

Package ‘BayesianTools’

December 20, 2017

Title General-Purpose MCMC and SMC Samplers and Tools for Bayesian Statistics

Version 0.1.4

Date 2017-12-20

Author Florian Hartig [aut, cre],
Francesco Minunno [aut],
Stefan Paul [aut],
David Cameron [ctb],
Tankred Ott [ctb]

Maintainer Florian Hartig <florian.hartig@biologie.uni-regensburg.de>

Description General-purpose MCMC and SMC samplers, as well as plot and diagnostic functions for Bayesian statistics, with a particular focus on calibrating complex system models. Implemented samplers include various Metropolis MCMC variants (including adaptive and/or delayed rejection MH), the T-walk, two differential evolution MCMCs, two DREAM MCMCs, and a sequential Monte Carlo (SMC) particle filter.

Depends R (>= 3.1.2)

License CC BY-SA 4.0

Imports coda, emulator, mvtnorm, tmvtnorm, IDPmisc, Rcpp (>= 0.12.12),
ellipse, numDeriv, msm, MASS, Matrix, stats, utils, graphics,
DHARMA, bridgesampling

Suggests DEoptim, lhs, sensitivity, knitr, rmarkdown, roxygen2,
testthat, gap

LinkingTo Rcpp

LazyData true

RoxygenNote 6.0.1

URL <https://github.com/florianhartig/BayesianTools>

BugReports <https://github.com/florianhartig/BayesianTools/issues>

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-12-20 12:13:37 UTC

R topics documented:

AM	3
applySettingsDefault	4
BayesianTools	5
bridgesample	5
checkBayesianSetup	7
convertCoda	8
correctThin	8
correlationPlot	9
createBayesianSetup	10
createBetaPrior	13
createBreakMat	14
createLikelihood	14
createMcmcSamplerList	15
createMixWithDefaults	16
createPosterior	16
createPrior	17
createPriorDensity	18
createProposalGenerator	19
createSmcSamplerList	22
createTruncatedNormalPrior	22
createUniformPrior	23
DE	24
DEzs	26
DIC	28
DR	29
DRAM	29
DREAM	30
DREAMzs	32
gelmanDiagnostics	34
generateParallelExecuter	35
generateTestDensityMultiNormal	36
getCredibleIntervals	37
getDharmaResiduals	38
getMetropolisDefaultSettings	38
getPanels	39
getPossibleSamplerTypes	39
getPredictiveDistribution	40
getPredictiveIntervals	40
getSample	41
getVolume	43
GOF	44
histMarginal	45
likelihoodAR1	46
likelihoodIidNormal	46
logSumExp	47
M	47

MAP	48
marginalLikelihood	49
marginalPlot	51
mcmcMultipleChains	52
Metropolis	53
metropolisRatio	55
plotHist	55
plotSensitivity	56
plotTimeSeries	56
plotTimeSeriesResiduals	57
plotTimeSeriesResults	58
rescale	59
runMCMC	59
sampleEquallySpaced	61
sampleMetropolis	62
setupStartProposal	62
smcSampler	63
stopParallel	64
testDensityBanana	65
testDensityInfinity	65
testDensityMultiNormal	66
testDensityNormal	66
testLinearModel	67
tracePlot	67
Twalk	68
updateProposalGenerator	69
violinPlot	70
VSEM	71
vsemC	75
VSEMcreateLikelihood	76
VSEMcreatePAR	76
VSEMgetDefaults	77
WAIC	77
Index	80

Description

The Adaptive Metropolis Algorithm (Haario et al. 2001)

Usage

```
AM(startValue = NULL, iterations = 10000, nBI = 0, parmin = NULL,
    parmax = NULL, FUN, f = 1, eps = 0)
```

Arguments

startValue	vector with the start values for the algorithm. Can be NULL if FUN is of class BayesianSetup. In this case startValues are sampled from the prior.
iterations	iterations to run
nBI	number of burnin
parmin	minimum values for the parameter vector or NULL if FUN is of class BayesianSetup
parmax	maximum values for the parameter vector or NULL if FUN is of class BayesianSetup
FUN	function to be sampled from or object of class bayesianSetup
f	scaling factor
eps	small number to avoid singularity

Author(s)

Francesco Minunno

References

Haario, Heikki, Eero Saksman, and Johanna Tamminen. "An adaptive Metropolis algorithm." *Bernoulli* (2001): 223-242.

applySettingsDefault *Provides the default settings for the different samplers in runMCMC*

Description

Provides the default settings for the different samplers in runMCMC

Usage

```
applySettingsDefault(settings = NULL, sampler = "DEzs", check = FALSE)
```

Arguments

settings	optional list with parameters that will be used instead of the defaults
sampler	one of the samplers in runMCMC
check	logical determines whether parameters should be checked for consistency

Details

see [runMCMC](#)

Author(s)

Florian Hartig

 BayesianTools

BayesianTools

Description

A package with general-purpose MCMC and SMC samplers, as well as plots and diagnostic functions for Bayesian statistics

Details

A package with general-purpose MCMC and SMC samplers, as well as plots and diagnostic functions for Bayesian statistics, particularly for process-based models.

The package contains 2 central functions, `createBayesianSetup`, which creates a standardized Bayesian setup with likelihood and priors, and `runMCMC`, which allows to run various MCMC and SMC samplers.

The package can of course also be used for general (non-Bayesian) target functions.

To use the package, a first step to use `createBayesianSetup` to create a `BayesianSetup`, which usually contains prior and likelihood densities, or in general a target function.

Those can be sampled with `runMCMC`, which can call a number of general purpose Metropolis sampler, including the `Metropolis` that allows to specify various popular Metropolis variants such as adaptive and/or delayed rejection Metropolis; two variants of differential evolution MCMC `DE`, `DEzs`, two variants of DREAM `DREAM` and `DREAMzs`, the `Twalk` MCMC, and a Sequential Monte Carlo sampler `smcSampler`.

The output of `runMCMC` is of class `mcmcSampler` / `smcSampler` if one run is performed, or `mcmcSamplerList` / `smcSamplerList` if several sampler are run. Various functions are available for plotting, model comparison (DIC, marginal likelihood), or to use the output as a new prior.

For details on how to use the package, run `vignette("BayesianTools", package="BayesianTools")`.

To get the suggested citation, run `citation("BayesianTools")`

To report bugs or ask for help, post a [reproducible example](#) via the BayesianTools [issue tracker](#) on GitHub.

Acknowledgements: The creation of this package was facilitated through meetings of Cost Action FP 1304 Profound.

 bridgesample

Calculates the marginal likelihood of a chain via bridge sampling

Description

Calculates the marginal likelihood of a chain via bridge sampling

Usage

```
bridgesample(chain, nParams, lower = NULL, upper = NULL, ...)
```

Arguments

chain	a single mcmc chain with samples as rows and parameters and posterior density as columns.
nParams	number of parameters
lower	optional - lower bounds of the prior
upper	optional - upper bounds of the prior
...	arguments passed to bridge_sampler

Details

This function uses "bridge_sampler" from the package "bridgesampling".

Author(s)

Tankred Ott

See Also

[marginalLikelihood](#)

Examples

```
means <- c(0, 1, 2)
sds <- c(1, 0.6, 3)

# log-likelihood
ll <- function (x) {
  return(sum(dnorm(x, mean = means, sd = sds, log = TRUE)))
}

# lower and upper bounds for prior
lb <- rep(-10, 3)
ub <- rep(10, 3)

# create setup and run MCMC
setup <- createBayesianSetup(likelihood = ll,
                             lower = lb,
                             upper = ub)

out <- runMCMC(bayesianSetup = setup,
               settings = list(iterations = 1000),
               sampler = "DEzs")

# sample from MCMC output with "burn-in" of 25%
sample <- getSample(out$chain, start = 250, numSamples = 500)

# use bridge sampling to get marginal likelihood
bs_result <- bridgesample(chain = sample,
                          nParams = out$setup$numPars,
                          lower = lb,
```

```
bs_result          upper = ub)
```

checkBayesianSetup *Checks if an object is of class 'BayesianSetup'*

Description

Function used to assure that an object is of class 'BayesianSetup'. If you pass a function, it is converted to an object of class 'BayesianSetup' (using [createBayesianSetup](#)) before it is returned.

Usage

```
checkBayesianSetup(bayesianSetup, parallel = F)
```

Arguments

`bayesianSetup` either object of class `bayesianSetup` or a log posterior function

`parallel` if `bayesianSetup` is a function, this will set the parallelization option for the class `BayesianSetup` that is created internally. If `bayesianSetup` is already a `BayesianSetup`, then this will check if `parallel = T` is requested but not supported by the `BayesianSetup`. This option is for internal use in the samplers

Note

The recommended option to use this function in the samplers is to have `parallel` with default `NULL` in the samplers, so that `checkBayesianSetup` with a function will create a `bayesianSetup` without parallelization, while it will do nothing with an existing `BayesianSetup`. If the user sets parallelization, it will set the appropriate parallelization for a function, and check in case of an existing `BayesianSetup`. The `checkBayesianSetup` call in the samplers should then be followed by a check for `parallel = NULL` in sampler, in which case `parallel` can be set from the `BayesianSetup`

Author(s)

Florian Hartig

See Also

[createBayesianSetup](#)

convertCoda	<i>Convert coda::mcmc objects to BayesianTools::mcmcSampler</i>
-------------	---

Description

Function is used to make the plot and diagnostic functions available for coda::mcmc objects

Usage

```
convertCoda(sampler, names = NULL, info = NULL, likelihood = NULL)
```

Arguments

sampler	An object of class mcmc or mcmc.list
names	vector giving the parameter names (optional)
info	matrix (or list with matrices for mcmc.list objects) with three columns containing log posterior, log likelihood and log prior of the sampler for each time step (optional; but see Details)
likelihood	likelihood function used in the sampling (see Details)

Details

The parameter 'likelihood' is optional for most functions but can be needed e.g for using the [DIC](#) function.

Also the parameter info is optional for most uses. However for some functions (e.g. [MAP](#)) the matrix or single columns (e.g. log posterior) are necessary for the diagnostics.

correctThin	<i>Checks if thin is consistent with nTotalSamples samples and if not corrects it.</i>
-------------	--

Description

Checks if thin is consistent with nTotalSamples samples and if not corrects it.

Usage

```
correctThin(nTotalSamples, thin, autoThinFraction = 0.001)
```

Arguments

nTotalSamples	total number of rows/samples
thin	thinning
autoThinFraction	fraction of the data that will be sampled when thin is set to "auto". E.g. 0.5 means thin will be nTotalSamples * 0.5. The resulting thin value is rounded down to the next integer.

Details

Checks if the thin argument is consistent with the data consisting of nTotalSamples samples/rows and corrects thin if not.

