Package 'mcmcsae'

October 13, 2022

Version 0.7.1

License GPL-3

Title Markov Chain Monte Carlo Small Area Estimation

Type Package

LazyLoad yes

Encoding UTF-8

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

Date 2022-09-02

Depends R (>= 3.2.0)

Imports Matrix (>= 1.2.0), Rcpp (>= 0.11.0), methods, GIGrvg, loo (>=
2.0.0), matrixStats

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

LinkingTo Rcpp, RcppEigen, Matrix, GIGrvg

VignetteBuilder knitr

RoxygenNote 7.2.1

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

NeedsCompilation yes

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

Maintainer Harm Jan Boonstra <hjboonstra@gmail.com>

Repository CRAN

Date/Publication 2022-09-02 13:30:02 UTC

R **topics documented:**

---

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

Description

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

Details

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

---

acceptance_rates   *Return Metropolis-Hastings acceptance rates*

Description

Return Metropolis-Hastings acceptance rates

Usage

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

Arguments

obj   an mcdraws object, i.e. the output of function `MCMCsim`.

aggregate.chains

               whether to return averages over chains or results per chain.

Value

A list of acceptance rates.

Examples

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

---

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

Description

Utility function to construct a sparse aggregation matrix from a factor

Usage

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

Arguments

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

Value

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)))
```

---

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

Description

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

Usage

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

Arguments

...   objects of class mcdraws.

Value

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

---

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

Description

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

Usage

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

Arguments

...   objects of class mcdraws

Value

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

---

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

Description

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

Usage

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

Arguments

formula	model formula.
data	data frame to be used in deriving the design matrices.
sampler	a sampler environment (as created by e.g. by create_sampler).
labels	if TRUE, column names are assigned.

Value

A list of design matrices.

Examples

```
n <- 1000
dat <- data.frame(
  x = rnorm(n),
  f = factor(sample(1:50, n, replace=TRUE))
)
str(computeDesignMatrix(~ x, dat)[[1]])
model <- ~ reg(~x, name="beta") + gen(~x, factor=~f, name="v")
X <- computeDesignMatrix(model, dat)
str(X)
sampler <- create_sampler(model, dat, prior.only=TRUE)
str(computeDesignMatrix(sampler=sampler))
str(computeDesignMatrix(sampler=sampler, labels=FALSE))
newdata <- data.frame(x=rnorm(10), f=dat$f[1:10])
str(computeDesignMatrix(sampler=sampler, data=newdata))
```

___

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

___

Description

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

Usage

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

Arguments

factor	factor formula of a generic model component, see [gen](#).
data	data frame to be used in deriving the matrices.
D	if TRUE compute the incidence matrix.
Q	if TRUE compute the precision matrix.
R	if TRUE compute the restriction matrix.
cols2remove	if an integer vector is passed, the dimensions (columns of D, rows and columns of Q and rows of R) that are removed. This can be useful in the case of empty domains.
remove.redundant.R.cols	
	whether to test for and remove redundant restrictions from restriction matrix R
enclos	enclosure to look for objects not found in `data`.
n.parent	for internal use; in case of custom factor, the number of frames up the calling stack in which to evaluate any custom matrices
...	further arguments passed to `economizeMatrix`.

Value

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

Examples

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

---

correlation   *Correlation structures*

Description

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

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

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

RW2(f) Second-order random walk.

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

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

spatial(f, poly.df, snap, queen, derive.constraints=FALSE) CAR spatial correlation. Argument poly.df should be an object of class SpatialPolygonsDataFrame obtained e.g. from reading in a shape file with [readShapeSpatial](#) from package **maptools**. Arguments snap and queen are passed to [poly2nb](#). If derive.constraints=TRUE the constraint matrix for an IGMRF model component is formed by computing the singular vectors of the precision matrix.

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.

References

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

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

---

create_sampler *Create a sampler object*

Description

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

Usage

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

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 (if any) can be found.

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

modeled factors at a certain level specified by a factor variable. By using unit-level inverse-chi-squared factors the marginal sampling distribution becomes a Student-t distribution, and by using unit-level exponential factors it becomes a Laplace or double exponential distribution.

logJacobian
: if the data are transformed the logarithm of the Jacobian can be supplied so that it is incorporated in all log-likelihood computations. This can be useful for comparing information criteria for different transformations. It should be supplied as a vector of the same size as the response variable, and is currently only supported if `family="gaussian"`. For example, when a log-transformation is used on response vector y, the vector `-log(y)` should be supplied.

linpred
: a list of matrices defining (possibly out-of-sample) linear predictors to be simulated. This allows inference on e.g. (sub)population totals or means. The list must be of the form `list(name_1=X_1, ...)` where the names refer to the model component names and predictions are computed by summing `X_i %*% p[[name_i]]`. Alternatively, `linpred="fitted"` can be used for simulations of the full in-sample linear predictor.

compute.weights
: if `TRUE` weights are computed for each element of `linpred`. Note that for a large dataset in combination with vector-valued linear predictors the weights can take up a lot of memory. By default only means are stored in the simulation carried out using [`MCMCsim`](MCMCsim).

block
: if `TRUE` all coefficients are sampled in a single block. Alternatively, a list of character vectors indicating which coefficients should be sampled together in blocks.

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`](generate_data), which samples from the prior predictive distribution.

control
: a list with further computational options, see details section.

Details

The right hand side of the `formula` argument to `create_sampler` can be used to specify additive model components. Currently two specialized model components are supported, [`reg`](reg)`(...)` and [`gen`](gen)`(...)` for regression and generic random effects components, respectively.

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

Further computational options can be set using the `control` parameter, which should be passed as a list with possible elements

add.outer.R whether to add the outer product of the constraint matrix for a better conditioned solve system for blocks. This is done by default when using blocked Gibbs sampling for blocks with constraints.

recompute.e when `FALSE` residuals or linear predictors are only computed at the start of the simulation. This may give a modest speedup but in some cases may be less accurate due to roundoff error accumulation. Default is `TRUE`.

CG for a conjugate gradient iterative algorithm instead of Cholesky updates for sampling the
model's coefficients. This must be a list with possible components `max.it`, `stop.criterion`,
`verbose`, `preconditioner` and `scale`, see `setup_CG_sampler`. This is currently an experi-
mental feature.

Value

A sampler object, which is the main input for the MCMC simulation function `MCMCsim`. The sam-
pler 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.

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

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

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

Examples

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

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

---

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

Description

This function sets up an object for multivariate normal sampling based on a specified precision matrix or cholesky decomposition thereof. Linear equality and inequality restrictions are supported. For sampling under inequality restrictions three algorithms are available. The default in that case is an exact Hamiltonian Monte Carlo algorithm (Pakman and Paninski, 2014). Alternatively, a Gibbs sampling algorithm can be used (Rodriguez-Yam et al., 2004). The third option is a data augmentation method to sample from a smooth approximation to the truncated multivariate normal distribution (Souris et al., 2018).

Usage

