combining their draws</i>
---------------	--

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.

<code>computeDesignMatrix</code>	<i>Compute a list of design matrices for all terms in a model formula, or based on a sampler environment</i>
----------------------------------	--

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

<code>formula</code>	model formula.
<code>data</code>	data frame to be used in deriving the design matrices.
<code>labels</code>	if <code>TRUE</code> , 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)
```

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, control=NULL) First-order autoregressive correlation structure among the levels of f. Argument phi can be a single numerical value of the autoregressive parameter, or an appropriate prior specification if phi should be inferred. If not supplied, a uniform prior on (-1, 1] is assumed. 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, graph, snap, queen) CAR spatial correlation. Argument graph can either be an object of (S4) class SpatialPolygonsDataFrame or an object of (S3) class sf. The latter can be obtained, e.g., by reading in a shape file using function [st_read](#). Arguments snap and queen are passed to [poly2nb](#), which computes a neighbours list. Alternatively, a neighbours list object of class nb can be passed directly as argument graph.

splines(f, knots, degree) P-splines, i.e. penalised 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, control = NULL)

season(name, period)

spatial(
  name,
  graph = NULL,
  snap = sqrt(.Machine$double.eps),
  queen = TRUE,
  poly.df = NULL,
  derive.constraints = FALSE
)

splines(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	prior distribution, or fixed value, for an autoregressive parameter. The default is a uniform prior over the interval $[-1, 1]$. A single numeric value is interpreted as a fixed value, corresponding to a degenerate prior, which can also be specified as <code>pr_fixed(value)</code> . Alternatively, <code>link{pr_truncnormal}</code> can be used to specify a truncated normal prior.
control	options for Metropolis-Hastings sampling from the conditional posterior for an autoregressive parameter. These options can be set using function <code>set_MH</code> . Supported proposal types are "TN" and "RWTN". By default an independence truncated normal proposal (<code>type="TN"</code>), or a random walk truncated normal proposal (<code>type="RWTN"</code>) with adaptive scale initialised at 0.025 is used, depending on whether the specified random effects' distribution is Gaussian or non-Gaussian.
period	a positive integer specifying the seasonal period.
graph	either a spatial object of class <code>SpatialPolygons</code> , <code>sf</code> , <code>sfc</code> , or a neighbours list of class <code>nb</code> .
snap	passed to <code>poly2nb</code> . Ignored if <code>graph</code> is a neighbours list.
queen	passed to <code>poly2nb</code> . Ignored if <code>graph</code> is a neighbours list.
poly.df	a spatial data frame. DEPRECATED, use argument <code>graph</code> instead.
derive.constraints	whether to derive the constraint matrix for an IGMRF model component numerically from the precision matrix. The use of <code>derive.constraints</code> in function <code>spatial</code> is DEPRECATED, as it is no longer needed.
knots	passed to <code>splineDesign</code> .

degree	passed to <code>splineDesign</code> .
D	custom incidence matrix.
Q	custom precision matrix.
R	custom restriction matrix.

References

- B. Allevius (2018). On the precision matrix of an irregularly sampled AR(1) process. arXiv:1801.03791v2.
- H. Rue and L. Held (2005). Gaussian Markov Random Fields. Chapman & Hall/CRC.

Examples

```
## Not run:
# 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, graph=nc), name="vs"),
    family=f_gaussian(var.prior = 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, graph=nc), name="vs"),
    data=nc
  )
  # increase burnin and n.iter below to improve MCMC convergence
  sim <- MCMCsim(sampler, store.all=TRUE, burnin=200, n.iter=500, 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")
}

## End(Not run)
```

create_cMVN_sampler	<i>Set up a function for direct sampling from a constrained multivariate normal distribution</i>
---------------------	--

Description

Set up a function for direct sampling from a constrained multivariate normal distribution

Usage

```
create_cMVN_sampler(
  D = NULL,
  Q = NULL,
  update.Q = FALSE,
  R = NULL,
  r = NULL,
  eps1 = sqrt(.Machine$double.eps),
  eps2 = sqrt(.Machine$double.eps),
  chol.control = chol_control(perm = TRUE)
)
```

Arguments

D	factor of precision matrix Q such that $Q=D'D$.
Q	precision matrix.
update.Q	whether to update (D and) Q for each draw.
R	equality restriction matrix.
r	rhs vector for equality constraints $R'x = r$, where R' denotes the transpose of R.
eps1	scalar parameter to control numerical robustness against singularity of Q. Currently ignored in case of no constraints.
eps2	scalar parameter associated with the constraint part to control numerical robustness.
chol.control	options for Cholesky decomposition, see chol_control .

Value

An environment with precomputed quantities and a method 'draw' for sampling from a multivariate normal distribution subject to equality constraints.

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 = NULL,
  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'. Alternatively, functions starting with 'f_' followed by the family name can be used to specify the sampling distribution and possibly further options. See f_gaussian , f_binomial , f_multinomial , f_negbinomial , f_poisson , f_gamma and f_gaussian_gamma . For categorical or multinomial data, use family="multinomial" or family=f_multinomial() where the second form allows to additionally specify the number of trials (if not equal to 1). A stick-breaking representation of the multinomial distribution is used for model fitting, 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	NO LONGER USED. Please use f_binomial to specify the numbers of trials.
ry	NO LONGER USED. Please use f_negbinomial to specify further options for the negative binomial sampling distribution.
r.mod	NO LONGER USED. Please use f_negbinomial to specify further options for the negative binomial sampling distribution.
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. DEPRECATED, please use f_gaussian to specify variance options. In particular, a fixed scalar variance parameter with value 1 can be specified with var.prior=pr_fixed(value=1).
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, generalised 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. DEPRECATED, please use f_gaussian to specify variance options. In particular, to change the default prior for the scalar variance parameter use argument var.prior.
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. DEPRECATED, please use f_gaussian , in particular its precision argument, to specify unequal variances, or a non-diagonal precision 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 modelled 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. DEPRECATED, please use f_gaussian , in particular its var.model argument, to specify a variance model.

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. For example, when a log-transformation is used on response vector y , the vector $-\log(y)$ should be supplied. DEPRECATED, this argument has moved to f_gaussian .
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 <code>list(name_1=X_1, ...)</code> where the names refer to the model component names and predictions are computed by summing $X_i \% \% p[[name_i]]$. Alternatively, <code>linpred="fitted"</code> 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 <code>linpred</code> . 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 <code>control</code> 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\(...\)](#). As usual, the left hand side of formula specifies the response variable. For binomial and multinomial models see their respective help pages ([f_binomial](#), [f_multinomial](#)) for different ways to specify the response variable. If no response variable is specified, `create_sampler` only sets up infrastructure to sample from the model parameters' prior distributions.

To specify family-specific parameters or options, see the help pages for the various family specification functions, e.g. [f_gaussian](#) or [f_negbinomial](#).

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.
- J.S. Liu and Y.N. Wu (1999). Parameter expansion for data augmentation. *Journal of the American Statistical Association*, 94(448), 1264-1274.
- P.A. Parker, S.H. Holan and R. Janicki (2024). Conjugate Modeling Approaches for Small Area Estimation with Heteroscedastic Structure. *Journal of Survey Statistics and Methodology* 12(4), 1061-1080.
- 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.

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	<i>Set up a sampler object for sampling from a possibly truncated and degenerate multivariate normal distribution</i>
---------------------	---

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,
  constraints = NULL,
  check.constraints = FALSE,
  method = NULL,
  reduce = NULL,
  chol.control = chol_control(),
  debug = FALSE
)
```

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. Currently only supported by methods 'direct' and 'HMC'.
update.mu	whether mu is updated for each draw. By default equal to update.Q. Currently only supported by methods 'direct' and 'HMC'.
name	name of the TMVN vector parameter.
coef.names	optional labels for the components of the vector parameter.
constraints	optional linear equality and/or inequality constraints. Use function set_constraints to specify the constraint matrices and right-hand sides.
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 <code>m_direct</code> , <code>m_HMC</code> , <code>m_HMC_ZigZag</code> , <code>m_Gibbs</code> or <code>m_softTMVN</code> 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 .
debug	if TRUE a breakpoint is set at the beginning of the TMVN sampling function draw.