Author(s)

Tankred Ott

Tankred Ott

correlationPlot *Flexible function to create correlation density plots*

Description

Flexible function to create correlation density plots

Usage

```
correlationPlot(mat, density = "smooth", thin = "auto",
  method = "pearson", whichParameters = NULL, ...)
```

Arguments

mat	object of class "bayesianOutput" or a matrix or data frame of variables
density	type of plot to do. Either "smooth" (default), "corellipseCor", or "ellipse"
thin	thinning of the matrix to make things faster. Default is to thin to 5000
method	method for calculating correlations. Possible choices are "pearson" (default), "kendall" and "spearman"
whichParameters	indices of parameters that should be plotted
...	additional parameters to pass on to the getSample , for example parametersOnly =F, or start = 1000

Author(s)

Florian Hartig

References

The code for the correlation density plot originates from Hartig, F.; Dislich, C.; Wiegand, T. & Huth, A. (2014) Technical Note: Approximate Bayesian parameterization of a process-based tropical forest model. *Biogeosciences*, 11, 1261-1272.

See Also

[marginalPlot](#)
[plotTimeSeries](#)
[tracePlot](#)

Examples

```

dat = generateTestDensityMultiNormal(n = 100000, sample = TRUE)
correlationPlot(dat(100000))

```

createBayesianSetup *Creates a standardized collection of prior, likelihood and posterior functions, including error checks etc.*

Description

Creates a standardized collection of prior, likelihood and posterior functions, including error checks etc.

Usage

```

createBayesianSetup(likelihood, prior = NULL, priorSampler = NULL,
  parallel = FALSE, lower = NULL, upper = NULL, best = NULL,
  names = NULL, parallelOptions = list(variables = "all", packages = "all",
  dlls = NULL), catchDuplicates = FALSE)

```

Arguments

likelihood	log likelihood density function
prior	either a prior class (see createPrior) or a log prior density
priorSampler	if a prior density (and not a prior class) is provided to prior, the optional prior sampling function can be provided here
parallel	parallelization option. Default is F. Other options are T, or "external". See details.
lower	vector with lower prior limits
upper	vector with upper prior limits
best	vector with best values
names	optional vector with parameter names
parallelOptions	list containing three lists. First "packages" determines the R packages necessary to run the likelihood function. Second "variables" the objects in the global environment needed to run the likelihood function and third "dlls" the DLLs needed to run the likelihood function (see Details and Examples).

catchDuplicates

Logical, determines whether unique parameter combinations should only be evaluated once. Only used when the likelihood accepts a matrix with parameter as columns.

Details

If lower and upper (and optionally best) but not a prior object are passed, the function will create a uniform prior with the limits lower and upper. If a prior object and lower and upper are passed, the function will ignore lower and upper, and will only use the prior object.

For parallelization, option T means that an automatic parallelization via R is attempted, or "external", in which case it is assumed that the likelihood is already parallelized. In this case it needs to accept a matrix with parameters as columns. Further you can specify the packages, objects and DLLs that are exported to the cluster. By default a copy of your workspace is exported. However, depending on your workspace this can be very inefficient.

For more details, make sure to read the vignette (run `vignette("BayesianTools", package="BayesianTools")`)

Author(s)

Florian Hartig

See Also

[checkBayesianSetup](#)
[createLikelihood](#)
[createPrior](#)

Examples

```
ll <- function(x) sum(dnorm(x, log = TRUE))

test <- createBayesianSetup(ll, prior = NULL, priorSampler = NULL, lower = -10, upper = 10)
str(test)
test$prior$density(0)

test$likelihood$density(c(1,1))
test$likelihood$density(1)
test$posterior$density(1)
test$posterior$density(1, returnAll = TRUE)

test$likelihood$density(matrix(rep(1,4), nrow = 2))
#test$posterior$density(matrix(rep(1,4), nrow = 2), returnAll = TRUE)
test$likelihood$density(matrix(rep(1,4), nrow = 4))

## Example of how to use parallelization using the VSEM model
# Note that the parallelization produces overhead and is not always
# speeding things up. In this example, due to the small
# computational cost of the VSEM the parallelization is
```

```

# most likely to reduce the speed of the sampler.

# Creating reference data
PAR <- VSEMcreatePAR(1:1000)
refPars <- VSEMgetDefaults()
refPars[12,] <- c(0.2, 0.001, 1)
rownames(refPars)[12] <- "error-sd"

referenceData <- VSEM(refPars$best[1:11], PAR)
obs = apply(referenceData, 2, function(x) x + rnorm(length(x),
                                                    sd = abs(x) * refPars$best[12]))

# Selecting parameters
parSel = c(1:6, 12)

## Building the likelihood function
likelihood <- function(par, sum = TRUE){
  x = refPars$best
  x[parSel] = par
  predicted <- VSEM(x[1:11], PAR)
  diff = c(predicted[,1:3] - obs[,1:3])
  llValues = dnorm(diff, sd = max(abs(c(predicted[,1:3])),0.0001) * x[12], log = TRUE)
  if (sum == FALSE) return(llValues)
  else return(sum(llValues))
}

# Prior
prior <- createUniformPrior(lower = refPars$lower[parSel], upper = refPars$upper[parSel])

####
## Definition of the packages and objects that are exported to the cluster.
## These are the objects that are used in the likelihood function.
opts <- list(packages = list("BayesianTools"), variables = list("refPars", "obs", "PAR" ),
            dlls = NULL)

# Create Bayesian Setup
BSVSEM <- createBayesianSetup(likelihood, prior, best = refPars$best[parSel],
                             names = rownames(refPars)[parSel], parallel = 2,
                             parallelOptions = opts)

## The bayesianSetup can now be used in the runMCMC function.
# Note that not all samplers can make use of parallel
# computing.

# Remove the Bayesian Setup and close the cluster
stopParallel(BSVSEM)
rm(BSVSEM)

```

createBetaPrior	<i>Convenience function to create a beta prior</i>
-----------------	--

Description

Convenience function to create a beta prior

Usage

```
createBetaPrior(a, b, lower = 0, upper = 1)
```

Arguments

a	shape1 of the beta distribution
b	shape2 of the beta distribution
lower	lower values for the parameters
upper	upper values for the parameters

Details

This creates a beta prior, assuming that lower / upper values for parameters are fixed. The beta is calculated relative to this lower / upper space.

Note

for details see [createPrior](#)

Author(s)

Florian Hartig

See Also

[createPriorDensity](#)
[createPrior](#)
[createTruncatedNormalPrior](#)
[createUniformPrior](#)
[createBayesianSetup](#)

createBreakMat	<i>create break matrix</i>
----------------	----------------------------

Description

A function for use with plotHist. Creates a matrix representing breaks of a histogram. The matrix will contain the upper bounds, the lower bounds and the frequencies of the breaks in the columns, and the individual breaks in the rows.

Usage

```
createBreakMat(x, breaks = 15, scale = FALSE)
```

Arguments

x	vector of values
breaks	number of breaks
scale	logical, if TRUE, the area within the rectangle will be scaled to one (density)

Author(s)

Tankred Ott

createLikelihood	<i>Creates a standardized likelihood class#'</i>
------------------	--

Description

Creates a standardized likelihood class#'

Usage

```
createlikelihood(likelihood, names = NULL, parallel = F,  
  catchDuplicates = T, sampler = NULL, parallelOptions = NULL)
```

Arguments

likelihood	Log likelihood density
names	Parameter names (optional)
parallel	parallelization , either i) no parallelization -> F, ii) native R parallelization -> T / "auto" will select n-1 of your available cores, or provide a number for how many cores to use, or iii) external parallelization -> "external". External means that the likelihood is already able to execute parallel runs in form of a matrix with

catchDuplicates	Logical, determines whether unique parameter combinations should only be evaluated once. Only used when the likelihood accepts a matrix with parameter as columns.
sampler	sampler
parallelOptions	list containing two lists. First "packages" determines the R packages necessary to run the likelihood function. Second "objects" the objects in the global environment needed to run the likelihood function (for details see createBayesianSetup).

Author(s)

Florian Hartig

See Also

[likelihoodIidNormal](#)
[likelihoodAR1](#)

createMcmcSamplerList *Convenience function to create an object of class mcmcSamplerList from a list of mcmc samplers*

Description

Convenience function to create an object of class mcmcSamplerList from a list of mcmc samplers

Usage

```
createMcmcSamplerList(mcmcList)
```

Arguments

mcmcList a list with each object being an mcmcSampler

Value

Object of class "mcmcSamplerList"

Author(s)

Florian Hartig

`createMixWithDefaults` *Allows to mix a given parameter vector with a default parameter vector*

Description

This function is deprecated and will be removed by v0.2.

Usage

```
createMixWithDefaults(pars, defaults, locations)
```

Arguments

<code>pars</code>	vector with new parameter values
<code>defaults</code>	vector with default parameter values
<code>locations</code>	indices of the new parameter values

`createPosterior` *Creates a standardized posterior class*

Description

Creates a standardized posterior class

Usage

```
createPosterior(prior, likelihood)
```

Arguments

<code>prior</code>	prior class
<code>likelihood</code>	Log likelihood density

Details

Function is internally used in [createBayesianSetup](#) to create a standardized posterior class.

Author(s)

Florian Hartig

createPrior	<i>Creates a standardized prior class</i>
-------------	---

Description

Creates a standardized prior class

Usage

```
createPrior(density = NULL, sampler = NULL, lower = NULL, upper = NULL,  
            best = NULL)
```

Arguments

density	Prior density
sampler	Sampling function for density (optional)
lower	vector with lower bounds of parameters
upper	vector with upper bounds of parameter
best	vector with "best" parameter values

Details

This is the general prior generator. It is highly recommended to not only implement the density, but also the sampler function. If this is not done, the user will have to provide explicit starting values for many of the MCMC samplers. Note the existing, more specialized prior function. If your prior can be created by those, they are preferred. Note also that priors can be created from an existing MCMC output from BT, or another MCMC sample, via [createPriorDensity](#).

Note

min and max truncate, but not re-normalize the prior density (so, if a pdf that integrated to one is truncated, the integral will in general be smaller than one). For MCMC sampling, this doesn't make a difference, but if absolute values of the prior density are a concern, one should provide a truncated density function for the prior.

Author(s)

Florian Hartig

See Also

[createPriorDensity](#)
[createBetaPrior](#)
[createUniformPrior](#)
[createTruncatedNormalPrior](#)
[createBayesianSetup](#)

Examples

```

# Create a general prior distribution by specifying an arbitrary density function and a
# corresponding sampling function
density = function(par){
  d1 = dunif(par[1], -2,6, log =TRUE)
  d2 = dnorm(par[2], mean= 2, sd = 3, log =TRUE)
  return(d1 + d2)
}

# The sampling is optional but recommended because the MCMCs can generate automatic starting
# conditions if this is provided
sampler = function(n=1){
  d1 = runif(n, -2,6)
  d2 = rnorm(n, mean= 2, sd = 3)
  return(cbind(d1,d2))
}

prior <- createPrior(density = density, sampler = sampler,
                    lower = c(-3,-3), upper = c(3,3), best = NULL)

# Use this prior in an MCMC

ll <- function(x) sum(dnorm(x, log = TRUE)) # multivariate normal ll
bayesianSetup <- createBayesianSetup(likelihood = ll, prior = prior)

settings = list(iterations = 1000)
out <- runMCMC(bayesianSetup = bayesianSetup, settings = settings)

# see ?createPriorDensity for how to create a new prior from this output

```

createPriorDensity *Fits a density function to a multivariate sample*

Description

Fits a density function to a multivariate sample

Usage

```

createPriorDensity(sampler, method = "multivariate", eps = 1e-10,
                  lower = NULL, upper = NULL, best = NULL, ...)