```
create_TMVN_sampler(
  Q,
  perm = NULL,
  mu = NULL,
  Xy = NULL,
  update.Q = FALSE,
  update.mu = update.Q,
  name = "x",
  coef.names = NULL,
  R = NULL,
  r = NULL,
  S = NULL,
  s = NULL,
  lower = NULL,
  upper = NULL,
  method = NULL,
  reduce = (method == "Gibbs" && !is.null(R)),
  T.HMC = pi/2,
  print.info = FALSE,
  sharpness = 100,
  useV = FALSE
)
```

Arguments

Q	precision matrix (unconstrained) (n x n); used in case cholQ is not supplied.
perm	whether permutation/pivoting should be used to build a Cholesky object.
mu	mean of the (unconstrained) multivariate normal distribution.
Xy	alternative to specifying mu; in this case mu is computed as Q^{-1} Xy.
update.Q	whether Q is updated for each draw.

update.mu	whether mu is updated for each draw. By default equal to update.Q.
name	name of the TMVN vector parameter.
coef.names	optional labels for the components of the vector parameter.
R	equality restriction matrix.
r	rhs vector for equality constraints R^T x = r.
S	inequality restriction matrix.
s	rhs vector for inequality constraints S^T x >= s.
lower	alternative to s for two-sided inequality restrictions lower <= S^T x <= upper.
upper	alternative to s for two-sided inequality restrictions lower <= S^T x <= upper.
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, "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.
reduce	whether to a priori restrict the simulation to the subspace defined by the equality constraints.
T.HMC	the duration of a Hamiltonian Monte Carlo simulated particle trajectory. If a vector of length 2 is supplied it is interpreted as an interval from which the duration is drawn uniformly, independently in each HMC iteration.
print.info	whether information about violations of inequalities and bounces off inequality walls is printed to the screen. This sometimes provides useful diagnostic information.
sharpness	for method 'softTMVN', the sharpness of the soft inequalities; the larger the better the approximation of exact inequalities. It must be either a scalar value or a vector of length equal to the number of inequality restrictions.
useV	for method 'softTMVN' whether to base computations on variance instead of precision matrices.

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.

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

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

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

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

Examples

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

---

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

Description

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

Usage

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

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`](create_sampler) or [`generate_data`](generate_data), or in `environment(formula)`.

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

[create_sampler](#) or [generate_data](#), or in environment(formula).

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

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

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

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

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

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

PX
: whether parameter expansion should be used. Default is TRUE, which applies parameter expansion with default options. Alternative options can be specified by supplying a list with one or more of the following components:

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

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

  mu0 location (vector) parameter for parameter expansion. By default 0.

  Q0 precision (matrix) parameter for parameter expansion. Default is the identity matrix.

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

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

Leroux
: this option alters the precision matrix determined by factor by taking a weighted average of it with the identity matrix. If TRUE the model gains an additional parameter, the 'Leroux' parameter, being the weight of the original, structured, precision matrix in the weighted average. By default a uniform prior for the weight and a uniform Metropolis-Hastings proposal density are employed. This

	default can be changed by supplying a list with elements a, b, and a.star, b.star, implying a beta(a, b) prior and a beta(a.star, b.star) independence proposal density. A third option is to supply a single number between 0 and 1, which is then used as a fixed value for the Leroux parameter.
R0	an optional equality restriction matrix acting on the coefficients defined by `formula`, for each level defined by `factor`. If c is the number of restrictions, R0 is a q0 x c matrix where q0 is the number of columns of the design matrix derived from `formula`. Together with RA it defines the set of equality constraints to be imposed on the vector of coefficients. Only allowed in combination with `var="scalar"`.
RA	an optional equality restriction matrix acting on the coefficients defined by `factor`, for each effect defined by `formula`. If c is the number of restrictions, RA is a l x c matrix where l is the number of levels defined by `factor`. Together with R0 this defines the set of equality constraints to be imposed on the vector of coefficients. If `constr=TRUE`, additional constraints are imposed, corresponding to the null-vectors of the singular precision matrix in case of an intrinsic Gaussian Markov Random Field.
constr	whether constraints corresponding to the null-vectors of the precision matrix are to be imposed on the vector of coefficients. By default this is `TRUE` for improper or intrinsic GMRF model components, i.e. components with a singular precision matrix such as random walks or CAR spatial components.
S0	an optional inequality restriction matrix acting on the coefficients defined by `formula`, for each level defined by `factor`. If c is the number of restrictions, S0 is a q0 x c matrix where q0 is the number of columns of the design matrix derived from `formula`. Together with SA it defines the set of inequality constraints to be imposed on the vector of coefficients.
SA	an optional inequality restriction matrix acting on the coefficients defined by `factor`, for each effect defined by `formula`. If c is the number of restrictions, SA is a l x c matrix where l is the number of levels defined by `factor`. Together with S0 this defines the set of constraints to be imposed on the vector of coefficients.
formula.gl	a formula of the form `~ glreg(...)` for group-level predictors around which the random effect component is hierarchically centered. See [glreg](#) for details.
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.
perm	whether permutation should be used in the Cholesky decomposition used for updating the model component's coefficient vector. Default is based on a simple heuristic.
debug	if `TRUE` a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.
e	for internal use only.

Value

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

References

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

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

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

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

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

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

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

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

---

generate_data   *Generate a data vector according to a model*

Description

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

Usage

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

Value

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

Examples

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

---

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

Description

Extract a list of parameter values for a single draw

Usage

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

Arguments

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

Value

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

Examples

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

---

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

Description

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

Usage

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

Arguments

formula   a formula specifying the group-level predictors to be used within a model com-
                     ponent. If no data is supplied the group-level predictors are derived as group-
                     level means from the unit-level data passed as data argument to `create_sampler`
                     or `generate_data`.

remove.redundant
                     whether redundant columns should be removed from the design matrix. Default
                     is FALSE.

Q0   prior precision matrix for the group-level effects. The default is a zero matrix
                     corresponding to a noninformative improper prior.

data   group-level data frame in which the group-level variables specified in formula
                     are looked up.

name   the name of the model component. This name is used in the output of the MCMC
                     simulation function `MCMCsim`. By default this name will be the name of the
                     corresponding generic random effects component appended by '_gl'.

e   for internal use only.

Value

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

---

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

Description

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

Usage

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

labels(object) <- value
```