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 (2024). Zigzag path connects two Monte Carlo samplers: Hamiltonian counterpart to a piecewise deterministic Markov process. *Journal of the American Statistical Association*, 1-13.
- 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 (2019). The Soft Multivariate Truncated Normal Distribution with Applications to Bayesian Constrained Estimation. *arXiv:1807.09155v2*.
- 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
C <- set_constraints(S=S) # create a constraints object
# 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), constraints=C, method="HMC")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 2. exact Hamiltonian Monte Carlo with Laplace momentum (Nishimura et al., 2021)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), constraints=C, method="HMCZigZag")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 3. Gibbs sampling approach (Rodriguez-Yam et al., 2004)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), constraints=C, method="Gibbs")
sim <- MCMCsim(sampler, burnin=500, n.iter=2000, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
# 4. soft TMVN approximation (Souris et al., 2018)
sampler <- create_TMVN_sampler(Q=diag(2), mu=c(4, 4), constraints=C, method="softTMVN")
sim <- MCMCsim(sampler, n.iter=600, verbose=FALSE)
summary(sim)
plot(as.matrix(sim$x), pch=".")
```

f_binomial

Specify a binomial sampling distribution

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a binomial sampling distribution.

Usage

```
f_binomial(
  link = c("logit", "probit"),
  n.trial = NULL,
  control = binomial_control()
)
```

Arguments

link the name of a link function. Currently the only allowed link functions for the binomial distribution are "logit" (default) and "probit".

n.trial	the number of binomial trials. This can be specified either as a formula for a variable number of trials, or as a scalar value for a common number of trials for all units.
control	a list with computational options. These options can be specified using function binomial_control .

Details

For the binomial family, the left hand side of the formula argument of [create_sampler](#) can be specified in several ways (similar to the options allowed by [glm](#) for binomial family):

- 1** as a factor, character, or boolean variable, say *y*. In this case the first level of `as.factor(y)` is interpreted as 'failures' and all other values as 'successes'. This option can only be used for binary data.
- 2a** as a numeric or integer vector with no values strictly between 0 and 1. The values are then interpreted as numbers of successes. This requires specifying the number of trials through argument `n.trial`, unless the data is binary (values 0 and 1 only). For non-integer values a warning is issued.
- 2b** as a numeric vector with values between 0 and 1. The values are then interpreted as the proportion of the number of successes. This requires specifying the number of trials through argument `n.trial`.
- 3** as a two-column integer matrix. The first column is interpreted as the number of successes, the second column as the number of failures.

Value

A family object.

Examples

```
y <- c(TRUE, FALSE, TRUE)
sampler <- create_sampler(y ~ 1, family=f_binomial())
# logistic binary regression is the default, and may also be
# specified as family="binomial"
sim <- MCMCsim(sampler, n.chain=2, n.iter=100, burnin=50, verbose=FALSE)
summary(predict(sim, newdata=data.frame(id=1)))

y <- c(0, 0, 1, 1, 0, 0, 0, 0, 1)
sampler <- create_sampler(y ~ 1, family=f_binomial(link="probit"))
sim <- MCMCsim(sampler, n.chain=2, n.iter=100, burnin=50, verbose=FALSE)
summary(predict(sim, newdata=data.frame(id=1)))

sampler <- create_sampler(c(0.2, 0.3, 0.5, 0.0) ~ 1, family=f_binomial(n.trial=10))
# is interpreted as
sampler <- create_sampler(c(2, 3, 5, 0) ~ 1, family=f_binomial(n.trial=10))
sim <- MCMCsim(sampler, n.chain=2, n.iter=100, burnin=50, verbose=FALSE)
summary(predict(sim, newdata=data.frame(id=1)))

n <- 1000
dat <- data.frame(
```

```

x = runif(n),
g = factor(sample(1:10, n, replace=TRUE)),
trials = sample(1:5, n, replace=TRUE)
)
v <- rnorm(10)
dat$y <- rbinom(n, size=dat$trials, prob=1 / (1 + exp(-(1 - dat$x + v[dat$g]))))
sampler <- create_sampler(y ~ x + (1|g), data=dat, family=f_binomial(n.trial = ~ trials))
# alternatively:
sampler <- create_sampler(cbind(y, trials - y) ~ x + (1|g), data=dat, family="binomial")
sim <- MCMCsim(sampler, store.all=TRUE, burnin=200, n.iter=300, n.chain=2, verbose=FALSE)
summ <- summary(sim)
plot(v, summ$gen2[, "Mean"]); abline(0, 1)

```

f_gamma

Specify a Gamma sampling distribution

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a Gamma sampling distribution.

Usage

```

f_gamma(
  link = "log",
  shape.vec = ~1,
  shape.prior = pr_gamma(0.1, 0.1),
  control = set_MH(type = "RWLN", scale = 0.2, adaptive = TRUE)
)

```

Arguments

link	the name of a link function. Currently the only allowed link function for the gamma distribution is "log".
shape.vec	optional formula specification of unequal shape parameter.
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 <code>pr_gamma(shape=0.1, rate=0.1)</code> .
control	options for the Metropolis-Hastings algorithm employed in case the shape parameter is to be inferred. Function set_MH can be used to change the default options. The two choices of proposal distribution type supported are "RWLN" 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 often quite accurate and efficient.

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.

Examples

```
## Not run:
n <- 3000
m <- 25
dat <- data.frame(
  x = rnorm(n),
  g = factor(sample(1:m, n, replace=TRUE), levels=1:m)
)
v <- rnorm(m, sd=0.6)
alpha <- 1
mu <- exp(with(dat, 1 - 0.5*x + v[g]))
dat$y <- rgamma(n, shape=alpha, rate=alpha/mu)

sampler <- create_sampler(
  y ~ reg(~ x, name="beta") +      # fixed effects
    gen(factor = ~ g, name="v"),  # random intercepts
  data=dat, family="gamma"
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
waic(sim)
summary(sim)

bayesplot::mcmc_recover_intervals(as.array(sim$gamma_shape_), alpha)
bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_recover_intervals(as.array(sim$v_sigma), 0.6)
yrep <- predict(sim, iters=sample(1:1000, 10))
bayesplot::pp_check(dat$y, as.matrix(yrep), bayesplot::ppc_dens_overlay) + ggplot2::xlim(0, 25)

## End(Not run)
```

f_gaussian

Specify a Gaussian sampling distribution

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a Gaussian sampling distribution.

Usage

```
f_gaussian(
  link = "identity",
  var.prior = pr_invchisq(df = 0, scale = 1),
  var.vec = ~1,
  prec.mat = NULL,
  var.model = NULL,
  logJacobian = NULL,
  control = gaussian_control()
)
```

Arguments

link	the name of a link function. Currently the only allowed link function for a Gaussian distribution is "identity".
var.prior	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, generalised 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.
var.vec	a formula to specify unequal variances, i.e. heteroscedasticity. The default corresponds to equal variances.
prec.mat	a possibly non-diagonal positive-definite symmetric matrix interpreted as the precision matrix, i.e. inverse of the covariance matrix. If this argument is specified <code>var.vec</code> is ignored.
var.model	a formula specifying the terms of a variance model in the case of a Gaussian likelihood. Several types of terms are supported: a regression term for the log-variance specified with vreg(...) , and a term vfac(...) for multiplicative modelled 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. In addition, reg and gen can be used to specify regression or random effect terms. In that case the prior distribution of the coefficients is not exactly normal, but instead Multivariate Log inverse Gamma (MLiG), see also pr_MLiG .
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. For example, when a log-transformation is used on response vector y , the vector $-\log(y)$ should be supplied.
control	a list with computational options. These options can be specified using function gaussian_control .

Value

A family object.

Examples

```
## Not run:
n <- 4000
m <- 25
dat <- data.frame(
  x = rnorm(n),
  g = factor(sample(1:m, n, replace=TRUE), levels=1:m)
)
v <- rnorm(m, sd=0.6)
dat$y <- rnorm(n, mean = with(dat, 1 - 0.5*x + v[g]), sd=0.4)

sampler <- create_sampler(
  y ~ x + (1|g), data=dat
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
summary(sim)

# more explicit specification, allowing non-default names, priors etc.
sampler <- create_sampler(
  y ~ reg(~ x, name="beta") + gen(~1, factor = ~ g, name="v"),
  data=dat,
  family = f_gaussian(var.prior = pr_fixed(value = 0.4^2))
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
summary(sim)
bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_recover_hist(as.array(sim$v_sigma), 0.6)
bayesplot::mcmc_recover_scatter(as.array(sim$v), v)

# heteroscedastic data
dat$y <- rnorm(n,
  mean = with(dat, 1 - 0.5*x + v[g]),
  sd = with(dat, exp(0.5*(0.2 + 0.5*x)))
)

sampler <- create_sampler(
  y ~ reg(~ x, name="beta") + gen(~1, factor = ~ g, name="v"),
  data=dat,
  family = f_gaussian(
    var.prior = pr_fixed(value = 1),
    var.model = ~ 1 + x
  )
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
compute_WAIC(sim)
```

```
summary(sim)
bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_recover_intervals(as.array(sim$vreg1), c(0.2, 0.5))
bayesplot::mcmc_recover_hist(as.array(sim$v_sigma), 0.6)
bayesplot::mcmc_recover_scatter(as.array(sim$v), v)

## End(Not run)
```

f_gaussian_gamma

Specify a Gaussian-Gamma sampling distribution

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a Gaussian-Gamma sampling distribution, i.e., a Gaussian sampling distribution whose variances are observed subject to error according to a Gamma distribution.

Usage