```

Arguments

sampler	an object of class BayesianOutput or a matrix
method	method to generate prior - default and currently only option is multivariate
eps	numerical precision to avoid singularity

lower	vector with lower bounds of parameter for the new prior, independent of the input sample
upper	vector with upper bounds of parameter for the new prior, independent of the input sample
best	vector with "best" values of parameter for the new prior, independent of the input sample
...	parameters to pass on to the getSample function

Author(s)

Florian Hartig

See Also

[createPrior](#)
[createBetaPrior](#)
[createTruncatedNormalPrior](#)
[createUniformPrior](#)
[createBayesianSetup](#)

Examples

```
# Create a BayesianSetup
ll <- generateTestDensityMultiNormal(sigma = "no correlation")
bayesianSetup = createBayesianSetup(likelihood = ll,
                                   lower = rep(-10, 3),
                                   upper = rep(10, 3))

settings = list(iterations = 2500)
out <- runMCMC(bayesianSetup = bayesianSetup, settings = settings)

newPrior = createPriorDensity(out, method = "multivariate",
                              eps = 1e-10, lower = rep(-10, 3),
                              upper = rep(10, 3), best = NULL)

bayesianSetup <- createBayesianSetup(likelihood = ll, prior = newPrior)

settings = list(iterations = 1000)
out <- runMCMC(bayesianSetup = bayesianSetup, settings = settings)
```

createProposalGenerator

Factory that creates a proposal generator

Description

Factory that creates a proposal generator

Usage

```
createProposalGenerator(covariance, gibbsProbabilities = NULL,
  gibbsWeights = NULL, otherDistribution = NULL,
  otherDistributionLocation = NULL, otherDistributionScaled = F,
  message = F, method = "chol", scalingFactor = 2.38)
```

Arguments

covariance	covariance matrix. Can also be vector of the sqrt of diagonal elements → standard deviation
gibbsProbabilities	optional probabilities for the number of parameters to vary in a Metropolis within gibbs style - for 4 parameters, c(1,1,0.5,0) means that at most 3 parameters will be varied, and it is double as likely to vary one or two than varying 3
gibbsWeights	optional probabilities for parameters to be varied in a Metropolis within gibbs style - default is equal weight for all parameters - for 4 parameters, c(1,1,1,100) would mean that if 2 parameters would be selected, parameter 4 would be 100 times more likely to be picked than the others. If 4 is selected, the remaining parameters have equal probability.
otherDistribution	optional additional distribution to be mixed with the default multivariate normal. The distribution needs to accept a parameter vector (to allow for the option of making the distribution dependent on the parameter values), but it is still assumed that the change from the current values is returned, not the new absolute values.
otherDistributionLocation	a vector with 0 and 1, denoting which parameters are modified by the otherDistribution
otherDistributionScaled	should the other distribution be scaled if gibbs updates are calculated?
message	print out parameter settings
method	method for covariance decomposition
scalingFactor	scaling factor for the proposals

Author(s)

Florian Hartig

See Also

[updateProposalGenerator](#)

Examples

```
testMatrix = matrix(rep(c(0,0,0,0), 1000), ncol = 4)
testVector = c(0,0,0,0)

##Standard multivariate normal proposal generator

testGenerator <- createProposalGenerator(covariance = c(1,1,1,1), message = TRUE)

methods(class = "proposalGenerator")
print(testGenerator)

x = testGenerator$returnProposal(testVector)
x

x <- testGenerator$returnProposalMatrix(testMatrix)
boxplot(x)

##Changing the covariance
testGenerator$covariance = diag(rep(100,4))
testGenerator <- testGenerator$updateProposalGenerator(testGenerator, message = TRUE)

testGenerator$returnProposal(testVector)
x <- testGenerator$returnProposalMatrix(testMatrix)
boxplot(x)

##-Changing the gibbs probabilities / probability to modify 1-n parameters

testGenerator$gibbsProbabilities = c(1,1,0,0)
testGenerator <- testGenerator$updateProposalGenerator(testGenerator)

testGenerator$returnProposal(testVector)
x <- testGenerator$returnProposalMatrix(testMatrix)
boxplot(x)

##-Changing the gibbs weights / probability to pick each parameter

testGenerator$gibbsWeights = c(0.3,0.3,0.3,100)
testGenerator <- testGenerator$updateProposalGenerator(testGenerator)

testGenerator$returnProposal(testVector)
x <- testGenerator$returnProposalMatrix(testMatrix)
boxplot(x)

##-Adding another function

otherFunction <- function(x) sample.int(10,1)

testGenerator <- createProposalGenerator(
```

```

covariance = c(1,1,1),
otherDistribution = otherFunction,
otherDistributionLocation = c(0,0,0,1),
otherDistributionScaled = TRUE
)

testGenerator$returnProposal(testVector)
x <- testGenerator$returnProposalMatrix(testMatrix)
boxplot(x)
table(x[,4])

```

`createSmcSamplerList` *Convenience function to create an object of class `SMCSamplerList` from a list of mcmc samplers*

Description

Convenience function to create an object of class `SMCSamplerList` from a list of mcmc samplers

Usage

```
createSmcSamplerList(...)
```

Arguments

... a list of MCMC samplers

Value

a list of class `smcSamplerList` with each object being an `smcSampler`

Author(s)

Florian Hartig

`createTruncatedNormalPrior`
Convenience function to create a truncated normal prior

Description

Convenience function to create a truncated normal prior

Usage

```
createTruncatedNormalPrior(mean, sd, lower, upper)
```

Arguments

mean	best estimate for each parameter
sd	standard deviation
lower	vector of lower prior range for all parameters
upper	vector of upper prior range for all parameters

Note

for details see [createPrior](#)

Author(s)

Florian Hartig

See Also

[createPriorDensity](#)
[createPrior](#)
[createBetaPrior](#)
[createUniformPrior](#)
[createBayesianSetup](#)

Examples

```
prior <- createTruncatedNormalPrior(c(0,0),c(0.4,5), lower = c(-2,-2), upper = c(1,1))
prior$density(c(2,3))
prior$density(c(0.2,0.9))
prior$sampler()
```

createUniformPrior *Convenience function to create a simple uniform prior distribution*

Description

Convenience function to create a simple uniform prior distribution

Usage

```
createUniformPrior(lower, upper, best = NULL)
```

Arguments

lower	vector of lower prior range for all parameters
upper	vector of upper prior range for all parameters
best	vector with "best" values for all parameters

Note

for details see [createPrior](#)

Author(s)

Florian Hartig

See Also

[createPriorDensity](#), [createPrior](#), [createBetaPrior](#), [createTruncatedNormalPrior](#), [createBayesianSetup](#)

Examples

```
set.seed(1)

prior <- createUniformPrior(lower = c(0,0), upper = c(0.4,5))

# c(2, 3) outside of limits
prior$density(c(2, 3))
# -Inf

# c(0.2, 2) within limits
prior$density(c(0.2, 2))
# -0.6931472

# sample from prior
prior$sampler()
# [1] 0.2291413 4.5410389

## the prior object can be passed to createBayesianSetup()

# log-likelihood density function (needed for createBayesianSetup)
ll <- function(x) sum(dnorm(x, log = TRUE))

setup <- createBayesianSetup(prior = prior, likelihood = ll)
```

Description

Differential-Evolution MCMC

Usage

```
DE(bayesianSetup, settings = list(startValue = NULL, iterations = 10000, f =
  -2.38, burnin = 0, thin = 1, eps = 0, consoleUpdates = 100, blockUpdate =
  list("none", k = NULL, h = NULL, pSel = NULL, pGroup = NULL, groupStart =
  1000, groupIntervall = 1000), currentChain = 1, message = TRUE))
```

Arguments

bayesianSetup	a BayesianSetup with the posterior density function to be sampled from
settings	list with parameter settings
startValue	(optional) either a matrix with start population, a number to define the number of chains that are run or a function that samples a starting population.
iterations	number of function evaluations.
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thinning parameter. Determines the interval in which values are recorded.
f	scaling factor gamma
eps	small number to avoid singularity
blockUpdate	list determining whether parameters should be updated in blocks. For possible settings see Details.
message	logical determines whether the sampler's progress should be printed

Details

For blockUpdate the first element in the list determines the type of blocking. Possible choices are

- "none" (default), no blocking of parameters
- "correlation" blocking based on correlation of parameters. Using h or k (see below)
- "random" random blocking. Using k (see below)
- "user" user defined groups. Using groups (see below)

Further seven parameters can be specified. "k" determined the number of groups, "h" the strength of the correlation used to group parameter and "groups" is used for user defined groups. "groups" is a vector containing the group number for each parameter. E.g. for three parameters with the first two in one group, "groups" would be c(1,1,2). Further pSel and pGroup can be used to influence the choice of groups. In the sampling process a number of groups is randomly drawn and updated. pSel is a vector containing relative probabilities for an update of the respective number of groups. E.g. for always updating only one group pSel = 1. For updating one or two groups with the same probability pSel = c(1,1). By default all numbers have the same probability. The same principle is used in pGroup. Here the user can influence the probability of each group to be updated. By default all groups have the same probability. Finally "groupStart" defines the starting point of the groupUpdate and "groupIntervall" the intervall in which the groups are evaluated.

Author(s)

Francesco Minunno and Stefan Paul

References

Braak, Cajo JF Ter. "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces." *Statistics and Computing* 16.3 (2006): 239-249.