Arguments

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

Value

The extractor function returns the variable labels.

Examples

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

---

matrix-vector	*Fast matrix-vector multiplications*

Description

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

Usage

```
M %m*v% v

crossprod_mv(M, v)
```

Arguments

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

Value

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

Examples

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

---

maximize_llh   *Maximize log-likelihood defined inside a sampler function*

Description

Maximize log-likelihood defined inside a sampler function

Usage

```
maximize_llh(sampler, method = "BFGS", control = list(fnscale = -1), ...)
```

Arguments

sampler	sampler function.
method	optimization method, passed to optim.
control	control parameters, passed to optim.
...	other parameters passed to optim.

Value

A list of parameter values that, provided the optimization was successful, maximize the likelihood.

Examples

```
n <- 1000
dat <- data.frame(
  x = rnorm(n),
  f = factor(sample(1:50, n, replace=TRUE))
)
df <- generate_data(
  ~ reg(~x, name="beta", Q0=1) + gen(~x, factor=~f, name="v"),
  sigma.fixed=TRUE, data=dat
)
dat$y <- df$y
sampler <- create_sampler(y ~ x + gen(~x, factor=~f, name="v"), data=dat)
opt <- maximize_llh(sampler)
```

```
str(opt)
plot(df$par$v, opt$par$v); abline(0, 1, col="red")
```

MCMC-diagnostics   *Compute MCMC diagnostic measures*

Description

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

Usage

```
R_hat(dc)

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

Arguments

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

Value

In case of `R_hat` the split-Rhat 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.

Usage

```
to_mcmc(x)

to_draws_array(x)

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

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

Arguments

x	a component of an mcdraws object corresponding to a scalar or vector model parameter.
...	currently ignored.
colnames	whether column names should be set.