```
f_gaussian_gamma(
  link = "identity",
  var.model,
  gaussian.control = gaussian_control(),
  ...
)
```

Arguments

link	the name of a link function. Currently the only allowed link function for this distribution family is "identity".
var.model	a formula specifying the terms of a variance model. The left-hand side of the formula should specify the observed variances, unless the family object is used for data generation only. Several types of model terms on the right-hand side of the formula are supported: a regression term for the log-variance specified with vreg(...) , and a term vfac(...) for multiplicative modelled factors at a certain level specified by a factor variable. In addition, reg and gen can be used to specify regression or random effect terms. In that case the prior distribution of the coefficients is not normal, but Multivariate Log inverse Gamma (MLiG), see also pr_MLiG .
gaussian.control	a list with computational options passed to f_gaussian .
...	further arguments passed to f_gamma .

Value

A family object.

f_multi	<i>Specify a multi-response sampling distribution</i>
---------	---

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a multi-response sampling distribution. The current implementation requires the data to be in long format, disregarding the possibility that observations may be on the same units.

Usage

```
f_multi(fam.list, fam.ind)
```

Arguments

fam.list	a named list of response types.
fam.ind	a formula specifying a character or factor variable indicating which observations are of each response type specified in fam.list.

Value

A family object.

f_multinomial	<i>Specify a multinomial sampling distribution</i>
---------------	--

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a multinomial sampling distribution. This includes the special case of categorical (multinoulli) data.

Usage

```
f_multinomial(  
  link = "logit",  
  n.trial = NULL,  
  K = NULL,  
  control = multinomial_control()  
)
```

Arguments

link	the name of a link function. Currently the only allowed link function for the multinomial distribution is "logit".
n.trial	the number of multinomial trials. This can be specified either as a formula for a variable number of trials, or as a scalar value for a common number of trials for all units.
K	number of categories for multinomial model; only used for prior predictive sampling.
control	a list with computational options. These options can be specified using function <code>multinomial_control</code> .

Details

For the multinomial family, the left hand side of the formula argument of `create_sampler` can be specified in one of the following ways:

- 1 as a single factor, character, boolean or integer variable, say `y`. The categories then correspond to the levels of `as.factor(y)`. This option can only be used for categorical data.
- 2 as a $n \times (K-1)$ numeric matrix with values between 0 and 1, where n is the number of observations and K the number of categories. The values are interpreted as proportions of observations in each category. This requires specifying the number of multinomial trials through argument `n.trial`.
- 3 a K -column integer matrix, where K is the number of categories, each column containing the number of 'successes' for the corresponding category.

Value

A family object.

Examples

```
y <- factor(sample(c("a", "b", "c"), 800, prob=c(0.3, 0.5, 0.2), replace=TRUE))
sampler <- create_sampler(y ~ 0 + cat_, family=f_multinomial())
sim <- MCMCsim(sampler, n.chain=2, burnin=200, n.iter=300, verbose=FALSE)
summary(sim)
summary(predict(sim, newdata=data.frame(id=1:5)))
```

f_negbinomial

Specify a negative binomial sampling distribution

Description

This function can be used in the family argument of `create_sampler` or `generate_data` to specify a negative binomial sampling distribution.

Usage

```
f_negbinomial(
  link = "log",
  shape.vec = ~1,
  inv.shape.prior = pr_invchisq(df = 1, scale = 1),
  control = negbinomial_control()
)
```

Arguments

link	the name of a link function. Currently the only allowed link function for the negative binomial sampling distribution is "log".
shape.vec	optional formula specification of unequal shape values. The negative binomial (vector) shape parameter is then equal to this vector of shape values, multiplied by the scalar shape parameter, whose prior is specified through <code>inv.shape.prior</code> .
inv.shape.prior	Prior on the (scalar) <i>reciprocal</i> shape parameter, i.e. the overdispersion parameter. Supported prior distributions are <code>pr_fixed</code> with a default value of 1, <code>pr_invchisq</code> and <code>pr_gig</code> . The current default is <code>pr_invchisq(df=1, scale=1)</code> .
control	a list with computational options. These options can be specified using function <code>negbinomial_control</code> .

Details

The negative binomial distribution with shape r and probability p has density

$$p(y|r, p) = \frac{\Gamma(y+r)}{y!\Gamma(r)} (1-p)^r p^y$$

with mean $\mu = E(y|r, p) = \frac{rp}{1-p}$ and variance $V(y|r, p) = \mu(1 + \mu/r)$. The second term of the variance can be interpreted as overdispersion with respect to a Poisson distribution, which would correspond to the limit $r \rightarrow \infty$. So the reciprocal shape $1/r$ is an overdispersion parameter, which typically is inferred. It is assigned a default prior, which may be changed through argument `inv.shape.prior`.

The only supported link function is "log". Strictly speaking the relation between mean μ and linear predictor η is

$$\log \mu = \log r + \log \frac{p}{1-p} = \log r + \eta$$

This way the likelihood function has the same form as that of logistic binomial regression, so that a Polya-Gamma data augmentation sampling algorithm can be employed. Note that the fact that the linear predictor η does not include $\log r$ effectively changes the interpretation of its intercept.

Value

A family object.

References

- 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.
- 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

```
## Not run:
# first generate some data based on a negative binomial structured additive regression model
n <- 1200
nT <- 50
dat <- data.frame(
  x = runif(n),
  t = factor(sample(1:nT, n, replace=TRUE), levels=1:nT)
)
# generate random walk over time t
v <- cumsum(rnorm(nT, sd=0.1))
v <- v - mean(v) # subtract mean to separate random walk from intercept
lp <- with(dat, 1 - 0.5*x + v[t])
shape <- 2 # inverse negative binomial dispersion parameter
dat$y <- rnbinom(n, size=shape, mu = shape * exp(lp))

# now fit a negative binomial model to this data
sampler <- create_sampler(
  y ~ reg(~ x, name="beta") + gen(factor = ~ RW1(t), name="v"),
  data=dat, family = f_negbinomial()
)
sim <- MCMCsim(sampler, store.all=TRUE, plot.trace=c("v_sigma", "negbin_shape_"))
(summ <- summary(sim))
loo(sim)
bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_scatter(as.array(sim$beta))
bayesplot::mcmc_recover_hist(as.array(sim$negbin_shape_), shape)
bayesplot::mcmc_recover_hist(as.array(sim$v_sigma), 0.1)
plot.ts(v); lines(summ$v[, "Mean"], col=2)
pred <- predict(sim, type="response")
summpred <- summary(pred)
# NB prediction with type="response" does not include factor shape
plot(exp(lp), summpred[, "Mean"]); abline(0, 1)
plot(dat$y, shape * summpred[, "Mean"]); abline(0, 1)
mean(dat$y); shape * mean(summpred[, "Mean"])
# posterior predictive p-values at the observation level:
ppred <- predict(sim, ppcheck=TRUE)
hist(attr(ppred, "ppp"))
# NB the peak at 1 is due to zeros in the data:
dat$y[attr(ppred, "ppp") == 1]
# as Pr(y_rep >= y) = 1 by construction for these cases

## End(Not run)
```

f_poisson

*Specify a Poisson sampling distribution***Description**

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a Poisson sampling distribution.

Usage

```
f_poisson(link = "log", control = poisson_control())
```

Arguments

link	the name of a link function. Currently the only allowed link function for the Poisson distribution is "log".
control	a list with computational options. These options can be specified using function poisson_control .

Value

A family object.

Examples

```
## Not run:
n <- 3000
m <- 25
dat <- data.frame(
  x = rnorm(n),
  g = factor(sample(1:m, n, replace=TRUE), levels=1:m)
)
v <- rnorm(m, sd=0.6)
dat$y <- rpois(n, lambda=exp(with(dat, 1 - 0.5*x + v[g])))

sampler <- create_sampler(
  y ~ reg(~ x, name="beta") +      # fixed effects
    gen(factor = ~ g, name="v"),  # random intercepts
  data=dat, family="poisson"
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
waic(sim)
summary(sim)

bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_recover_intervals(as.array(sim$v_sigma), 0.6)
yrep <- predict(sim, iters=sample(1:1000, 10))
bayesplot::pp_check(dat$y, as.matrix(yrep), bayesplot::ppc_dens_overlay)
```

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

f_student_t

Specify a Student-t sampling distribution

Description

This function can be used in the family argument of [create_sampler](#) or [generate_data](#) to specify a Student-t sampling distribution.

Usage

```
f_student_t(
  link = "identity",
  df.prior = pr_gamma(2, 0.1),
  control = student_t_control(),
  ...
)
```

Arguments

link	the name of a link function. Currently the only allowed link function for a Student-t distribution is "identity".
df.prior	prior specification for the (scalar) number of degrees of freedom of the Student-t sampling distribution. Currently allowed priors are gamma priors specified using pr_gamma , or a degenerate prior fixing the degrees of freedom parameter at a fixed value.
control	sampling options for the degrees of freedom parameter (if not fixed).
...	further arguments passed to f_gaussian . Note that var.prior, var.vec and var.model now refer to the Student-t scale parameters.

Value

A family object.

Examples