See Also

[DEzs](#)

 DEzs

Differential-Evolution MCMC zs

Description

Differential-Evolution MCMC zs

Usage

```
DEzs(bayesianSetup, settings = list(iterations = 10000, Z = NULL, startValue =
  NULL, pSnooker = 0.1, burnin = 0, thin = 1, f = 2.38, eps = 0, parallel =
  NULL, pGamma1 = 0.1, eps.mult = 0.2, eps.add = 0, consoleUpdates = 100,
  zUpdateFrequency = 1, currentChain = 1, blockUpdate = list("none", k = NULL, h
  = NULL, pSel = NULL, pGroup = NULL, groupStart = 1000, groupIntervall = 1000),
  message = TRUE))
```

Arguments

bayesianSetup	a BayesianSetup with the posterior density function to be sampled from
settings	list with parameter settings
startValue	(optional) either a matrix with start population, a number to define the number of chains that are run or a function that samples a starting population.
Z	starting Z population
iterations	iterations to run
pSnooker	probability of Snooker update
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thinning parameter. Determines the interval in which values are recorded.
eps	small number to avoid singularity
f	scaling factor gamma
parallel	logical, determines weather parallel computing should be attempted (see details)
pGamma1	probability determining the frequency with which the scaling is set to 1 (allows jumps between modes)
eps.mult	random term (multiplicative error)

eps.add	random term
blockUpdate	list determining whether parameters should be updated in blocks. For possible settings see Details.
message	logical determines whether the sampler's progress should be printed

Details

For parallel computing FUN needs to take a matrix of proposals

For blockUpdate the first element in the list determines the type of blocking. Possible choices are

- "none" (default), no blocking of parameters
- "correlation" blocking based on correlation of parameters. Using h or k (see below)
- "random" random blocking. Using k (see below)
- "user" user defined groups. Using groups (see below)

Further seven parameters can be specified. "k" determined the number of groups, "h" the strength of the correlation used to group parameter and "groups" is used for user defined groups. "groups" is a vector containing the group number for each parameter. E.g. for three parameters with the first two in one group, "groups" would be c(1,1,2). Further pSel and pGroup can be used to influence the choice of groups. In the sampling process a number of groups is randomly drawn and updated. pSel is a vector containing relative probabilities for an update of the respective number of groups. E.g. for always updating only one group pSel = 1. For updating one or two groups with the same probability pSel = c(1,1). By default all numbers have the same probability. The same principle is used in pGroup. Here the user can influence the probability of each group to be updated. By default all groups have the same probability. Finally "groupStart" defines the starting point of the groupUpdate and "groupIntervall" the intervall in which the groups are evaluated.

Author(s)

Francesco Minunno and Stefan Paul

References

ter Braak C. J. F., and Vrugt J. A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. *Statistics and Computing* <http://dx.doi.org/10.1007/s11222-008-9104-9>

See Also

[DE](#)

DIC *Deviance information criterion*

Description

Deviance information criterion

Usage

DIC(sampler, ...)

Arguments

sampler An object of class `bayesianOutput` (`mcmcSampler`, `smcSampler`, or `mcmcList`)
... further arguments passed to `getSample`

Details

Output: list with the following elements:
DIC : Deviance Information Criterion
IC : Bayesian Predictive Information Criterion
pD : Effective number of parameters ($pD = \bar{D} - D_{hat}$)
pV : Effective number of parameters ($pV = \text{var}(D)/2$)
Dbar : Expected value of the deviance over the posterior
Dhat : Deviance at the mean posterior estimate

Author(s)

Florian Hartig

References

Spiegelhalter, D. J.; Best, N. G.; Carlin, B. P. & van der Linde, A. (2002) Bayesian measures of model complexity and fit. *J. Roy. Stat. Soc. B*, 64, 583-639.

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

See Also

[WAIC](#), [MAP](#), [marginalLikelihood](#)

DR *The Delayed Rejection Algorithm*

Description

The Delayed Rejection Algorithm (Tierney and Mira, 1999)

Usage

```
DR(startValue = NULL, iterations = 10000, nBI = 0, parmin = NULL,
    parmax = NULL, f1 = 1, f2 = 0.5, FUN)
```

Arguments

startValue	vector with the start values for the algorithm. Can be NULL if FUN is of class BayesianSetup. In this case startValues are sampled from the prior.
iterations	iterations to run
nBI	number of burnin
parmin	minimum values for the parameter vector or NULL if FUN is of class BayesianSetup
parmax	maximum values for the parameter vector or NULL if FUN is of class BayesianSetup
f1	scaling factor for first proposal
f2	scaling factor for second proposal
FUN	function to be sampled from or object of class bayesianSetup

Author(s)

Francesco Minunno

References

Tierney, Luke, and Antonietta Mira. "Some adaptive Monte Carlo methods for Bayesian inference." *Statistics in medicine* 18.1718 (1999): 2507-2515.

DRAM *The Delayed Rejection Adaptive Metropolis Algorithm*

Description

The Delayed Rejection Adaptive Metropolis Algorithm (Haario et al. 2001)

Usage

```
DRAM(startValue = NULL, iterations = 10000, nBI = 0, parmin = NULL,
    parmax = NULL, FUN, f = 1, eps = 0)
```

Arguments

startValue	vector with the start values for the algorithm. Can be NULL if FUN is of class BayesianSetup. In this case startValues are sampled from the prior.
iterations	iterations to run
nBI	number of burnin
parmin	minimum values for the parameter vector or NULL if FUN is of class BayesianSetup
parmax	maximum values for the parameter vector or NULL if FUN is of class BayesianSetup
FUN	function to be sampled from
f	scaling factor
eps	small number to avoid singularity or object of class bayesianSetup

Author(s)

Francesco Minunno

References

Haario, Heikki, Eero Saksman, and Johanna Tamminen. "An adaptive Metropolis algorithm." *Bernoulli* (2001): 223-242.

DREAM

DREAM

Description

DREAM

Usage

```
DREAM(bayesianSetup, settings = list(iterations = 10000, nCR = 3, gamma =
  NULL, eps = 0, e = 0.05, pCRupdate = TRUE, updateInterval = 10, burnin = 0,
  thin = 1, adaptation = 0.2, DEpairs = 2, consoleUpdates = 10, startValue =
  NULL, currentChain = 1, message = TRUE))
```

Arguments

bayesianSetup	Object of class 'bayesianSetup' or 'bayesianOuput'.
settings	list with parameter values
iterations	Number of model evaluations
nCR	parameter determining the number of cross-over proposals. If nCR = 1 all parameters are updated jointly.
updateInterval	determining the intervall for the pCR update
gamma	Kurtosis parameter Bayesian Inference Scheme. Default is 0.

eps	Ergodicity term. Default to 0.
e	Ergodicity term. Default to 5e-2.
pCRupdate	Update of crossover probabilities, default is TRUE
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thin thinning parameter. Determines the interval in which values are recorded.
adaptation	Number or percentage of samples that are used for the adaptation in DREAM (see Details).
DEpairs	Number of pairs used to generate proposal
startValue	either a matrix containing the start values (see details), an integer to define the number of chains that are run, a function to sample the start values or NULL, in which case the values are sampled from the prior.
consoleUpdates	Intervall in which the sampling progress is printed to the console
message	logical determines whether the sampler's progress should be printed

Details

Insted of a `bayesianSetup`, the function can take the output of a previous run to restart the sampler from the last iteration. Due to the sampler's internal structure you can only use the output of DREAM. If you provide a matrix with start values the number of rows determines the number of chains that are run. The number of coloumns must be equivalent to the number of parameters in your `bayesianSetup`.

There are several small differences in the algorithm presented here compared to the original paper by Vrugt et al. (2009). Mainly the algorithm implemented here does not have an automatic stopping criterion. Hence, it will always run the number of iterations specified by the user. Also, convergence is not monitored and left to the user. This can easily be done with `coda::gelman.diag(chain)`. Further the proposed delayed rejectio step in Vrugt et al. (2009) is not implemented here.

During the adaptation phase DREAM is running two mechanisms to enhance the sampler's efficiency. First the disribution of crossover values is tuned to favor large jumps in the parameter space. The crossover probabilities determine how many parameters are updated simultaneously. Second outlier chains are replanced as they can largely deteriorate the sampler's performance. However, these steps destroy the detailed balance of the chain. Consequently these parts of the chain should be discarded when summarizing posterior moments. This can be done automatically during the sampling process (i.e. `burnin > adaptation`) or subsequently by the user. We chose to distinguish between the burnin and adaptation phase to allow the user more flexibility in the sampler's settings.

Value

`mcmc.object` containing the following elements: `chains`, `X`, `pCR`

Author(s)

Stefan Paul

References

Vrugt, Jasper A., et al. "Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling." *International Journal of Nonlinear Sciences and Numerical Simulation* 10.3 (2009): 273-290.

See Also

[DREAMzs](#)

DREAMzs

DREAMzs

Description

DREAMzs

Usage

```
DREAMzs(bayesianSetup, settings = list(iterations = 10000, nCR = 3, gamma =
  NULL, eps = 0, e = 0.05, pCRupdate = FALSE, updateInterval = 10, burnin = 0,
  thin = 1, adaptation = 0.2, parallel = NULL, Z = NULL, ZupdateFrequency = 10,
  pSnooker = 0.1, DEpairs = 2, consoleUpdates = 10, startValue = NULL,
  currentChain = 1, message = FALSE))
```

Arguments

bayesianSetup	Object of class 'bayesianSetup' or 'bayesianOutput'.
settings	list with parameter values
iterations	Number of model evaluations
nCR	parameter determining the number of cross-over proposals. If nCR = 1 all parameters are updated jointly.
updateInterval	determining the interval for the pCR (crossover probabilities) update
gamma	Kurtosis parameter Bayesian Inference Scheme. Default is 0.
eps	Ergodicity term. Default to 0.
e	Ergodicity term. Default to 5e-2.
pCRupdate	Update of crossover probabilities, default is TRUE
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thin thinning parameter. Determines the interval in which values are recorded.
adaptation	Number or percentage of samples that are used for the adaptation in DREAM (see Details)
DEpairs	Number of pairs used to generate proposal

ZupdateFrequency	frequency to update Z matrix
pSnooker	probability of snooker update
Z	starting matrix for Z
startValue	either a matrix containing the start values (see details), an integer to define the number of chains that are run, a function to sample the start values or NULL, in which case the values are sampled from the prior.
consoleUpdates	Intervall in which the sampling progress is printed to the console
message	logical determines whether the sampler's progress should be printed

Details

Insted of a `bayesianSetup`, the function can take the output of a previous run to restart the sampler from the last iteration. Due to the sampler's internal structure you can only use the output of `DREAMzs`. If you provide a matrix with start values the number of rows detemines the number of chains that are run. The number of coloumns must be equivalent to the number of parameters in your `bayesianSetup`.

There are several small differences in the algorithm presented here compared to the original paper by Vrugt et al. (2009). Mainly the algorithm implemented here does not have an automatic stopping criterion. Hence, it will always run the number of iterations specified by the user. Also, convergence is not monitored and left to the user. This can easily be done with `coda::gelman.diag(chain)`. Further the proposed delayed rejectio step in Vrugt et al. (2009) is not implemented here.

During the adaptation phase DREAM is running two mechanisms to enhance the sampler's efficiency. First the distribution of crossover values is tuned to favor large jumps in the parameter space. The crossover probabilities determine how many parameters are updated simultaneously. Second outlier chains are replanced as they can largely deteriorate the sampler's performance. However, these steps destroy the detailed balance of the chain. Consequently these parts of the chain should be discarded when summarizing posterior moments. This can be done automatically during the sampling process (i.e. `burnin > adaptation`) or subsequently by the user. We chose to distinguish between the burnin and adaptation phase to allow the user more flexibility in the sampler's settings.