Value

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

Examples

```
data(iris)
sampler <- create_sampler(Sepal.Length ~ reg(~ Petal.Length + Species, name="beta"), data=iris)
sim <- MCMCsim(sampler, burnin=100, n.chain=2, n.iter=400)
summary(sim)
if (require("coda", quietly=TRUE)) {
  mcbeta <- to_mcmc(sim$beta)
  geweke.diag(mcbeta)
}
if (require("posterior", quietly=TRUE)) {
  mcbeta <- to_draws_array(sim$beta)
  mcbeta
}
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, Q0=1, name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
sim <- MCMCsim(sampler, n.chain=2, n.iter=400)
str(sim$beta)
str(as.array(sim$beta))
bayesplot::mcmc_hist(as.array(sim$beta))
bayesplot::mcmc_dens_overlay(as.array(sim$beta))
# fake data simulation check:
bayesplot::mcmc_recover_intervals(as.array(sim$beta), gd$pars$beta)
bayesplot::mcmc_recover_hist(as.array(sim$beta), gd$pars$beta)

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

---

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

Description

These functions are intended for use in the `family` argument of [create_sampler](#). In future versions these functions may gain additional arguments, but currently the corresponding functions gaussian

and `binomial` can be used as well.

Usage

```
f_gaussian(link = "identity")

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

f_negbinomial(link = "logit")

f_poisson(link = "log")

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

Arguments

link	the name of a link function. Currently the only allowed link functions are: `"identity"` for (log-)Gaussian sampling distributions, `"logit"` (default) and `"probit"` for binomial distributions and `"log"` for negative binomial sampling distributions.
K	number of categories for multinomial model; this must be specified for prior predictive sampling

Value

A family object.

---

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

Description

This function is used to generate data for several examples.

Usage

```
mcmcsae_example(n = 100L)
```

Arguments

n	the size of the generated dataset.

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` traceplots or pair plots of multiple parameters are displayed during the simulation.

Usage

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

```
  n.cores = 1L,
  cl = NULL,
  seed = NULL,
  export = NULL
)
```

Arguments

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

stop.on.convergence

> if TRUE stop the simulation if the R-hat diagnostics for all parameters in `trace.convergence` are less than `convergence.bound`.

convergence.bound

> threshold used with `stop.on.convergence`.

plot.trace
:   character vector of parameter names for which to plot draws during the simulation. For one or two parameters traceplots will be shown, and if more parameters are specified the results will be displayed in a pairs plot. For vector parameters a specific component can be selected using brackets, e.g. `"beta[2]"`.

add.to.plot
:   if TRUE traceplot is added to every `n.progress` iterations, otherwise a new plot (with new scales etc) is made after each `n.progress` iterations.

plot.type
:   default is "l" (lines).

n.cores
:   the number of cpu cores to use. Default is one, i.e. no parallel computation. If an existing cluster `cl` is provided, `n.cores` will be set to the number of workers in that cluster.

cl
:   an existing cluster can be passed for parallel computation. If NULL and `n.cores` > 1, a new cluster is created.

seed
:   a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.

export
:   a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in `pred` depend on global variables.

Details

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

start function to generate starting values.

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

rprior function to draw from a prior distribution.

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

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

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

Value

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

Examples

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

```
sim <- MCMCsim(sampler, store=c("x", "y"))
str(sim)
summary(sim)

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

# example using create_sampler; first generate some data
n <- 100
dat <- data.frame(x=runif(n), f=as.factor(sample(1:4, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, Q0=1, name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ x + f, data=dat)
sim <- MCMCsim(sampler, burnin=100, n.iter=400, n.chain=2)
(summary(sim))
gd$pars
```

---

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

Description

This function is intended to be used on the right hand side of the formula argument to create_sampler or generate_data. It creates an additive regression term in the model's linear predictor. By default, the prior for the regression coefficients is improper uniform. If b0 or Q0 are specified the prior becomes normal with mean b0 (default 0) and variance (matrix) sigma_^2 Q0^-1 where sigma_^2 is the overall scale parameter of the model, if any. 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.