```
## Not run:
n <- 4000
m <- 25
dat <- data.frame(
  x = rnorm(n),
  g = factor(sample(1:m, n, replace=TRUE), levels=1:m)
)
v <- rnorm(m, sd=0.6)
dat$y <- with(dat, 1 - 0.5*x + v[g]) + 0.4 * rt(n, df=2.5)
```

```

sampler <- create_sampler(
  y ~ x + (1|g), data=dat, family="student_t"
)
sim <- MCMCsim(sampler, store.all=TRUE)
compute_DIC(sim)
waic(sim)
summary(sim)

#' # more explicit specification, allowing non-default names, priors etc.
sampler <- create_sampler(
  y ~ reg(~ x, name="beta") + gen(~1, factor = ~ g, name="v"),
  data=dat,
  family = f_student_t(df.prior=pr_gamma(2, 0.2))
)
sim <- MCMCsim(sampler, store.all=TRUE)
loo(sim)
summary(sim)
bayesplot::mcmc_recover_hist(as.array(sim$student_t_df), 2.5)
bayesplot::mcmc_recover_intervals(as.array(sim$beta), c(1, -0.5))
bayesplot::mcmc_recover_hist(as.array(sim$v_sigma), 0.6)
bayesplot::mcmc_recover_scatter(as.array(sim$v), v)

## End(Not run)

```

gaussian_control

Set computational options for the sampling algorithms

Description

Set computational options for the sampling algorithms

Usage

```
gaussian_control(block.V = TRUE)
```

Arguments

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.
---------	--

Value

A list with computational options.

gen	<i>Create a model component object for a generic random effects component in the linear predictor</i>
-----	---

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,
  priorA = NULL,
  strucA = GMRF_structure(),
  GMRFconstr = NULL,
  constraints0 = NULL,
  constraintsA = 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 modelled 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)`.

<code>remove.redundant</code>	whether redundant columns should be removed from the model matrix associated with <code>formula</code> . Default is <code>FALSE</code> .
<code>drop.empty.levels</code>	whether to remove factor levels without observations.
<code>X</code>	A (possibly sparse) design matrix. If <code>X</code> is specified, <code>formula</code> and <code>factor</code> are only used to derive the random effects' structured precision matrix.
<code>var</code>	the (co)variance structure among the varying effects defined by <code>formula</code> over the levels of the factors defined by <code>factor</code> . The default is <code>"unstructured"</code> , meaning that a full covariance matrix parametrization is used. For uncorrelated effects with unequal variances use <code>var="diagonal"</code> . For uncorrelated effects with equal variances use <code>var="scalar"</code> . In the case of a single varying effect there is no difference between these choices.
<code>prior</code>	the prior specification for the variance parameters of the random effects. These can currently be specified by a call to pr_invwishart in case <code>var="unstructured"</code> or by a call to pr_invchisq otherwise. See the documentation of those prior specification functions for more details.
<code>Q0</code>	precision matrix associated with <code>formula</code> . This can only be used in combination with <code>var="scalar"</code> .
<code>PX</code>	whether parameter expansion should be used. Default is <code>TRUE</code> , which applies parameter expansion with default options. The only exception is that for gamma sampling distributions the default is <code>FALSE</code> , 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 <code>formula</code>. The default is <code>TRUE</code> except when <code>var="scalar"</code>. If <code>FALSE</code> 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 <code>TRUE</code> for gaussian, student-t and gaussian-gamma families.

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 .
struca	this option can be used to modify the default structure encoded by factor to a 'bym2' or 'leroux' structure. See GMRF_structure for details.
GMRFconstr	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 Gaussian Markov Random Field model components, i.e. components with a singular precision matrix such as random walks or CAR spatial components.
constraints0	an optional set of linear (in)equality restrictions acting on the coefficients defined by formula, for each level defined by factor. The constraint matrices, which can be specified using function set_constraints , must have number of rows equal to the number of columns of the design matrix derived from formula, each column corresponding to a linear constraint. Currently only constraints with zero right-hand side are supported.
constraintsA	an optional set of linear (in)equality restrictions acting on the coefficients defined by factor, for each effect defined by formula. The constraint matrices, which can be specified using function set_constraints , must have number of rows equal to the number of levels defined by factor, each column corresponding to a linear constraint. The overall set of constraints imposed on the full vector of coefficients is determined by constraints0 and constraintsA together. If GMRFconstr=TRUE, these user-defined constraints (if any) are supplemented by equality constraints corresponding to the null-vectors of the singular precision matrix in case of an intrinsic Gaussian Markov Random Field. Currently only constraints with zero right-hand side are supported.
formula.gl	a formula of the form <code>~ glreg(...)</code> for group-level predictors around which the random effect component is hierarchically centred. See glreg for details.
a	only used in case the effects are MLiG distributed, as 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.
- 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	<i>Generate a data vector according to a model</i>
---------------	--

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 = NULL,
  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	NO LONGER USED; see create_sampler .
ry	NO LONGER USED; see create_sampler .
r.mod	NO LONGER USED; see create_sampler .
sigma.fixed	DEPRECATED; see create_sampler .
sigma.mod	DEPRECATED; see create_sampler .
Q0	DEPRECATED; see create_sampler .
formula.v	DEPRECATED; 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, prior=pr_normal(precision=10, mean=c(0, 1)), name="beta") +
  gen(factor = ~ g, name="v"),
  family=f_binomial(n.trial = ~ ny), data=dat
)
gd
plot(dat$x, gd$y)
```

gen_control	<i>Set computational options for the sampling algorithms used for a 'gen' model component</i>
-------------	---

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 α is large, such that the coefficients' prior is approximately normal.

Value

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

get_draw	<i>Extract a list of parameter values for a single draw</i>
----------	---

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	<i>Create a model object for group-level regression effects within a generic random effects component.</i>
-------	--

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 centring 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 <code>data</code> 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 <code>prior</code> instead, i.e. <code>prior = pr_normal(precision = Q0.value)</code> .
data	group-level data frame in which the group-level variables specified in <code>formula</code> 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 <code>'_gl'</code> .

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.

GMRF_structure

*Set up a GMRF structure for a generic model component***Description**

This function is used to specify a (non-default) GMRF structure to pass to argument `strucA` of function [gen](#).

Usage

```
GMRF_structure(
  type = c("default", "bym2", "leroux", "Leroux"),
  scale.precision = (type == "bym2"),
  prior = NULL,
  control = NULL
)
```

Arguments

<code>type</code>	one of "default", "bym2" or "leroux". The default choice corresponds to the precision matrix Q_A as implied or specified by argument <code>factor</code> of gen . Type "bym2" modifies the default structure to one with covariance matrix $\phi \tilde{Q}_A^- + (1 - \phi)I$ where $\tilde{Q}_A^-$ is the generalised inverse of Q_A , by default scaled such that the geometric mean of the marginal variances equals 1. Type "leroux" modifies the default structure to one with precision matrix $\phi Q_A + (1 - \phi)I$.
<code>scale.precision</code>	whether to scale the structured precision matrix. By default set to TRUE only for type "bym2".
<code>prior</code>	prior for the parameter ϕ in the "bym2" or "leroux" extension. Supported priors can be set using functions pr_fixed or pr_unif .
<code>control</code>	options for the Metropolis-Hastings sampler used to sample from the full conditional distribution of parameter ϕ in case of "bym2" or "leroux" extensions. If NULL a reasonable default configuration is used. A user can change these settings using function set_MH . Supported proposal distribution types are "RWTN", "RWN", "unif" and "beta".

Value

An environment defining the desired GMRF structure, for use by other package functions.

References

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.

A. Riebler, S.H. Sorbye, D. Simpson and H. Rue (2016). An intuitive Bayesian spatial model for disease mapping that accounts for scaling. *Statistical methods in medical research*, 25(4), 1145-1165.