Value

`mcmc.object` containing the following elements: `chains`, `X`, `pCR`, `Z`

Author(s)

Stefan Paul

References

Vrugt, Jasper A., et al. "Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling." *International Journal of Nonlinear Sciences and Numerical Simulation* 10.3 (2009): 273-290.

ter Braak C. J. F., and Vrugt J. A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. *Statistics and Computing* <http://dx.doi.org/10.1007/s11222-008-9104-9>

See Also[DREAM](#)

gelmanDiagnostics *Runs Gelman Diagnostics over an BayesianOutput*

Description

Runs Gelman Diagnostics over an BayesianOutput

Usage

```
gelmanDiagnostics(sampler, thin = "auto", plot = F, ...)
```

Arguments

sampler	an object of class <code>mcmcSampler</code> or <code>mcmcSamplerList</code>
thin	parameter determining the thinning intervall. Either an integer or "auto" (default) for automatic thinning.
plot	should a Gelman plot be generated
...	further arguments passed to getSample

Details

The function calls the coda package to calculate Gelman diagnostics and plots

The original idea is that this function is applied to the outcome of several independent MCMC runs. Technically and practically, it can also be applied to a single MCMC run that has several internal chains, such as DE, DEzs, DREAM, DREAMzs or T-Walk. As argued in ter Braak et al. (2008), the internal chains should be independent after burn-in. While this is likely correct, it also means that they are not completely independent before, and we observed this behavior in the use of the algorithms (i.e. that internal DEzs chains are more similar to each other than the chains of independent DEzs algorithms). A concern is that this non-independence could lead to a failure to detect that the sampler hasn't converged yet. We would therefore recommend to run several DEzs and check convergence with those, instead of running only one.

ter Braak, Cajo JF, and Jasper A. Vrugt. "Differential evolution Markov chain with snooker updater and fewer chains." *Statistics and Computing* 18.4 (2008): 435-446.

Author(s)

Florian Hartig

`generateParallelExecuter`*Factory to generate a parallel executer of an existing function*

Description

Factory to generate a parallel executer of an existing function

Usage

```
generateParallelExecuter(fun, parallel = F, parallelOptions = list(variables  
  = "all", packages = "all", dlls = NULL))
```

Arguments

<code>fun</code>	function to be changed to parallel execution
<code>parallel</code>	should a parallel R cluster be used or not. If set to T, cores will be detected automatically and n-1 of the available n cores of the machine will be used. Alternatively, you can set the number of cores used by hand
<code>parallelOptions</code>	list containing three lists. First "packages" determines the R packages necessary to run the likelihood function. Second "variables" the objects in the global environment needed to run the likelihood function and third "dlls" the DLLs needed to run the likelihood function (see Details).

Details

For parallelization, option T means that an automatic parallelization via R is attempted, or "external", in which case it is assumed that the likelihood is already parallelized. In this case it needs to accept a matrix with parameters as columns. Further you can specify the packages, objects and DLLs that are exported to the cluster. By default a copy of your workspace is exported. However, depending on your workspace this can be very inefficient.

Alternatively you can specify the environments and packages in the likelihood function (e.g. BayesianTools::VSEM() instead of VSEM()).

Note

Can also be used to make functions compatible with library sensitivity

Author(s)

Florian Hartig

Examples

```
testDensityMultiNormal <- generateTestDensityMultiNormal()

parDen <- generateParallelExecuter(testDensityMultiNormal)$parallelFun
x = matrix(runif(9,0,1), nrow = 3)
parDen(x)
```

```
generateTestDensityMultiNormal
      Multivariate normal likelihood
```

Description

Generates a 3 dimensional multivariate normal likelihood function.

Usage

```
generateTestDensityMultiNormal(mean = c(0, 0, 0),
  sigma = "strongcorrelation", sample = F, n = 1, throwErrors = -1)
```

Arguments

mean	vector with the three mean values of the distribution
sigma	either a correlation matrix, or "strongcorrelation", or "no correlation"
sample	should the function create samples
n	number of samples to create
throwErrors	parameter for test purpose. Between 0 and 1 for proportion of errors

Details

3-d multivariate normal density function with mean 2,4,0 and either strong correlation (default), or no correlation.

Author(s)

Florian Hartig

See Also

[testDensityBanana](#)
[testLinearModel](#)

Examples

```
# sampling from the test function
x = generateTestDensityMultiNormal(sample = TRUE, n = 1000)(1000)
correlationPlot(x)
marginalPlot(x)

# generating the the density
density = generateTestDensityMultiNormal(sample = FALSE)
density(x[1,])
```

getCredibleIntervals *Calculate confidence region from an MCMC or similar sample*

Description

Calculate confidence region from an MCMC or similar sample

Usage

```
getCredibleIntervals(sampleMatrix, quantiles = c(0.025, 0.975))
```

Arguments

sampleMatrix	matrix of outcomes. Could be parameters or predictions
quantiles	quantiles to be calculated

Author(s)

Florian Hartig

See Also

[getPredictiveDistribution](#)
[getPredictiveIntervals](#)

getDharmaResiduals *Creates a DHARMA object*

Description

Creates a DHARMA object

Usage

```
getDharmaResiduals(model, parMatrix, numSamples, observed, error, plot = TRUE)
```

Arguments

model	function that calculates model predictions for a given parameter vector
parMatrix	a parameter matrix from which the simulations will be generated
numSamples	the number of samples
observed	a vector of observed values
error	function with signature <code>f(mean, par)</code> that generates error expectations from mean model predictions. <code>Par</code> is a vector from the matrix with the parameter samples (full length). <code>f</code> needs to know which of these parameters are parameters of the error function
plot	logical, determining whether the simulated residuals should be plotted

Author(s)

Tankred Ott

getMetropolisDefaultSettings
Returns Metropolis default settings

Description

Returns Metropolis default settings

Usage

```
getMetropolisDefaultSettings()
```

Author(s)

Tankred Ott

getPanels *Calculates the panel combination for par(mfrow =)*

Description

Calculates the panel combination for par(mfrow =)

Usage

getPanels(x)

Arguments

x the desired number of panels

Author(s)

Florian Hartig

getPossibleSamplerTypes
Returns possible sampler types

Description

Returns possible sampler types

Usage

getPossibleSamplerTypes()

Author(s)

Florian Hartig

getPredictiveDistribution

Calculates predictive distribution based on the parameters

Description

Calculates predictive distribution based on the parameters

Usage

```
getPredictiveDistribution(parMatrix, model, numSamples = 1000)
```

Arguments

parMatrix	matrix of parameter values
model	model / function to calculate predictions. Outcome should be a vector
numSamples	number of samples to be drawn

Details

If numSamples is greater than the number of rows in parMatrix, or NULL, or FALSE, or less than 1 all samples in parMatrix will be used.

Author(s)

Florian Hartig

See Also

[getPredictiveIntervals](#)
[getCredibleIntervals](#)

getPredictiveIntervals

Calculates Bayesian credible (confidence) and predictive intervals based on parameter sample

Description

Calculates Bayesian credible (confidence) and predictive intervals based on parameter sample

Usage

```
getPredictiveIntervals(parMatrix, model, numSamples = 1000,  
  quantiles = c(0.025, 0.975), error = NULL)
```


Arguments

parMatrix	matrix of parameter values
model	model / function to calculate predictions. Outcome should be a vector
numSamples	number of samples to be drawn
quantiles	quantiles to calculate
error	function with signature $f(\text{mean}, \text{par})$ that generates error expectations from mean model predictions. Par is a vector from the matrix with the parameter samples (full length). f needs to know which of these parameters are parameters of the error function. If supplied, will calculate also predictive intervals additional to credible intervals

Details

If numSamples is greater than the number of rows in parMatrix, or NULL, or FALSE, or less than 1 all samples in parMatrix will be used.

Author(s)

Florian Hartig

See Also

[getPredictiveDistribution](#)
[getCredibleIntervals](#)

getSample	<i>Extracts the sample from a bayesianOutput</i>
-----------	--

Description

Extracts the sample from a bayesianOutput

Usage

```
getSample(sampler, parametersOnly = T, coda = F, start = 1, end = NULL,
  thin = 1, numSamples = NULL, whichParameters = NULL,
  includesProbabilities = F, reportDiagnostics = FALSE, ...)
```

Arguments

sampler	an object of class mcmcSampler, mcmcSamplerList, smcSampler, smcSamplerList, mcmc, mcmc.list, double, numeric
parametersOnly	if F, likelihood, posterior and prior values are also provided in the output

coda	works only for mcmc classes - provides output as a coda object. Note: if mcmcSamplerList contains mcmc samplers such as DE that have several chains, the internal chains will be collapsed. This may not be the desired behavior for all applications.
start	for mcmc samplers start value in the chain. For SMC samplers, start particle
end	for mcmc samplers end value in the chain. For SMC samplers, end particle
thin	thinning parameter. Either an integer determining the thinning intervall (default is 1) or "auto" for automatic thinning.
numSamples	sample size (only used if thin = 1). If you want to use numSamples set thin to 1.
whichParameters	possibility to select parameters by index
includesProbabilities	applies only to getSample.Matrix. logical, determining whether probabilities should be included in the result.
reportDiagnostics	logical, determines whether settings should be included in the output
...	further arguments

Details

If thin is greater than the total number of samples in the sampler object the first and the last element (of each chain if a sampler with multiples chains is used) are sampled. If numSamples is greater than the total number of samples all samples are selected. In both cases a warning is displayed.

If thin and numSamples is passed, the function will use the thin argument if it is valid and greater than 1, else numSamples will be used.

Author(s)

Florian Hartig

Examples

```
ll = function(x) sum(dnorm(x, log = TRUE))

setup = createBayesianSetup(ll, lower = c(-10,-10), upper = c(10,10))

settings = list(nrChains = 2, iterations = 1000)
out <- runMCMC(bayesianSetup = setup, sampler = "DEzs", settings = settings)

# population MCMCs divide the interations by the number of internal chains,
# so the end of the 3 chains is 1000/3 = 334
sample <- getSample(out, start = 100, end = 334, thin = 10)

# sampling with number of samples instead of thinning and
# returning a coda object
sample <- getSample(out, start = 100, numSamples = 60, coda = TRUE)
plot(sample)
```

```
# MCMC with a single chain:
settings_2 <- list(nrChains = 1, iterations = 1000)
out_2 <- runMCMC(setup, sampler = "Metropolis", settings = settings_2)
sample_2 <- getSample(out_2, numSamples = 100)
```

getVolume	<i>Calculate posterior volume</i>
-----------	-----------------------------------

Description

Calculate posterior volume

Usage

```
getVolume(sampler, prior = F, method = "MVN", ...)
```

Arguments

sampler	an object of superclass <code>bayesianOutput</code> or any other class that has the <code>getSample</code> function implemented (e.g. <code>Matrix</code>)
prior	should also prior volume be calculated
method	method for volume estimation. Currently, the only option is "MVN"
...	additional parameters to pass on to the getSample

Details

The idea of this function is to provide an estimate of the "posterior volume", i.e. how "broad" the posterior is. One potential application is to the overall reduction of parametric uncertainty between different data types, or between prior and posterior.