Usage

```
mec(
  formula = ~1,
  sparse = NULL,
  X = NULL,
  V = NULL,
  Q0 = NULL,
  b0 = NULL,
  R = NULL,
  r = NULL,
```

```
    S = NULL,
    s = NULL,
    lower = NULL,
    upper = NULL,
    name = "",
    perm = NULL,
    debug = FALSE,
    e = parent.frame()
)
```

Arguments

formula	a formula specifying the predictors subject to measurement error and possibly their variances as well. In the latter case the formula syntax ~ (x1 | V.x1) + (x2 | V.x2) + ... should be used where x1, x2, ... are the names of (quantitative) predictors and V.x1, V.x2, ... are the names of the variables holding the corresponding measurement error variances. If only the predictors are specified the formula has the usual form ~ x1 + x2 + ... In that case variances should be specified using argument V. All variable names are looked up in the data frame passed as data argument to [create_sampler](#) or [generate_data](#), or in environment(formula).
sparse	whether the model matrix associated with formula should be sparse. The default is to base this on a simple heuristic.
X	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on formula. If X is specified formula is ignored.
V	measurement error variance; can contain zeros
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.
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.
R	optional constraint matrix for equality restrictions $R'x = r$ where x is the vector of regression effects.
r	right hand side for the equality constraints.
S	optional constraint matrix for inequality constraints $S'x >= s$ where x is the vector of regression effects.
s	right hand side for the inequality constraints.
lower	as an alternative to s, lower and upper may be specified for two-sided constraints $lower <= S'x <= upper$.
upper	as an alternative to s, lower and upper may be specified for two-sided constraints $lower <= S'x <= upper$.
name	the name of the model component. This name is used in the output of the MCMC simulation function [MCMCsim](#). By default the name will be 'reg' with the number of the model term attached.

perm	whether permutation should be used in the Cholesky decomposition used for updating the model component's coefficient. Default is based on a simple heuristic.
debug	if TRUE a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.
e	for internal use only.

Value

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

References

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

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

Examples

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

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

model-information-criteria

Compute DIC, WAIC and leave-one-out cross-validation model measures

Description

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

Usage

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

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

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

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

Arguments

x	an object of class mcdraws.
use.pV	whether half the posterior variance of the deviance should be used as an alternative estimate of the effective number of model parameters for DIC.
diagnostic	whether vectors of log-pointwise-predictive-densities and pointwise contributions to the WAIC effective number of model parameters should be returned.
batch.size	number of data units to process per batch.
show.progress	whether to show a progress bar.
cl	an existing cluster can be passed for parallel computation. If cl is provided, n.cores will be set to the number of workers in that cluster. If NULL and n.cores > 1, a new cluster is created.
n.cores	the number of cpu cores to use. Default is one, i.e. no parallel computation.

by.unit	if TRUE the computation is carried out unit-by-unit, which is slower but uses much less memory.
...	Other arguments, passed to loo. Not currently used by waic.mcdraws.
r_eff	whether to compute relative effective sample size estimates for the likelihood of each observation. This takes more time, but should result in a better PSIS approximation. See loo.

Value

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

References

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

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

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

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

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

P.-C. Buerkner, J. Gabry and A. Vehtari (2020). Efficient leave-one-out cross-validation for Bayesian non-factorized normal and Student-t models. arXiv:1810.10559.

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)
}
```

Description

Compute possibly sparse model matrix

Usage

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

Arguments

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

Value

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

---

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

Description

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

Usage

```
nchains(obj)

ndraws(obj)

nvars(dc)
```

Arguments

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

dc   a draws component object.

Value

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

Examples

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

---

par_names   *Get the parameter names from an mcdraws object*

Description

Get the parameter names from an mcdraws object

Usage

```
par_names(obj)
```

Arguments

obj   an mcdraws object.

Value

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

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Examples

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

---

plot.mcdraws   *Trace, density and autocorrelation plots*

Description

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

Usage

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

Arguments

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

Examples

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

---

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

Description

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

Usage

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

Arguments

`...`	`dc_summary` objects (output by the `summary` method for simulation objects of class dc), `sae` objects (output by the functions of package **hbsae**), or lists. In case of a list the components used are those with name `est` for point estimates, `se` for standard error based intervals or `lower` and `upper` for custom intervals. Instead of `dc_summary` objects matrix objects are also supported as long as they contain columns named "Mean" and "SD" as do `dc_summary` objects. Named parameters of other types that do not match any other argument names are passed to lower-level plot functions.
`n.se`	number of standard errors below and above the point estimates to use for error bars. By default equal to 1. This only refers to the objects of class `dc_summary` and `sae`.
`est.names`	labels to use in the legend for the components of the `...` argument
`sort.by`	vector by which to sort the coefficients, referring to the first object passed.
`decreasing`	if TRUE, sort in decreasing order (default).
`index`	vector of names or indices of the selected areas to be plotted.
`maxrows`	maximum number of rows in a column.

maxcols	maximum number of columns of estimates on a page.
offset	space used between plots of multiple estimates for the same area.
cex.var	the fontsize of the variable names, default=0.8.
mar	a numerical vector of the form c(bottom, left, top, right) which gives the number of lines of margin to be specified on 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, Q0=1, name="beta") + gen(factor=~f, name="v")
gd <- generate_data(model, data=dat)
dat$y <- gd$y
# fit a base model
model0 <- y ~ reg(~ 1, name="beta") + gen(factor=~f, name="v")
sampler <- create_sampler(model0, data=dat, block=TRUE)
sim <- MCMCsim(sampler, store.all=TRUE)
(summ0 <- summary(sim))
# fit 'true' model
model <- y ~ reg(~ x, name="beta") + gen(factor=~f, name="v")
sampler <- create_sampler(model, data=dat, block=TRUE)
sim <- MCMCsim(sampler, store.all=TRUE)
(summ <- summary(sim))
# compare random effect estimates against true parameter values
plot_coef(summ0$v, summ$v, list(est=gd$pars$v), n.se=2, offset=0.2,
  maxrows=10, est.names=c("base model", "true model", "true"))
```

---

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

Description

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

Usage

```
get_means(obj, vnames = NULL)

get_sds(obj, vnames = NULL)
```

Arguments

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

Value

A list with simulation means or standard deviations.