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

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 <- mcmc_sae_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	<i>Fast matrix-vector multiplications</i>
---------------	---

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

<code>M</code>	a matrix of class <code>'matrix'</code> , <code>'dgCMatrix'</code> , <code>'dsCMatrix'</code> , <code>'tabMatrix'</code> , or <code>'ddiMatrix'</code> .
<code>v</code>	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	<i>Maximise the log-likelihood or log-posterior as defined by a sampler closure</i>
-------------------	---

Description

Maximise 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 optimisation 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"),
  family=f_gaussian(var.prior=pr_fixed(value=1)), 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")
```


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

<code>x</code>	a component of an <code>mcdraws</code> object corresponding to a scalar or vector model parameter. In the case of <code>to_draws_array</code> <code>x</code> can also be a full <code>mcdraws</code> object.
<code>components</code>	optional character vector of names of draws components in an <code>mcdraws</code> object. This can be used to select a subset of components to convert to <code>draws_array</code> format.
<code>...</code>	currently ignored.
<code>colnames</code>	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

```
## Not run:
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=200)
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, burnin=100, n.chain=2, n.iter=300)
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 <- mcmc_sae_example()
plot(ex$dat$fT, ex$dat$y)
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, burnin=100, n.chain=2, n.iter=200, 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)

## End(Not run)
```

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 <code>create_sampler</code> .
from.prior	whether to sample from the prior. By default <code>from.prior=FALSE</code> 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 <code>sampler\$store_default</code> .
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 <code>sampler\$store_mean_default</code> exists it provides the default.

<code>store.sds</code>	if TRUE store for all parameters in <code>store.mean</code> , besides the mean, also the standard deviation. Default is FALSE.
<code>to.file</code>	vector of names of parameters to write to file.
<code>filename</code>	name of file to write parameter draws to. Each named parameter is written to a separate file, named <code>filename_parametername</code> .
<code>write.single.prec</code>	Whether to write to file in single precision. Default is FALSE.
<code>verbose</code>	if FALSE no output is sent to the screen during the simulation. TRUE by default.
<code>n.progress</code>	update diagnostics and plots after so many iterations.
<code>trace.convergence</code>	vector of names of parameters for which Gelman-Rubin R-hat diagnostics are printed to the screen every <code>n.progress</code> iterations.
<code>stop.on.convergence</code>	if TRUE stop the simulation if the R-hat diagnostics for all parameters in <code>trace.convergence</code> are less than <code>convergence.bound</code> .
<code>convergence.bound</code>	threshold used with <code>stop.on.convergence</code> .
<code>plot.trace</code>	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. <code>"beta[2]"</code> .
<code>add.to.plot</code>	if TRUE the plot is updated every <code>n.progress</code> iterations, otherwise a new plot (with new scales) is created after every <code>n.progress</code> iterations.
<code>plot.type</code>	default is "l" (lines).
<code>n.cores</code>	the number of cpu cores to use. Default is 1, i.e. no parallel computation. If an existing cluster <code>cl</code> is provided, <code>n.cores</code> will be set to the number of workers in that cluster.
<code>cl</code>	an existing cluster can be passed for parallel computation. If NULL and <code>n.cores</code> > 1, a new cluster is created.
<code>seed</code>	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
<code>export</code>	a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in <code>pred</code> 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.

get_labels function of a single variable `par.name` that returns labels for the components of the corresponding vector parameter.

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
```

mc_offset

Create a model component object for an offset, i.e. fixed, non-parametrised term 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

```
mc_offset(formula, value = NULL, name = "")
```

Arguments

formula	model formula.
value	alternative specification of an offset as a single scalar value that is the same for each data unit.
name	the name of the model component. This name is used in the output of the MCMC simulation function <code>MCMCsim</code> . By default the name will be 'mc_offset' with the number of the model term attached.

Value

An model component object with data and methods needed for dealing with an offset term in model estimation, and prior and posterior prediction, intended for internal use by other package functions.

mec	<i>Create a model component object for a regression (fixed effects) component in the linear predictor with measurement errors in quantitative covariates</i>
-----	--

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 σ^2 .

Usage

```
mec(
  formula = ~1,
  sparse = NULL,
  X = NULL,
  V = NULL,
  prior = NULL,
  scale.prior = TRUE,
  constraints = NULL,
  Q0 = NULL,
  b0 = 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 $\sim (x1 \mid V.x1) + (x2 \mid V.x2) + \dots$ should be used where $x1, x2, \dots$ are the names of (quantitative) predictors and $V.x1, V.x2, \dots$ are the names of the variables holding the corresponding measurement error variances. If only the predictors are specified the formula has the usual form $\sim x1 + x2 + \dots$. In that case variances should be specified using argument <code>V</code> . All variable names are looked up in the data frame passed as <code>data</code> argument to <code>create_sampler</code> or <code>generate_data</code> , or in <code>environment(formula)</code> .
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 <code>X</code> 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 <code>pr_normal</code> .
scale.prior	whether the normal prior's scale includes the model's global residual scale as a factor. By default TRUE, but only relevant for a proper normal prior specified by <code>pr_normal</code> for families "gaussian", "student-t" and "gaussian-gamma".
constraints	optional linear equality and/or inequality constraints imposed on the vector of regression coefficients. Use function <code>set_constraints</code> to specify the constraint matrices and right-hand sides.
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 <code>prior</code> instead, i.e. <code>prior = pr_normal(mean = b0.value, precision = Q0.value)</code> .
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 <code>prior</code> instead, i.e. <code>prior = pr_normal(mean = b0.value, precision = Q0.value)</code> .
name	the name of the model component. This name is used in the output of the MCMC simulation function <code>MCMCsim</code> . 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, name="ME") + gen(factor=~local_),
  family=f_gaussian(var.prior=pr_fixed(1), var.vec=~psi),
  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$ME_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

<code>x</code>	an object of class <code>mcdraws</code> .
<code>use.pV</code>	whether half the posterior variance of the deviance should be used as an alternative estimate of the effective number of model parameters for DIC.
<code>diagnostic</code>	whether vectors of log-pointwise-predictive-densities and pointwise contributions to the WAIC effective number of model parameters should be returned.
<code>batch.size</code>	number of data units to process per batch.
<code>show.progress</code>	whether to show a progress bar.
<code>cl</code>	an existing cluster can be passed for parallel computation. If <code>cl</code> is provided, <code>n.cores</code> will be set to the number of workers in that cluster. If <code>NULL</code> and <code>n.cores > 1</code> , a new cluster is created.
<code>n.cores</code>	the number of cpu cores to use. Default is one, i.e. no parallel computation.
<code>by.unit</code>	if <code>TRUE</code> the computation is carried out unit-by-unit, which is slower but uses much less memory.
<code>...</code>	Other arguments, passed to <code>loo</code> . Not currently used by <code>waic.mcdraws</code> .
<code>r_eff</code>	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 (2024). Pareto smoothed importance sampling. [arXiv:1507.02646v9](https://arxiv.org/abs/1507.02646v9).
- 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
)
```

Arguments

<code>formula</code>	model formula.
<code>data</code>	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.
<code>contrasts.arg</code>	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.
<code>drop.unused.levels</code>	whether empty levels of individual factor variables should be removed.
<code>sparse</code>	if TRUE a sparse matrix of class <code>dgCMatrix</code> is returned. This can be efficient for large datasets and a model containing categorical variables with many categories. If <code>sparse=NULL</code> , the default, whether a sparse or dense model matrix is returned is based on a simple heuristic.
<code>drop0</code>	whether to drop any remaining explicit zeros in resulting sparse matrix.
<code>catsep</code>	separator for concatenating factor variable names and level names. By default it is the empty string, reproducing the labels of <code>model.matrix</code> .
<code>by</code>	a vector by which to aggregate the result.
<code>tabM</code>	if TRUE return a list of <code>tabMatrix</code> objects.

Value

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

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

Description

Set computational options for the sampling algorithms

Usage

```
multinomial_control(PG.approx = TRUE, PG.approx.m = -2L)
```

Arguments

<code>PG.approx</code>	whether Polya-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, <code>BayesLogit::rpg</code> is used, which is exact for some values of the shape parameter.
<code>PG.approx.m</code>	if <code>PG.approx=TRUE</code> , 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: <code>-2L</code> for a default choice depending on the value of the shape parameter balancing performance and accuracy, <code>-1L</code> for a moment-matching normal approximation, and <code>0L</code> for a moment-matching gamma approximation.

Value

A list with computational options.

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

Description

Set computational options for the sampling algorithms

Usage

```
negbinomial_control(PG.approx = TRUE, PG.approx.m = -2L, CRT.approx.m = 20L)
```

Arguments

<code>PG.approx</code>	whether Poly-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, <code>BayesLogit:rpg</code> is used, which is exact for some values of the shape parameter.
<code>PG.approx.m</code>	if <code>PG.approx=TRUE</code> , the number of explicit gamma draws in the sum-of-gammas representation of the Poly-Gamma distribution. The remainder (infinite) convolution is approximated by a single moment-matching gamma draw. Special values are: <code>-2L</code> for a default choice depending on the value of the shape parameter balancing performance and accuracy, <code>-1L</code> for a moment-matching normal approximation, and <code>0L</code> for a moment-matching gamma approximation.
<code>CRT.approx.m</code>	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 computational options for the sampling algorithm.

`n_chains-n_draws-n_vars`
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

```
n_chains(obj)
```

```
n_draws(obj)
```

```
n_vars(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)
n_chains(sim); n_chains(sim$beta)
n_draws(sim); n_draws(sim$beta)
n_vars(sim$beta); n_vars(sim$sigma_); n_vars(sim$llh_); n_vars(sim$v)
plot(sim, "beta")
n_chains(subset(sim$beta, chains=1:2))
n_draws(subset(sim$beta, draws=sample(1:n_draws(sim), 100)))
n_vars(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	<i>Trace, density and autocorrelation plots for (parameters of a) draws component (dc) object</i>
---------	---

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

plot.mcdraws	<i>Trace, density and autocorrelation plots</i>
--------------	---

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	<i>Plot a set of model coefficients or predictions with uncertainty intervals based on summaries of simulation results or other objects.</i>
-----------	--

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 numeric 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)
  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)
  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"))

```

poisson_control

*Set computational options for the sampling algorithms***Description**

Set computational options for the sampling algorithms

Usage

```
poisson_control(nb.shape = 100, PG.approx = TRUE, PG.approx.m = -2L)
```

Arguments

nb.shape	shape parameter of the negative binomial distribution used internally to approximate the Poisson distribution. This should be set to a relatively large value (default is 100), corresponding to negligible overdispersion, to obtain a good approximation to the Poisson sampling distribution. However, note that very large values may cause slow MCMC exploration of the posterior distribution.
PG.approx	whether Polya-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, <code>BayesLogit::rpg</code> is used, which is exact for some values of the shape parameter.
PG.approx.m	if <code>PG.approx=TRUE</code> , 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: <code>-2L</code> for a default choice depending on the value of the shape parameter balancing performance and accuracy, <code>-1L</code> for a moment-matching normal approximation, and <code>0L</code> for a moment-matching gamma approximation.