Implemented methods for volume estimation:

Option "MVN" - in this option, the volume is calculated as the determinant of the covariance matrix of the prior / posterior sample.

Author(s)

Florian Hartig

Examples

```
bayesianSetup = createBayesianSetup(
  likelihood = generateTestDensityMultiNormal(sigma = "no correlation"),
  lower = rep(-10, 3), upper = rep(10, 3))

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "Metropolis",
  settings = list(iterations = 2000, message = FALSE))
```

```

getVolume(out, prior = TRUE)

bayesianSetup = createBayesianSetup(
  likelihood = generateTestDensityMultiNormal(sigma = "strongcorrelation"),
  lower = rep(-10, 3), upper = rep(10, 3))

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "Metropolis",
  settings = list(iterations = 2000, message = FALSE))

getVolume(out, prior = TRUE)

```

GOF *Standard GOF metrics Startvalues for sampling with nrChains > 1 : if you want to provide different start values for the different chains, provide a list*

Description

Standard GOF metrics Startvalues for sampling with nrChains > 1 : if you want to provide different start values for the different chains, provide a list

Usage

```
GOF(observed, predicted, plot = F)
```

Arguments

observed	observed values
predicted	predicted values
plot	should a plot be created

Details

The function considers predicted ~ observed and calculates

1) rmse = root mean squared error 2) mae = mean absolute error 3) a linear regression with slope, intercept and coefficient of determination R2

For the linear regression, the x axis is centered, meaning that the intercept is the difference between observed / predicted for the MEAN predicted value. This setting avoids a correlation between slope and intercept (that the intercept is != 0 as soon as the slope is !=0)

Value

A list with the following entries: rmse = root mean squared error, mae = mean absolute error, slope = slope of regression, offset = intercept of regression, R2 = R2 of regression

Author(s)

Florian Hartig

Examples

```
x = runif(500,-1,1)
y = 0.2 + 0.9 *x + rnorm(500, sd = 0.5)

summary(lm(y ~ x))

GOF(x,y)

GOF(x,y, plot = TRUE)
```

histMarginal	<i>histogram for marginalPlot</i>
--------------	-----------------------------------

Description

Internal function to plot histograms for the marginalPlot function

Usage

```
histMarginal(x, at = 0, .range = NULL, breaks = 15, col = c("#FF5000C0",
"#4682B4A0"), border = c("black", "black"), main = "", dens = FALSE,
densCol = c("black", "black"), densLwd = c(2, 2), densLty = c(2, 2),
add = TRUE, res = 500)
```

Arguments

x	list of vectors. Each vector will be plotted as a separate histogram.
at	y position of the histograms
.range	maximum height of the histogram. If NULL, will be determined automatically
breaks	integer, determining the number of breaks
col	vector of colors determining histogram colors
border	vector of colors determining histogram border colors
main	Character, determining the title of the plot
dens	Logical, determining whether a density plot should be plotted additionally to the histogram
densCol	vector of colors for the density plots
densLwd	line width of the density plot
densLty	line type of the density plot
add	Logical, determining whether the histogram should be added to an existing plot window.
res	resolution of the density overlay

Author(s)

Tankred Ott

likelihoodAR1 *AR1 type likelihood function*

Description

AR1 type likelihood function

Usage

likelihoodAR1(predicted, observed, sd, a)

Arguments

predicted	vector of predicted values
observed	vector of observed values
sd	standard deviation of the iid normal likelihood
a	temporal correlation in the AR1 model

Note

The AR1 model considers the process:

$$y(t) = a y(t-1) + E$$

e = i.i.d. N(0, sd)

|a| < 1

At the moment, no NAs are allowed in the time series.

Author(s)

Florian Hartig

likelihoodIidNormal *Normal / Gaussian Likelihood function*

Description

Normal / Gaussian Likelihood function

Usage

likelihoodIidNormal(predicted, observed, sd)

Arguments

predicted	vector of predicted values
observed	vector of observed values
sd	standard deviation of the i.i.d. normal likelihood

Author(s)

Florian Hartig

logSumExp	<i>Funktion to compute $\log(\text{sum}(\exp(x)))$</i>
-----------	---

Description

Funktion to compute $\log(\text{sum}(\exp(x)))$

Usage

```
logSumExp(x, mean = F)
```

Arguments

x	values
mean	logical, determines whether the mean should be used

Author(s)

Florian Hartig

M	<i>The Metropolis Algorithm</i>
---	---------------------------------

Description

The Metropolis Algorithm (Metropolis et al. 1953)

Usage

```
M(startValue = NULL, iterations = 10000, nBI = 0, parmin = NULL,  
  parmax = NULL, f = 1, FUN, consoleUpdates = 1000)
```

Arguments

startValue	vector with the start values for the algorithm. Can be NULL if FUN is of class BayesianSetup. In this case startValues are sampled from the prior.
iterations	iterations to run
nBI	number of burnin
parmin	minimum values for the parameter vector or NULL if FUN is of class BayesianSetup
parmax	maximum values for the parameter vector or NULL if FUN is of class BayesianSetup
f	scaling factor
FUN	function to be sampled from or object of class bayesianSetup
consoleUpdates	interger, determines the frequency with which sampler progress is printed to the console

Author(s)

Francesco Minunno

References

Metropolis, Nicholas, et al. "Equation of state calculations by fast computing machines." The journal of chemical physics 21.6 (1953): 1087-1092.

MAP

calculates the Maximum APosteriori value (MAP)

Description

calculates the Maximum APosteriori value (MAP)

Usage

```
MAP(bayesianOutput, ...)
```

Arguments

bayesianOutput	an object of class BayesianOutput (mcmcSampler, smcSampler, or mcmcList)
...	optional values to be passed on the the getSample function

Details

Currently, this function simply returns the parameter combination with the highest posterior in the chain. A more refined option would be to take the MCMC sample and do additional calculations, e.g. use an optimizer, a kernel density estimator, or some other tool to search / interpolate around the best value in the chain

Author(s)

Florian Hartig

See Also[WAIC, DIC, marginalLikelihood](#)

marginalLikelihood *Calculated the marginal likelihood from a set of MCMC samples*

Description

Calculated the marginal likelihood from a set of MCMC samples

Usage

```
marginalLikelihood(sampler, numSamples = 1000, method = "Chib", ...)
```

Arguments

sampler	an object that implements the <code>getSample</code> function, i.e. a <code>mcmc / smc Sampler</code> (list)
numSamples	number of samples to use. How this works, and if it requires recalculating the likelihood, depends on the method
method	method to choose. Currently available are "Chib" (default), the harmonic mean "HM", sampling from the prior "prior", and bridge sampling "Bridge". See details
...	further arguments passed to getSample

Details

The function currently implements four ways to calculate the marginal likelihood.

The recommended way is the method "Chib" (Chib and Jeliazkov, 2001). which is based on MCMC samples, but performs additional calculations. Despite being the current recommendation, note there are some numeric issues with this algorithm that may limit reliability for larger dimensions.

The harmonic mean approximation, is implemented only for comparison. Note that the method is numerically unreliable and usually should not be used.

The third method is simply sampling from the prior. While in principle unbiased, it will only converge for a large number of samples, and is therefore numerically inefficient.

The Bridge method uses bridge sampling as implemented in the R package "bridgesampling".

Author(s)

Florian Hartig

References

Chib, Siddhartha, and Ivan Jeliazkov. "Marginal likelihood from the Metropolis-Hastings output." *Journal of the American Statistical Association* 96.453 (2001): 270-281.

See Also

[WAIC, DIC, MAP](#)

Examples

```
# Harmonic mean works OK for a low-dim case with

likelihood <- function(x) sum(dnorm(x, log = TRUE))
prior = createUniformPrior(lower = rep(-1,2), upper = rep(1,2))
bayesianSetup <- createBayesianSetup(likelihood = likelihood, prior = prior)
out = runMCMC(bayesianSetup = bayesianSetup, settings = list(iterations = 5000))

plot(out)

marginalLikelihood(out, numSamples = 500)[[1]]
marginalLikelihood(out, method = "HM", numSamples = 500)[[1]]
marginalLikelihood(out, method = "Prior", numSamples = 500)[[1]]
marginalLikelihood(out, method = "Bridge", numSamples = 500)[[1]]

# True marginal likelihood (brute force approximation)

marginalLikelihood(out, method = "Prior", numSamples = 10000)[[1]]

# Harmonic mean goes totally wrong for higher dimensions - wide prior.
# Same goes for standard bridge sampling.
# Could also be a problem of numeric stability of the implementation

likelihood <- function(x) sum(dnorm(x, log = TRUE))
prior = createUniformPrior(lower = rep(-10,3), upper = rep(10,3))
bayesianSetup <- createBayesianSetup(likelihood = likelihood, prior = prior)
out = runMCMC(bayesianSetup = bayesianSetup, settings = list(iterations = 5000))

plot(out)

marginalLikelihood(out, numSamples = 500)[[1]]
marginalLikelihood(out, method = "HM", numSamples = 500)[[1]]
marginalLikelihood(out, method = "Prior", numSamples = 500)[[1]]
marginalLikelihood(out, method = "Bridge", numSamples = 500)[[1]]

# True marginal likelihood (brute force approximation)

marginalLikelihood(out, method = "Prior", numSamples = 10000)[[1]]
```

marginalPlot

Plot MCMC marginals

Description

Plot MCMC marginals

Usage

```
marginalPlot(mat, thin = "auto", scale = NULL, best = NULL,
             histogram = TRUE, plotPrior = TRUE, priorTop = FALSE,
             nDrawsPrior = 1000, breaks = 15, res = 500, singlePanel = FALSE,
             dens = TRUE, col = c("#FF5000D0", "#4682B4A0"), lwd = par("lwd"), ...)
```

Arguments

mat	object of class "bayesianOutput" or a matrix or data frame of variables
thin	thinning of the matrix to make things faster. Default is to thin to 5000
scale	should the results be scaled. Value can be either NULL (no scaling), T, or a matrix with upper / lower bounds as columns. If set to T, attempts to retrieve the scaling from the input object mat (requires that this is of class BayesianOutput)
best	if provided, will draw points at the given values (to display true / default parameter values). Value can be either NULL (no drawing), a vector with values, or T, in which case the function will attempt to retrieve the values from a BayesianOutput
histogram	Logical, determining whether a violin plot or a histogram should be plotted
plotPrior	Logical, determining whether the prior should be plotted in addition to the posteriors. Only applicable if mat is an object of class "bayesianOutput"
priorTop	Logical, determining whether the prior should be plotted top (TRUE) or bottom (FALSE)
nDrawsPrior	Integer, number of draws from the prior, when plotPrior is active
breaks	Integer, number of histogram breaks if histogram is set to TRUE
res	resolution parameter for violinPlot, determining how many discrete points should be used for the density kernel.
singlePanel	Logical, determining whether all histograms/violins should be plotted in a single plot panel or in separate panels.
dens	Logical, determining whether a density overlay should be plotted when 'histogram' is TRUE
col	vector of colors for posterior and prior
lwd	line width of the violin plots
...	additional parameters to pass on to the getSample