Examples

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

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

Description

Generate draws from the predictive distribution

Usage

```
## S3 method for class 'mcdraws'
predict(
  object,
  newdata = NULL,
  X. = if (is.null(newdata)) "in-sample" else NULL,
  type = c("data", "link", "response", "data_cat"),
  var = NULL,
  ny = NULL,
  ry = NULL,
  fun. = identity,
  labels = NULL,
  ppcheck = FALSE,
  iters = NULL,
```

```
    to.file = FALSE,
    filename,
    write.single.prec = FALSE,
    show.progress = TRUE,
    n.cores = 1L,
    cl = NULL,
    seed = NULL,
    export = NULL,
    ...
)
```

Arguments

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

iters	iterations in `object` to use for prediction. Default `NULL` means that all draws from `object` are used.
`to.file`	if `TRUE` the predictions are streamed to file.
filename	name of the file to write predictions to in case `to.file=TRUE`.
`write.single.prec`	
	Whether to write to file in single precision. Default is `FALSE`.
show.progress	whether to show a progress bar.
n.cores	the number of cpu cores to use. Default is one, i.e. no parallel computation. If an existing cluster `cl` is provided, `n.cores` will be set to the number of workers in that cluster.
cl	an existing cluster can be passed for parallel computation. If `NULL` and `n.cores` > 1, a new cluster is created.
seed	a random seed (integer). For parallel computation it is used to independently seed RNG streams for all workers.
export	a character vector with names of objects to export to the workers. This may be needed for parallel execution if expressions in `fun.` depend on global variables.
`...`	currently not used.

Value

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

Examples

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

---

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

Description

Display a summary of a dc object

Usage

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

Arguments

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

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Examples

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

---

pr_exp   *Create an object containing information about exponential prior distributions*

Description

Create an object containing information about exponential prior distributions

Usage

```
pr_exp(scale = 1, n = NULL, post = FALSE)
```

Arguments

scale   scale parameter of length 1 or n.

n   dimension, if known. For internal use only.

post   whether conditional posterior sampling function should be created. For internal
               use only.

Value

An environment with information about the prior and possibly conditional posterior distribution(s),
to be used by other package functions.

---

pr_fixed   *Create an object containing information about a degenerate prior fix-*
                            ing a parameter to a fixed value

Description

Create an object containing information about a degenerate prior fixing a parameter to a fixed value

Usage

```
pr_fixed(value = 1, n = NULL, post = FALSE)
```

Arguments

value   value parameter of length 1 or n.

n   dimension, if known. For internal use only.

post   whether conditional posterior sampling function should be created. For internal
               use only.

Value

An environment with information about the prior and possibly conditional posterior distribution(s),
to be used by other package functions.

---

pr_gig	*Create an object containing information about Generalized Inverse Gaussian (GIG) prior distributions*

Description

Create an object containing information about Generalized Inverse Gaussian (GIG) prior distributions

Usage

```
pr_gig(a, b, p, n = NULL, post = FALSE)
```

Arguments

a	parameter of length 1 or n.
b	parameter of length 1 or n.
p	parameter of length 1 or n.
n	dimension, if known. For internal use only.
post	whether conditional posterior sampling function should be created. For internal use only.

Value

An environment with information about the prior and possibly conditional posterior distribution(s), to be used by other package functions.

---

pr_invchisq	*Create an object containing information about inverse chi-squared priors with possibly modeled degrees of freedom and scale parameters*

Description

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

Usage

```
pr_invchisq(df = 1, scale = 1, n = NULL, post = FALSE)
```

Arguments

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

     alpha0 shape parameter of the gamma distribution

     beta0 rate parameter of the gamma distribution

     proposal "RW" for random walk Metropolis-Hastings or "mala" for Metropolis-adjusted Langevin

     tau (starting) scale of Metropolis-Hastings update

     adapt whether to adapt the scale of the proposal distribution during burnin to achieve better acceptance rates.

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

     df degrees of freedom (scalar or vector)

     scale scale (scalar or vector)

     common whether the modeled scale parameter of the inverse chi-squared distribution is (a scalar parameter) common to all n parameters.

n	dimension, if known. For internal use only.
post	whether conditional posterior sampling function should be created. For internal use only.

Value

An environment with information about the prior and possibly conditional posterior distribution(s), to be used by other package functions.