Value

A list with computational options for the sampling algorithm.

posterior-moments	<i>Get means or standard deviations of parameters from the MCMC output in an mcdraws object</i>
-------------------	---

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	<i>Generate draws from the predictive distribution</i>
-----------------	--

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"),
  weights = 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 models type="response" returns the simulations of the latent probabilities, and for negative binomial models the exponentiated linear predictor, which differs from the mean by a factor equal to the shape parameter. 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.
weights	an optional formula specifying a vector of nonnegative weights. Weights are only used for data-generating prediction (type = "data"). A weight w_i means that the <i>sum</i> of w_i individual predictions is generated for each unit i. For example, for the poststratification step of Multilevel Regression and Poststratification (MRP) the weights are the population counts and the units are the unique combinations of all auxiliary variables used in the model, typically stored in a single <code>data.frame</code>.
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 must be named 'p' and 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 <code>fun.</code>
ppcheck	if TRUE, function <code>fun.</code> 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 <code>to.file=TRUE</code>.
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 <code>cl</code> is provided, <code>n.cores</code> will be set to the number of workers in that cluster.
cl	an existing cluster can be passed for parallel computation. If NULL and <code>n.cores > 1</code>, 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 <code>fun.</code> 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	<i>Display a summary of a dc object</i>
------------------	---

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_beta`*Create an object representing beta prior distributions*

Description

Create an object representing beta prior distributions

Usage

```
pr_beta(a = 1, b = 1)
```

Arguments

`a` positive shape parameter.
`b` positive shape parameter.

Value

An environment representing the specified prior, for internal use.

`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	<i>Create an object representing Generalised Inverse Gaussian (GIG) prior distributions</i>
--------	---

Description

Create an object representing Generalised 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	<i>Create an object representing inverse chi-squared priors with possibly modelled degrees of freedom and scale parameters</i>
-------------	--

Description

Create an object representing inverse chi-squared priors with possibly modelled 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 modelled 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	<i>Create an object representing an inverse Wishart prior, possibly with modelled scale matrix</i>
---------------	--

Description

Create an object representing an inverse Wishart prior, possibly with modelled 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 modelled 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	<i>Create an object representing a Multivariate Log inverse Gamma (MLiG) prior distribution</i>
---------	---

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	<i>Create an object representing a possibly multivariate normal prior distribution</i>
-----------	--

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.

pr_truncnormal	<i>Create an object representing truncated normal prior distributions</i>
----------------	---

Description

Create an object representing truncated normal prior distributions

Usage

```
pr_truncnormal(mean = 0, precision = 1, lower = 0, upper = Inf)
```

Arguments

mean	scalar or vector mean parameter.
precision	scalar, vector or matrix precision parameter.
lower	lower limit of the truncated interval.
upper	lower limit of the truncated interval.

Value

An environment representing the specified prior, for internal use.

pr_unif	<i>Create an object representing uniform prior distributions</i>
---------	--

Description

Create an object representing uniform prior distributions

Usage

```
pr_unif(min = 0, max = 1)
```

Arguments

min	lower limit.
max	upper limit.

Value

An environment representing the specified prior, for internal use.

read_draws	<i>Read MCMC draws from a file</i>
------------	------------------------------------

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(mcre)

## End(Not run)
```

reg

Specify 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 σ^2 .

Usage

```
reg(
  formula = ~1,
  remove.redundant = FALSE,
  sparse = NULL,
  X = NULL,
  prior = NULL,
  scale.prior = NULL,
  constraints = NULL,
  Q0 = NULL,
  b0 = NULL,
  name = "",
  debug = FALSE
)
```


Arguments

<code>formula</code>	a formula specifying the predictors to be used in the model, in the same way as the right hand side of the <code>formula</code> argument of R's <code>lm</code> function. Variable names are looked up in the data frame passed as <code>data</code> argument to <code>create_sampler</code> or <code>generate_data</code> , or in <code>environment(formula)</code> .
<code>remove.redundant</code>	whether redundant columns should be removed from the design matrix. Default is <code>FALSE</code> . But note that treatment contrasts are automatically applied to all factor variables in <code>formula</code> .
<code>sparse</code>	whether the model matrix associated with <code>formula</code> should be sparse. The default is to base this on a simple heuristic.
<code>X</code>	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on <code>formula</code> . If <code>X</code> is specified <code>formula</code> is ignored.
<code>prior</code>	prior specification for the regression coefficients. Supported priors can be specified using functions <code>pr_normal</code> , <code>pr_fixed</code> , or <code>pr_MLiG</code> . The latter prior is only available in conjunction with a gamma family sampling distribution.
<code>scale.prior</code>	whether the coefficients' prior scale includes the family's global (residual) scale as a factor. The default, <code>NULL</code> , sets this option to <code>TRUE</code> in case of an informative normal prior and "gaussian", "student-t" or "gaussian-gamma" family with non-fixed scale parameter, and to <code>FALSE</code> otherwise.
<code>constraints</code>	optional linear equality and/or inequality constraints imposed on the vector of regression coefficients. Use function <code>set_constraints</code> to specify the constraint matrices and right-hand sides.
<code>Q0</code>	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 <code>prior</code> instead, i.e. <code>prior = pr_normal(mean = b0.value, precision = Q0.value)</code> .
<code>b0</code>	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 <code>prior</code> instead, i.e. <code>prior = pr_normal(mean = b0.value, precision = Q0.value)</code> .
<code>name</code>	the name of the model component. This name is used in the output of the MCMC simulation function <code>MCMCsim</code> . By default the name will be 'reg' with the number of the model term attached.
<code>debug</code>	if <code>TRUE</code> 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
C <- set_constraints(R=R, r=r)
sampler <- create_sampler(y ~ reg(~ 1 + x, constraints=C, 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(n_chains(object)),
  draws = seq_len(n_draws(object)),
  matrix = FALSE,
  type = c("link", "response"),
  ...
)

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

Arguments

<code>object</code>	an object of class <code>mcdraws</code> .
<code>mean.only</code>	if <code>TRUE</code> only the vector of posterior means is returned. In that case the subsequent arguments are ignored. Default is <code>FALSE</code> .
<code>units</code>	the data units (by default all) for which fitted values or residuals should be computed.
<code>chains</code>	optionally, a selection of chains.
<code>draws</code>	optionally, a selection of draws per chain.
<code>matrix</code>	whether a matrix should be returned instead of a <code>dc</code> object.
<code>type</code>	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.
<code>...</code>	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 <- mcmc_sae_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)))
```

s

Specify a smooth term component of the linear predictor

Description

This function can be used inside the formula specification of the linear predictor in [create_sampler](#) or [generate_data](#). The smooth term is set up by the smooth term specification function [s](#) of package **mgcv**. The smooth term is usually composed of random (penalised) effects as well as a few fixed (unpenalised) effects, not including an intercept.

Usage

```
s(..., unit.precision = FALSE, name = "", debug = FALSE)
```

Arguments

...	parameters passed to <code>mgcv::s</code> . Note that variables appearing in ... must be present in the data frame passed to the data argument of create_sampler or generate_data .
unit.precision	to be implemented.
name	the name of the model component. By default the name will be 's' with the number of the model term attached. This name is used in the output of the MCMC simulation function <code>MCMCsim</code> . The name is appended by '_f' for any unpenalised (fixed) effects, if any, and by '_r' for the penalised (random) effects.
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

S.N. Wood (2017). Generalized additive models: an introduction with R. Chapman and Hall/CRC.

Examples

```
## Not run:
library(mgcv)
set.seed(0)
dat <- gamSim(5, n=200, scale=2)
b <- gam(y ~ x0 + s(x1) + s(x2) + s(x3), data=dat)

sampler <- create_sampler(
  y ~ x0 + s(x1) + s(x2) + s(x3), data=dat
)
sim <- MCMCsim(sampler, store.all=TRUE)
(summ <- summary(sim))
plot(
  coef(b),
  c(summ$reg1[, "Mean"],
    summ$s2_r[, "Mean"], summ$s2_f[, "Mean"],
    summ$s3_r[, "Mean"], summ$s3_f[, "Mean"],
    summ$s4_r[, "Mean"], summ$s4_f[, "Mean"]
  )
); abline(0, 1)
predb <- predict(b, newdata=dat[1:5, ])
pred <- predict(sim, newdata=dat[1:5, ], type="response")
(summpred <- summary(pred))
plot(predb, summpred[, "Mean"]); abline(0, 1)

## End(Not run)
```

sampler_control

Set computational options for the sampling algorithms

Description

This function can be used to specify computational options by passing the result to the control argument of [create_sampler](#).