Author(s)

Florian Hartig, Tankred Ott

References

[tracePlot](#)
[correlationPlot](#)

Examples

```
dat = generateTestDensityMultiNormal(n = 100000, sample = TRUE)
marginalPlot(dat(10000))
```

mcmcMultipleChains *Run multiple chains*

Description

Run multiple chains

Usage

```
mcmcMultipleChains(bayesianSetup, settings, sampler)
```

Arguments

bayesianSetup	Object of class "BayesianSetup"
settings	list with settings for sampler
sampler	character, either "Metropolis" or "DE"

Value

list containing the single runs (`$sampler`) and the chains in a `coda::mcmc.list` (`$mcmc.list`)

Metropolis	<i>Creates a Metropolis-type MCMC with options for covariance adaptatin, delayed rejection, Metropolis-within-Gibbs, and tempering</i>
------------	--

Description

Creates a Metropolis-type MCMC with options for covariance adaptatin, delayed rejection, Metropolis-within-Gibbs, and tempering

Usage

```
Metropolis(bayesianSetup, settings = list(startValue = NULL, optimize = T,
  proposalGenerator = NULL, consoleUpdates = 100, burnin = 0, thin = 1, parallel
  = NULL, adapt = T, adaptationInterval = 500, adaptationNotBefore = 3000,
  DRlevels = 1, proposalScaling = NULL, adaptationDepth = NULL,
  temperingFunction = NULL, gibbsProbabilities = NULL, message = TRUE))
```

Arguments

bayesianSetup	either an object of class <code>bayesianSetup</code> created by createBayesianSetup (recommended), or a log target function
settings	a list of settings - possible options follow below
startValue	startValue for the MCMC and optimization (if <code>optimize = T</code>). If not provided, the sampler will attempt to obtain the startValue from the <code>bayesianSetup</code>
optimize	logical, determines whether an optimization for start values and proposal function should be run before starting the sampling
proposalGenerator	optional proposalgenerator object (see createProposalGenerator)
proposalScaling	additional scaling parameter for the proposals that controls the different scales of the proposals after delayed rejection (typical, after a rejection, one would want to try a smaller scale). Needs to be as long as <code>DRlevels</code> . Defaults to $0.5^{-(\text{mcmcSampler}\$settings\$DRlevels - 1)}$
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thinning parameter. Determines the interval in which values are recorded.
consoleUpdates	integer, determines the frequency with which sampler progress is printed to the console
adapt	logical, determines wheter an adaptive algorithm should be implemented. Default is <code>TRUE</code> .
adaptationInterval	integer, determines the interval of the adaption if <code>adapt = TRUE</code> .
adaptationNotBefore	integer, determines the start value for the adaption if <code>adapt = TRUE</code> .

DRlevels	integer, determines the number of levels for a delayed rejection sampler. Default is 1, which means no delayed rejection is used.
temperingFunction	function to implement simulated tempering in the algorithm. The function describes how the acceptance rate will be influenced in the course of the iterations.
gibbsProbabilities	vector that defines the relative probabilities of the number of parameters to be changes simultaneously.
message	logical determines whether the sampler's progress should be printed

Details

The 'Metropolis' function is the main function for all Metropolis based samplers in this package. To call the derivatives from the basic Metropolis-Hastings MCMC, you can either use the corresponding function (e.g. [AM](#) for an adaptive Metropolis sampler) or use the parameters to adapt the basic Metropolis-Hastings. The advantage of the latter case is that you can easily combine different properties (e.g. adaptive sampling and delayed rejection sampling) without changing the function.

Author(s)

Florian Hartig

References

- Haario, H., E. Saksman, and J. Tamminen (2001). An adaptive metropolis algorithm. *Bernoulli* , 223-242.
- Haario, Heikki, et al. "DRAM: efficient adaptive MCMC." *Statistics and Computing* 16.4 (2006): 339-354.
- Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika* 57 (1), 97-109.
- Green, Peter J., and Antonietta Mira. "Delayed rejection in reversible jump Metropolis-Hastings." *Biometrika* (2001): 1035-1053.
- Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics* 21 (6), 1087 - 1092.

Examples

```
# Running the metropolis via the runMCMC with a proposal covariance generated from the prior
# (can be useful for complicated priors)

ll = function(x) sum(dnorm(x, log = TRUE))
setup = createBayesianSetup(ll, lower = c(-10,-10), upper = c(10,10))

samples = setup$prior$sampler(1000)

generator = createProposalGenerator(diag(1, setup$numPars))
```

```

generator = updateProposalGenerator(generator, samples, manualScaleAdjustment = 1, message = TRUE)

settings = list(proposalGenerator = generator, optimize = FALSE, iterations = 500)

out = runMCMC(bayesianSetup = setup, sampler = "Metropolis", settings = settings)

```

metropolisRatio *Function to calculate the metropolis ratio*

Description

Function to calculate the metropolis ratio

Usage

```
metropolisRatio(LP2, LP1, tempering = 1)
```

Arguments

LP2	log posterior old position
LP1	log posterior of proposal
tempering	value for tempering

Author(s)

Florian Hartig

plotHist *plot histogram*

Description

A simple custom histogram plotting function

Usage

```
plotHist(x, at = NULL, col = "orangered", border = "black")
```

Arguments

x	matrix (e.g. constructed with createBreakMat) or dataframe containing the lower bounds, the upper bounds, and the frequencies of the breaks in the columns and the individual breaks in the rows.
at	y position of the histogram. If NULL a new plot will be generated.
col	color of the histogram
border	border color

Author(s)

Tankred Ott

plotSensitivity	<i>Performs a one-factor-at-a-time sensitivity analysis for the posterior of a given bayesianSetup within the prior range.</i>
-----------------	--

Description

Performs a one-factor-at-a-time sensitivity analysis for the posterior of a given bayesianSetup within the prior range.

Usage

```
plotSensitivity(bayesianSetup, selection = NULL)
```

Arguments

bayesianSetup	An object of class BayesianSetup
selection	indices of selected parameters

Note

This function can also be used for sensitivity analysis of an arbitrary output - just create a BayesianSetup with this output.

Author(s)

Florian Hartig

plotTimeSeries	<i>Plots a time series, with the option to include confidence and prediction band</i>
----------------	---

Description

Plots a time series, with the option to include confidence and prediction band

Usage

```
plotTimeSeries(observed = NULL, predicted = NULL, x = NULL,
  confidenceBand = NULL, predictionBand = NULL, xlab = "Time",
  ylab = "Observed / predicted values", ...)
```


Arguments

observed	observed values
predicted	predicted values
x	optional values for x axis (time)
confidenceBand	matrix with confidenceBand
predictionBand	matrix with predictionBand
xlab	a title for the x axis
ylab	a title for the y axis
...	further arguments passed to plot

Author(s)

Florian Hartig

See Also

[plotTimeSeriesResults](#)
[marginalPlot](#)
[tracePlot](#)
[correlationPlot](#)

Examples

```
# Create time series
ts <- VSEMcreatePAR(1:100)

# create fake "predictions"
pred <- ts + rnorm(length(ts), mean = 0, sd = 2)

# plot time series
par(mfrow=c(1,2))

plotTimeSeries(observed = ts, main="Observed")
plotTimeSeries(observed = ts, predicted = pred, main = "Observed and predicted")

par(mfrow=c(1,1))
```

plotTimeSeriesResiduals

Plots residuals of a time series

Description

Plots residuals of a time series

Usage

```
plotTimeSeriesResiduals(residuals, x = NULL, main = "residuals")
```

Arguments

residuals	x
x	optional values for x axis (time)
main	title of the plot

Author(s)

Florian Hartig

plotTimeSeriesResults *Creates a time series plot typical for an MCMC / SMC fit*

Description

Creates a time series plot typical for an MCMC / SMC fit

Usage

```
plotTimeSeriesResults(sampler, model, observed, error = NULL,
  plotResiduals = TRUE, start = 1, prior = FALSE, ...)
```

Arguments

sampler	Either a) a matrix b) an MCMC object (list or not), or c) an SMC object
model	function that calculates model predictions for a given parameter vector
observed	observed values
error	function with signature $f(\text{mean}, \text{par})$ that generates error expectations from mean model predictions. Par is a vector from the matrix with the parameter samples (full length). f needs to know which of these parameters are parameters of the error function
plotResiduals	logical determining whether residuals should be plotted
start	numeric start value for the plot (see getSample)
prior	if a prior sampler is implemented, setting this parameter to TRUE will draw model parameters from the prior instead of the posterior distribution
...	further arguments passed to plot

Author(s)

Florian Hartig

rescale	<i>Rescale</i>
---------	----------------

Description

Rescales values in the interval "from" (lower, upper) to the new interval "to" (lower, upper).

Usage

```
rescale(x, from, to)
```

Arguments

x	Vector of values
from	vector, interval of which x are elements. from[1] must be the lower, from[2] the upper bound.
to	vector, interval to which the elements should be scaled. to[1] must be the lower, to[2] the upper bound.

Author(s)

Tankred Ott

runMCMC	<i>Main wrapper function to start MCMCs, particle MCMCs and SMCs</i>
---------	--

Description

Main wrapper function to start MCMCs, particle MCMCs and SMCs

Usage

```
runMCMC(bayesianSetup, sampler = "DEzs", settings = NULL)
```

Arguments

bayesianSetup	either one of a) an object of class BayesianSetup with prior and likelihood function (recommended, see createBayesianSetup), b) a log posterior or other target function, or c) an object of class BayesianOutput created by runMCMC. The latter allows to continue a previous MCMC run. See details for further details.
sampler	sampling algorithm to be run. Default is DEzs. Options are "Metropolis", "AM", "DR", "DRAM", "DE", "DEzs", "DREAM", "DREAMzs", "SMC". For details see the help of the individual functions.
settings	list with settings for each sampler (see help of sampler for details). If a setting is not provided, defaults (see applySettingsDefault) will be used.

Details

The runMCMC function can be started with either one of a) an object of class BayesianSetup with prior and likelihood function (recommended, see [createBayesianSetup](#)), b) a log posterior or other target function, or c) an object of class BayesianOutput created by runMCMC. The latter allows to continue a previous MCMC run. If a bayesianSetup is provided, check if appropriate parallization options are used - many samplers can make use of parallelization if this option is activated when the class is created.