---

pr_invwishart	*Create an object containing information about an inverse Wishart prior, possibly with modeled scale matrix*

Description

Create an object containing information about an inverse Wishart prior, possibly with modeled scale matrix

Usage

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

Arguments

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

     df degrees of freedom (scalar or vector) of the chi-squared distribution(s)

     scale scale parameter(s) of the chi-squared distribution(s)

     common whether the modeled scale parameter of the inverse chi-squared distribution is (a scalar parameter) common to all n diagonal elements.

n	dimension, if known. For internal use only.

Value

An environment with information about the prior distribution used, to be used by other package functions.

References

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

---

read_draws	*Read MCMC draws from a file*

Description

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

Usage

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

Arguments

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

Value

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

Examples

```
## Not run:
# NB this example creates a file "MCdraws_e_.RData" 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, Q0=1, name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
# run the MCMC simulation and write draws of residuals to file:
sim <- MCMCsim(sampler, n.iter=500, to.file="e_")
summary(sim)
mcres <- read_draws("e_")
summary(mcres)

## End(Not run)
```

---

reg   *Create a model component object for a regression (fixed effects) com-*
                             ponent 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 de-
fault, the prior for the regression coefficients is improper uniform. If b0 or Q0 are specified the prior
becomes normal with mean b0 (default 0) and variance (matrix) sigma_^2 Q0^-1 where sigma_^2
is the overall scale parameter of the model, if any.

Usage

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

Arguments

formula	a formula specifying the predictors to be used in the model, in the same way as the right hand side of the formula argument of R's lm function. Variable names are looked up in the data frame passed as data argument to `create_sampler` or `generate_data`, or in environment(formula).
remove.redundant	
	whether redundant columns should be removed from the design matrix. Default is FALSE. But note that treatment contrasts are automatically applied to all factor variables in formula.
sparse	whether the model matrix associated with formula should be sparse. The default is to base this on a simple heuristic.
X	a (possibly sparse) design matrix can be specified directly, as an alternative to the creation of one based on formula. If X is specified formula is ignored.
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.
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.
R	optional constraint matrix for equality restrictions R'x = r where x is the vector of regression effects.
r	right hand side for the equality constraints.
S	optional constraint matrix for inequality constraints S'x >= s where x is the vector of regression effects.
s	right hand side for the inequality constraints.
lower	as an alternative to s, lower and upper may be specified for two-sided constraints lower <= S'x <= upper.
upper	as an alternative to s, lower and upper may be specified for two-sided constraints lower <= S'x <= upper.
name	the name of the model component. This name is used in the output of the MCMC simulation function `MCMCsim`. By default the name will be 'reg' with the number of the model term attached.
perm	whether permutation should be used in the Cholesky decomposition used for updating the model component's coefficient. Default is based on a simple heuristic.
debug	if TRUE a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.
e	for internal use only.

Value

an object with precomputed quantities and functions for sampling from prior or conditional posterior distributions for this model component. Only 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, Q0=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, Q=0.1, name="beta"),
  family="binomial", data=iris)
sim <- MCMCsim(sampler, burnin=100, n.iter=400)
summary(sim)
pred <- predict(sim)
str(pred)
# example with equality constrained regression effects
n <- 500
df <- data.frame(x=runif(n))
df$y <- rnorm(n, 1 + 2*df$x)
R <- matrix(1, 2, 1)
r <- 3
sampler <- create_sampler(y ~ reg(~ 1 + x, R=R, r=r, name="beta"), data=df)
sim <- MCMCsim(sampler)
summary(sim)
plot(sim, "beta")
summary(transform_dc(sim$beta, fun=function(x) crossprod_mv(R, x) - r))
```

---

residuals-fitted-values

Extract draws of fitted values or residuals from an mcdraws object

---

Description

For a model created with [create_sampler](#) and estimated using [MCMCsim](#), these functions return the posterior draws of fitted values or residuals. In the current implementation the fitted values correspond to the linear predictor and the residuals are computed as the data vector minus the fitted values, regardless of the model's distribution family. For large datasets the returned object can

become very large. One may therefore select a subset of draws or chains or use `mean.only=TRUE` to return a vector of posterior means only.

Usage

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

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

Arguments

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

Value

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

Examples

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

---

setup_cluster *Set up a cluster for parallel computing*

Description

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

Usage

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

Arguments

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

Value

an object representing the cluster.

---

set_opts *Set global options relating to computational details*

Description

Set global options relating to computational details

Usage