Usage

```
sampler_control(
  add.outer.R = TRUE,
  add.eps.I = FALSE,
  eps = sqrt(.Machine$double.eps),
  recompute.e = TRUE,
  compute.llh = TRUE,
  cMVN.sampler = NULL,
  CG = NULL,
  block = TRUE,
  auto.order.block = TRUE,
  chol.control = chol_control(),
```

```

    max.size.cps.template = 100
)

```

Arguments

<code>add.outer.R</code>	whether to add the outer product of a constraint matrix to the conditional posterior precision matrix of coefficients sampled in a block. This is used to resolve singularity due to intrinsic GMRF components. Default is TRUE. When set to NULL, a simple heuristic is used to decide whether to add the outer product of (possibly a submatrix of) the constraint matrix.
<code>add.eps.I</code>	whether to add a small positive multiple of the identity matrix to the conditional posterior precision matrix of coefficients sampled in a block. If needed, this can resolve singularity as an alternative to <code>add.outer.R=TRUE</code> . The advantage of <code>add.eps.I=TRUE</code> is that a sparse conditional posterior precision matrix remains sparse so that sampling is faster, at the cost of slightly deviating from the target posterior distribution, depending on the value of <code>eps</code> . If <code>add.eps.I=TRUE</code> <code>add.outer.R</code> will be set to FALSE.
<code>eps</code>	a positive scalar value, used only in case <code>add.eps.I=TRUE</code> . This should be a small value to ensure that one is not deviating too much from the desired posterior distribution of coefficients sampled in a block. On the other hand, if it is chosen too small it may not resolve the singularity of the conditional posterior precision matrix of coefficients sampled in a block.
<code>recompute.e</code>	when FALSE, residuals or linear predictors are only computed at the start of the simulation. This may give a modest speed-up but in some cases may be less accurate due to round-off error accumulation. Default is TRUE.
<code>compute.llh</code>	whether to compute the log-likelihood for each MCMC draw. Default is TRUE, but setting it to FALSE may reduce computation time a little.
<code>CMVN.sampler</code>	whether an extended 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, especially in case of a large number of equality constraints, typically (intrinsic) GMRF identifiability constraints.
<code>CG</code>	use a conjugate gradient iterative algorithm instead of Cholesky updates for sampling the model's coefficients. This must be a list with possible components <code>max.it</code> , <code>stop.criterion</code> , <code>verbose</code> , <code>preconditioner</code> and <code>scale</code> . 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.
<code>block</code>	if TRUE, the default, all coefficients are sampled in a single Gibbs block. If FALSE, the coefficients of each model component are sampled separately in sequence. Alternatively, a list of character vectors with names of model components can be passed to specify a grouping of model components whose coefficients should be sampled together in blocks.
<code>auto.order.block</code>	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.

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

Arguments

<code>...</code>	passed to <code>create_sampler</code> (can be all parameters except <code>prior.only</code>)
<code>pars</code>	named list with univariate functions of the parameters to use in test. This list is passed to argument <code>pred</code> of <code>MCMCsim</code> .
<code>n.draw</code>	number of posterior draws to retain in posterior simulations.
<code>n.sim</code>	number of simulation iterations.
<code>burnin</code>	burnin to use in posterior simulations, passed to <code>MCMCsim</code> .
<code>thin</code>	thinning to use in posterior simulations, passed to <code>MCMCsim</code> .
<code>show.progress</code>	whether a progress bar should be shown.
<code>verbose</code>	set to <code>FALSE</code> to suppress messages.
<code>n.cores</code>	the number of cpu cores to use. Default is one, i.e. no parallel computation. If an existing cluster <code>cl</code> is provided, <code>n.cores</code> will be set to the number of workers in that cluster.
<code>cl</code>	an existing cluster can be passed for parallel computation. If <code>NULL</code> and <code>n.cores > 1</code> , a new cluster is created.
<code>seed</code>	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
<code>export</code>	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"),
  family=f_gaussian(var.prior=pr_invchisq(df=1, scale=list(df=1, scale=1))),
  data=dat,
  pars=list(mu="beta[1]", beta_x="beta[2]", sigma="sigma_"),
  n.draw=9L, n.sim=10L*20L, thin=2L, burnin=20L
)
ranks

## End(Not run)
```

setup_cluster	<i>Set up a cluster for parallel computing</i>
---------------	--

Description

The cluster is set up for a number of workers by loading the **mcmc**sae 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.

set_constraints	<i>Set up a system of linear equality and/or inequality constraints</i>
-----------------	---

Description

Two-sided inequalities specified by S2, l2, u2 are currently transformed into the one-sided form $S'x \geq s$, combined with any directly specified constraints of this form. Some basic consistency checks are carried out, notably regarding the dimensions of the inputs.

Usage

```
set_constraints(
  R = NULL,
  r = NULL,
  S = NULL,
  s = NULL,
  S2 = NULL,
  l2 = NULL,
  u2 = NULL,
  scale = FALSE
)
```

Arguments

R	equality constraint matrix each column of which corresponds to a constraint.
r	right-hand side vector for equality constraints $R'x = r$, where R' denotes the transpose of R.
S	inequality constraint matrix each column of which corresponds to an inequality constraint.
s	rhs vector for inequality constraints $S'x \geq s$, where S' denotes the transpose of S.
S2	inequality constraint matrix each column of which corresponds to a two-sided inequality constraint.
l2	vector of lower bounds for two-sided inequality constraints $l_2 \leq S'_2x \leq u_2$.
u2	vector of upper bounds for two-sided inequality constraints $l_2 \leq S'_2x \leq u_2$.
scale	whether to scale the columns of all constraint matrices to unit Euclidean norm.

Value

An environment with constraint matrices and vectors and a method to check whether a numeric vector satisfies all constraints. Returns NULL in case of no constraints.

set_MH	<i>Set options for Metropolis-Hastings sampling</i>
--------	---

Description

Set options for Metropolis-Hastings sampling

Usage

```
set_MH(type = "RWTN", scale = 0.025, adaptive = NULL, ...)
```

Arguments

type	a character string defining the proposal distribution. Among supported types are random walk proposals "RWTN", "RWN" and "RWLN" with truncated normal, normal and log-normal proposal distributions. Other choices correspond to independence proposals: "TN" for a truncated normal proposal, "unif" for a uniform proposal, and "beta" and "gamma" for specific beta and gamma proposal distributions. Not all types are supported for a particular parameter; see the specific help of the function defining the model component of interest to see which proposal distribution types are supported.
scale	in case of the "RWTN" proposal, the (initial) scale of the distribution.
adaptive	in case of the random walk "RWTN" or "RWN" proposals, whether the scale parameter is adapted based on acceptance rates during the burnin phase of the MCMC simulation. The default is TRUE in these cases.

... additional parameters depending on the proposal type. Supported arguments are 'l' and 'u' to pass the lower and upper limits of uniform or random walk truncated normal proposals (defaults l=0 and u=1), and 'a' and 'b' to pass the shape parameters of a beta proposal distribution (defaults a = b = 0.5).

Value

An environment with variables and methods for Metropolis-Hastings sampling, for use by other package functions.

sim_marg_var	<i>Compute a Monte Carlo estimate of the marginal variances of a (I)GMRF</i>
--------------	--

Description

Estimate marginal variances of a (I)GMRF prior defined in terms of a sparse precision matrix and possibly a set of equality constraints. The marginal variances might be used to rescale the precision matrix such that a default prior for a corresponding variance component is more appropriate.

Usage

```
sim_marg_var(
  D,
  Q = NULL,
  R = NULL,
  r = NULL,
  eps1 = 1e-09,
  eps2 = 1e-09,
  nSim = 100L
)
```

Arguments

D	factor of precision matrix Q such that $Q=D'D$.
Q	precision matrix.
R	equality restriction matrix.
r	rhs vector for equality constraints $R'x = r$, where R' denotes the transpose of R.
eps1	passed to create_cMVN_sampler .
eps2	passed to create_cMVN_sampler .
nSim	number of Monte Carlo samples used to estimate the marginal variances.

Value

A vector of Monte Carlo estimates of the marginal variances.

References

S.H. Sorbye and H. Rue (2014). Scaling intrinsic Gaussian Markov random field priors in spatial modelling. *Spatial Statistics*, 8, 39-51.

stop_cluster	<i>Stop a cluster</i>
--------------	-----------------------

Description

Stop a cluster set up by [setup_cluster](#).

Usage