```
set_opts(
  auto.order.block = TRUE,
  chol.inplace = TRUE,
  chol.ordering = 0L,
  PG.approx = TRUE,
  PG.approx.m = -2L,
  CRT.approx.m = 20L
)
```

Arguments

`auto.order.block`

whether Gibbs blocks should be ordered automatically in such a way that those with the most sparse design matrices come first. This way of ordering can make Cholesky updates more efficient.

`chol.inplace`   whether sparse Cholesky updates should re-use the same memory location.

`chol.ordering` an integer passed to CHOLMOD routines determining which reordering schemes are tried to limit sparse Cholesky fill-in.

`PG.approx`   whether Polya-Gamma draws for logistic binomial models are approximated by a hybrid gamma convolution approach. If not, `BayesLogit::rpg` is used, which is exact for some values of the shape parameter.

`PG.approx.m`   if `PG.approx=TRUE`, the number of explicit gamma draws in the sum-of-gammas representation of the Polya-Gamma distribution. The remainder (infinite) convolution is approximated by a single moment-matching gamma draw. Special values are: `-2L` for a default choice depending on the value of the shape parameter and balancing performance and accuracy, `-1L` for a moment-matching normal approximation, and `0L` for a moment-matching gamma approximation.

`CRT.approx.m`   scalar integer specifying the degree of approximation to sampling from a Chinese Restaurant Table distribution. The approximation is based on Le Cam's theorem. Larger values yield a slower but more accurate sampler.

Value

This function sets or resets options in the option environment `.opts`.

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.

---

stop_cluster   *Stop a cluster*

Description

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

Usage

```
stop_cluster(cl)
```

Arguments

cl   the cluster object.

Value

NULL.

---

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

Description

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

Usage

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

Arguments

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

Value

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

Examples

```
n <- 300
dat <- data.frame(x=runif(n), f=as.factor(sample(1:7, n, replace=TRUE)))
gd <- generate_data(~ reg(~ x + f, Q0=1, name="beta"), data=dat)
dat$y <- gd$y
sampler <- create_sampler(y ~ reg(~ x + f, name="beta"), data=dat)
sim <- MCMCsim(sampler)
(summary(sim$beta))
(summary(subset(sim$beta, chains=1)))
(summary(subset(sim$beta, chains=1, draws=sample(1:ndraws(sim), 100))))
(summary(subset(sim$beta, vars=1:2)))
```

---

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

Description

Summarize a draws component (dc) object

Usage

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

Arguments

object	an object of class dc.
probs	vector of probabilities at which to evaluate quantiles.
na.rm	whether to remove NA/NaN draws in computing the summaries.
time	MCMC computation time; if specified the effective sample size per unit of time is returned in an extra column labeled 'efficiency'.
abbr	if TRUE abbreviate the labels in the output.

batch.size	number of parameter columns to process simultaneously. A larger batch size may speed things up a little, but if an out of memory error occurs it may be a good idea to use a smaller number and try again. The default is 100.
...	arguments passed to [n_eff](#).

Value

A matrix with summaries of class dc_summary.

Examples

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

---

summary.mcdraws   *Summarize an mcdraws object*

Description

Summarize an mcdraws object

Usage

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

Arguments

object	an object of class mcdraws, typically generated by function [MCMCsim](#).
vnames	optional character vector to select a subset of parameters.
probs	vector of probabilities at which to evaluate quantiles.
na.rm	whether to remove NA/NaN draws in computing the summaries.

efficiency	if TRUE the effective sample size per second of computation time is returned as well.
abbr	if TRUE abbreviate the labels in the output.
batch.size	number of parameter columns to process simultaneously for vector parameters. A larger batch size may speed things up a little, but if an out of memory error occurs it may be a good idea to use a smaller number and try again. The default is 100.
...	arguments passed to [n_eff]().

Value

A list of class mcdraws_summary summarizing object.

Examples

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

---

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

Description

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

Usage

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

Arguments

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

Value

Either a matrix or a draws component object.

Examples

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

---

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

---

Description

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

Usage

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

Arguments

factor   The name of a factor variable. The name "local_" has a special meaning,
                and assigns a different variance scale parameter to each data unit. In case of
                inverse chi-squared priors this implies that the marginal sampling distribution is
                a t distribution. In case of exponential priors the marginal sampling distribution
                is a Laplace or double exponential distribution.

prior   the prior assigned to the variance factors. Currently the prior can be inverse
                chi-squared or exponential, specified by a call to `pr_invchisq` or `pr_exp`, re-
                spectively. The default priors are inverse chi-squared with 1 degree of freedom.
                See the help pages of the prior specification functions for details on how to set
                non-default priors.

name	The name of the variance model component. This name is used in the output of the MCMC simulation function MCMCsim. By default the name will be 'vfac' with the number of the variance model term attached.
debug	If TRUE a breakpoint is set at the beginning of the posterior draw function associated with this model component. Mainly intended for developers.
e	For internal use only.

Value

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

---

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

Description

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

Usage

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

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.
Q0	prior precision matrix for the regression effects. The default is a zero matrix corresponding to a noninformative improper prior.
b0	prior mean for the regression effect. Defaults to a zero vector.
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.
e	for internal use only.

Value

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

References

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

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

---

weights.mcdraws   *Extract weights from an mcdraws object*

Description

Extract weights from an mcdraws object

Usage

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

Arguments

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

Value

A vector with (simulation means of) weights.

Examples

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

Index