```
stop_cluster(cl)
```

Arguments

cl the cluster object.

Value

NULL.

student_t_control	<i>Set (MCMC) computational options for a Student-t likelihood model</i>
-------------------	--

Description

Set (MCMC) computational options for a Student-t likelihood model

Usage

```
student_t_control(proposal = "RW", MHscale = 1, adapt = TRUE)
```

Arguments

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

Value

A list with computational options.

subset.dc	<i>Select a subset of chains, samples and parameters from a draws component (dc) object</i>
-----------	---

Description

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

Usage

```
## S3 method for class 'dc'
subset(x, chains = NULL, draws = NULL, vars = NULL, ...)
```

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$beta, chains=1, draws=sample(1:n_draws(sim), 100))))
(summary(subset(sim$beta, vars=1:2)))
```

summary.dc

*Summarise a draws component (dc) object***Description**

Summarise 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 labelled '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 <- mcmc_sae_example()
sampler <- create_sampler(ex$model, data=ex$dat)
sim <- MCMCsim(sampler, store.all=TRUE, n.chain=2, n.iter=500)
summary(sim$u)
```

summary.mcdraws	<i>Summarise an mcdraws object</i>
-----------------	------------------------------------

Description

Summarise 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, n.chain=2, n.iter=500)
```

```
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(use.cholV = NULL)

m_Gibbs(slice = FALSE, eps = sqrt(.Machine$double.eps), diagnostic = FALSE)

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

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

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

Arguments

<code>use.cholV</code>	whether to use the Cholesky factor of the variance instead of precision matrix for sampling. If NULL the choice is made based on a simple heuristic.
<code>slice</code>	if TRUE, a Gibbs within slice sampler is used.
<code>eps</code>	small positive value to control numerical robustness of the algorithm.

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.
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.
scale	vector of Laplace scale parameters for method 'HMCZigZag'. It must be a positive numeric vector of length equal to one or the number of variables.
prec.eq	positive numeric vector of length 1 or the number of equality restrictions, to control the precision by 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 be 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	<i>Transform one or more draws component objects into a new one by applying a function</i>
--------------	--

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 <- mcmc_sae_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	<i>Create a model component object for a variance factor component in the variance function of a gaussian sampling distribution</i>
------	---

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 <code>pr_invchisq</code> or <code>pr_exp</code> , respectively. Alternatively, the variance factors can be set to fixed values using <code>pr_fixed</code> . 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 <code>MCMCsim</code> . 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	<i>Create a model component object for a regression component in the variance function of a gaussian sampling distribution</i>
------	--

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. A normal prior can be specified using function pr_normal . Alternatively, fixed values for the coefficients can be specified using pr_fixed , e.g. to generate data with given coefficients.
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	<i>Extract weights from an mcdraws object</i>
-----------------	---

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

`%m*v%` (matrix-vector), 47

acceptance_rates, 4

aggrMatrix, 5

AR1 (correlation), 14

as.array.dc (MCMC-object-conversion), 50

as.data.frame.dc_summary, 6

as.data.frame.mcdraws_summary, 6

as.matrix.dc (MCMC-object-conversion), 50

bayesR2, 7

binomial_control, 7, 25

brt, 8, 20

CG_control, 10, 86, 97

chol_control, 10, 11, 12, 17, 23, 87

Cholesky, 11

cMVN_control, 11

combine_chains, 12

combine_iters, 12

compute_DIC
(model-information-criteria), 58

compute_WAIC
(model-information-criteria), 58

computeDesignMatrix, 13

correlation, 14, 39

create_cMVN_sampler, 17, 91

create_sampler, 8, 9, 18, 24–27, 30–32, 35, 36, 38, 39, 41, 42, 44, 48, 52, 53, 55–57, 80–82, 84, 85, 88, 98–100

create_TMVN_sampler, 21, 96

crossprod_mv (matrix-vector), 47

custom (correlation), 14

dbarts, 9

density, 64, 65

draws_array, 50

f_binomial, 19, 20, 24

f_gamma, 19, 26, 30

f_gaussian, 19, 20, 27, 30, 36

f_gaussian_gamma, 19, 30

f_multi, 31

f_multinomial, 19, 20, 31

f_negbinomial, 19, 20, 32

f_poisson, 19, 35

f_student_t, 36

fitted.mcdraws
(residuals-fitted-values), 82

gaussian_control, 28, 37

gen, 14, 18, 20, 28, 30, 38, 44, 45

gen_control, 40, 42

generate_data, 8, 9, 20, 24, 26, 27, 30–32, 35, 36, 38, 39, 41, 44, 55–57, 80, 81, 84, 98–100

get_draw, 43

get_means (posterior-moments), 68

get_sds (posterior-moments), 68

glm, 25

glreg, 40, 44

GMRF_structure, 40, 45

iid (correlation), 14

labels, 46

labels<- (labels), 46

loo, 59

loo.mcdraws
(model-information-criteria), 58

m_direct (TMVN-methods), 96

m_Gibbs (TMVN-methods), 96

m_HMC (TMVN-methods), 96

m_HMCZigZag (TMVN-methods), 96

m_softTMVN (TMVN-methods), 96

matrix-vector, 47

- maximize_log_lh_p, [47](#)
- mc_offset, [55](#)
- MCMC-diagnostics, [49](#)
- MCMC-object-conversion, [50](#)
- mcmc.list, [50](#)
- mcmc_sae (mcmc_sae-package), [4](#)
- mcmc_sae-package, [4](#)
- mcmc_sae_example, [51](#)
- MCMCsim, [4](#), [7](#), [9](#), [18](#), [20](#), [40](#), [44](#), [52](#), [56–58](#), [69](#), [72](#), [79](#), [81](#), [82](#), [84](#), [88](#), [95](#), [99](#), [100](#)
- mec, [18](#), [20](#), [56](#)
- model-information-criteria, [58](#)
- model_matrix, [60](#)
- multinomial_control, [32](#), [61](#)
- n_chains (n_chains-n_draws-n_vars), [62](#)
- n_chains-n_draws-n_vars, [62](#)
- n_draws (n_chains-n_draws-n_vars), [62](#)
- n_eff, [94](#), [95](#)
- n_eff (MCMC-diagnostics), [49](#)
- n_vars (n_chains-n_draws-n_vars), [62](#)
- negbinomial_control, [33](#), [62](#)
- offset, [20](#)
- optim, [48](#)
- par_names, [63](#)
- plot.dc, [64](#)
- plot.mcdraws, [65](#)
- plot_coef, [65](#)
- poisson_control, [35](#), [67](#)
- poly2nb, [14](#), [15](#)
- posterior-moments, [68](#)
- pr_beta, [73](#)
- pr_exp, [19](#), [26](#), [28](#), [40](#), [73](#), [99](#)
- pr_fixed, [19](#), [26](#), [28](#), [33](#), [45](#), [74](#), [81](#), [99](#), [100](#)
- pr_gamma, [26](#), [36](#), [74](#)
- pr_gig, [19](#), [28](#), [33](#), [75](#)
- pr_invchisq, [19](#), [28](#), [33](#), [39](#), [40](#), [75](#), [99](#)
- pr_invwishart, [39](#), [76](#)
- pr_MLiG, [28](#), [30](#), [39](#), [40](#), [77](#), [81](#)
- pr_normal, [39](#), [44](#), [56](#), [57](#), [78](#), [80](#), [81](#), [100](#)
- pr_truncnormal, [78](#)
- pr_unif, [45](#), [79](#)
- predict.mcdraws, [69](#)
- print.dc_summary, [71](#)
- print.mcdraws_summary, [72](#)
- R_hat (MCMC-diagnostics), [49](#)
- read_draws, [79](#)
- reg, [18](#), [20](#), [28](#), [30](#), [80](#)
- residuals-fitted-values, [82](#)
- residuals.mcdraws
(residuals-fitted-values), [82](#)
- RW1 (correlation), [14](#)
- RW2 (correlation), [14](#)
- s, [84](#), [84](#)
- sampler_control, [20](#), [85](#), [97](#)
- SBC_test, [87](#)
- season (correlation), [14](#)
- set_constraints, [22](#), [40](#), [57](#), [81](#), [89](#)
- set_MH, [15](#), [26](#), [45](#), [90](#)
- setup_cluster, [89](#), [92](#)
- sim_marg_var, [91](#)
- spatial (correlation), [14](#)
- splineDesign, [14–16](#)
- splines (correlation), [14](#)
- st_read, [14](#)
- stop_cluster, [92](#)
- student_t_control, [92](#)
- subset.dc, [93](#)
- summary.dc, [94](#)
- summary.mcdraws, [72](#), [95](#)
- TMVN-methods, [96](#)
- to_draws_array
(MCMC-object-conversion), [50](#)
- to_mcmc (MCMC-object-conversion), [50](#)
- transform_dc, [97](#)
- vfac, [19](#), [28](#), [30](#), [98](#)
- vreg, [19](#), [28](#), [30](#), [99](#)
- waic, [59](#)
- waic.mcdraws
(model-information-criteria),
[58](#)
- weights.mcdraws, [100](#)