For details about the different MCMC samplers, make sure you have read the Vignette (run vignette("BayesianTools", package="BayesianTools"). Also, see [Metropolis](#) for Metropolis based samplers, [DE](#) and [DEzs](#) for standard differential evolution samplers, [DREAM](#) and [DREAMzs](#) for DREAM sampler, [Twalk](#) for the Twalk sampler, and [smcSampler](#) for rejection and Sequential Monte Carlo sampling.

The samplers "AM", "DR", and "DRAM" are special cases of the "Metropolis" sampler and are shortcuts for predefined settings ("AM": adapt=TRUE; "DR": DRlevels=2; "DRAM": adapt=True, DRlevels=2).

The settings list allows to change the settings for the MCMC samplers and some other options. For the MCMC sampler settings, see their help files. Global options that apply for all MCMC samplers are: iterations (number of MCMC iterations), and nrChains (number of chains to run). Note that running several chains is not done in parallel, so if time is an issue it will be better to run the MCMCs individually and then combine them via [createMcmcSamplerList](#) into one joint object.

Startvalues: all samplers allow to provide explicit startvalues. If startvalues are not provided, they are sampled from the prior. Usually, this is a good choice, so don't feel compelled to provide startvalues.

Note that DE and DREAM variants as well as SMC and T-walk require a population to start, which should be provided as a matrix. Default (NULL) sets the population size for DE to 3 x dimensions of parameters, for DREAM to 2 x dimensions of parameters and for DEzs and DREAMzs to three, sampled from the prior. Note also that the zs variants of DE and DREAM require two populations, the current population and the z matrix (a kind of memory) - if you want to set both, provide a list with startvalue\$X and startvalue\$Z.

Startvalues for sampling with nrChains > 1 : if you want to provide different start values for the different chains, provide them as a list

Value

The function returns an object of class mcmcSampler (if one chain is run) or mcmcSamplerList. Both have the superclass bayesianOutput. It is possible to extract the samples as a coda object or matrix with [getSample](#). It is also possible to summarize the posterior as a new prior via [createPriorDensity](#).

Author(s)

Florian Hartig

See Also

[createBayesianSetup](#)

Examples

```

## Generate a test likelihood function.
ll <- generateTestDensityMultiNormal(sigma = "no correlation")

## Create a BayesianSetup object from the likelihood
## is the recommended way of using the runMCMC() function.
bayesianSetup <- createBayesianSetup(likelihood = ll, lower = rep(-10, 3), upper = rep(10, 3))

## Finally we can run the sampler and have a look
settings = list(iterations = 1000, adapt = FALSE)
out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "Metropolis", settings = settings)

## out is of class bayesianOutput. There are various standard functions
# implemented for this output

plot(out)
correlationPlot(out)
marginalPlot(out)
summary(out)

## additionally, you can return the sample as a coda object, and make use of the coda functions
# for plotting and analysis

codaObject = getSample(out, start = 500, coda = TRUE)

```

sampleEquallySpaced *Gets n equally spaced samples (rows) from a matrix or vector*

Description

Gets n equally spaced samples (rows) from a matrix or vector

Usage

```
sampleEquallySpaced(x, numSamples)
```

Arguments

x	matrix or vector
numSamples	number of samples (rows) to be drawn

Details

Gets n equally spaced samples (rows) from a matrix and returns a new matrix (or vector) containing those samples

Author(s)

Tankred Ott

sampleMetropolis	<i>gets samples while adopting the MCMC proposal generator</i>
------------------	--

Description

Function to sample with combinations of the basic Metropolis-Hastings MCMC algorithm (Metropolis et al., 1953), a variation of the adaptive Metropolis MCMC (Haario et al., 2001), the delayed rejection algorithm (Tierney & Mira, 1999), and the delayed rejection adaptive Metropolis algorithm (DRAM, Haario et al), and the Metropolis within Gibbs

Usage

```
sampleMetropolis(mcmcSampler, iterations)
```

Arguments

mcmcSampler	an mcmcSampler
iterations	iterations

Author(s)

Florian Hartig

setupStartProposal	<i>Help function to find starvalues and proposalGenerator settings</i>
--------------------	--

Description

Help function to find starvalues and proposalGenerator settings

Usage

```
setupStartProposal(proposalGenerator = NULL, bayesianSetup, settings)
```

Arguments

proposalGenerator	proposal generator
bayesianSetup	either an object of class bayesianSetup created by createBayesianSetup (recommended), or a log target function
settings	list with settings

Author(s)

Florian Hartig

smcSampler	<i>SMC sampler</i>
------------	--------------------

Description

Sequential Monte Carlo Sampler

Usage

```
smcSampler(bayesianSetup, initialParticles = 1000, iterations = 10,
           resampling = T, resamplingSteps = 2, proposal = NULL, adaptive = T,
           proposalScale = 0.5)
```

Arguments

bayesianSetup	either an object of class <code>bayesianSetup</code> created by <code>createBayesianSetup</code> (recommended), or a log target function
initialParticles	initial particles - either a draw from the prior, provided as a matrix with the single parameters as columns and each row being one particle (parameter vector), or a numeric value with the number of desired particles. In this case, the sampling option must be provided in the prior of the <code>BayesianSetup</code> .
iterations	number of iterations
resampling	if new particles should be created at each iteration
resamplingSteps	how many resampling (MCMC) steps between the iterations
proposal	optional proposal class
adaptive	should the covariance of the proposal be adapted during sampling
proposalScale	scaling factor for the proposal generation. Can be adapted if there is too much / too little rejection

Details

The sampler can be used for rejection sampling as well as for sequential Monte Carlo. For the former case set the iterations to one.

Note

The SMC currently assumes that the initial particle is sampled from the prior. If a better initial estimate of the posterior distribution is available, this the sampler should be modified to include this. Currently, however, this is not included in the code, so the appropriate adjustments have to be done by hand.

Author(s)

Florian Hartig

Examples

```
## Example for the use of SMC
# First we need a bayesianSetup - SMC makes most sense if we can for demonstration,
# we'll write a function that puts out the number of model calls

MultiNomialNoCor <- generateTestDensityMultiNormal(sigma = "no correlation")

parallell <- function(parMatrix){
  print(paste("Calling likelihood with", nrow(parMatrix), "parameter combinations"))
  out = apply(parMatrix, 1, MultiNomialNoCor)
  return(out)
}

bayesianSetup <- createBayesianSetup(likelihood = parallell, lower = rep(-10, 3),
                                     upper = rep(10, 3), parallel = "external")

# Defining settings for the sampler
# First we use the sampler for rejection sampling
settings <- list(initialParticles = 1000, iterations = 1, resampling = FALSE)

# Running the sampler
out1 <- runMCMC(bayesianSetup = bayesianSetup, sampler = "SMC", settings = settings)
#plot(out1)

# Now for sequential Monte Carlo
settings <- list(initialParticles = 100, iterations = 5, resamplingSteps = 1)
out2 <- runMCMC(bayesianSetup = bayesianSetup, sampler = "SMC", settings = settings)
#plot(out2)
```

stopParallel

Function to close cluster in BayesianSetup

Description

Function closes the parallel executer (if available)

Usage

```
stopParallel(bayesianSetup)
```

Arguments

bayesianSetup object of class BayesianSetup

Author(s)

Stefan Paul

testDensityBanana	<i>Banana-shaped density function</i>
-------------------	---------------------------------------

Description

Banana-shaped density function

Usage

```
testDensityBanana(p)
```

Arguments

p 2-dim parameter vector

Note

inspired from package FMEcmc, seems to go back to Laine M (2008). Adaptive MCMC Methods with Applications in Environmental and Models. Finnish Meteorological Institute Contributions 69. ISBN 978-951-697-662-7.

Author(s)

Florian Hartig

See Also

[generateTestDensityMultiNormal](#)
[testLinearModel](#)

testDensityInfinity	<i>Test function infinity ragged</i>
---------------------	--------------------------------------

Description

Test function infinity ragged

Usage

```
testDensityInfinity(x, error = F)
```

Arguments

x 2-dim parameter vector
error should error or infinity be returned

Author(s)

Florian Hartig

See Also

[generateTestDensityMultiNormal](#)
[testDensityBanana](#)

`testDensityMultiNormal`*3d Mutivariate Normal likelihood*

Description

3d Mutivariate Normal likelihood

Usage`testDensityMultiNormal(x, sigma = "strongcorrelation")`**Arguments**

<code>x</code>	a parameter vector of arbitrary length
<code>sigma</code>	either a correlation matrix, or "strongcorrelation", or "no correlation"

`testDensityNormal`*Normal likelihood*

Description

Normal likelihood

Usage`testDensityNormal(x, sum = T)`**Arguments**

<code>x</code>	a parameter vector of arbitrary length
<code>sum</code>	if likelihood should be summed or not

Author(s)

Florian Hartig

testLinearModel	<i>Fake model, returns a $ax + b$ linear response to 2-param vector</i>
-----------------	--

Description

Fake model, returns a $ax + b$ linear response to 2-param vector

Usage

```
testLinearModel(x, env = NULL)
```

Arguments

x	2-dim parameter vector
env	optional, environmental covariate

Author(s)

Florian Hartig

See Also

[generateTestDensityMultiNormal](#)
[testDensityBanana](#)

Examples

```
x = c(1,2)
y = testLinearModel(x)
plot(y)
```

tracePlot	<i>Trace plot for MCMC class</i>
-----------	----------------------------------

Description

Trace plot for MCMC class

Usage

```
tracePlot(sampler, thin = "auto", ...)
```

Arguments

sampler	an object of class MCMC sampler
thin	determines the thinning intervall of the chain
...	additional parameters to pass on to the getSample , for example parametersOnly =F, or start = 1000

See Also

[marginalPlot](#)
[plotTimeSeries](#)
[correlationPlot](#)

Examples

```
# set up and run the MCMC
ll <- function(x) sum(dnorm(x, log = TRUE))
setup <- createBayesianSetup(likelihood = ll, lower = c(-10, -10), upper = c(10,10))
settings <- list(iterations = 2000)
out <- runMCMC(bayesianSetup = setup, settings = settings, sampler = "Metropolis")

# plot the trace
tracePlot(sampler = out, thin = 10)
tracePlot(sampler = out, thin = 50)
```

Twalk

T-walk MCMC

Description

T-walk MCMC

Usage

```
Twalk(bayesianSetup, settings = list(iterations = 10000, at = 6, aw = 1.5, pn1
= NULL, Ptrav = 0.4918, Pwalk = 0.4918, Pblow = 0.0082, burnin = 0, thin = 1,
startValue = NULL, consoleUpdates = 100, message = TRUE))
```

Arguments

bayesianSetup	Object of class 'bayesianSetup' or 'bayesianOuput'.
settings	list with parameter values.
iterations	Number of model evaluations
at	"traverse" move proposal parameter. Default to 6
aw	"walk" move proposal parameter. Default to 1.5
pn1	Probability determining the number of parameters that are changed

P _{trav}	Move probability of "traverse" moves, default to 0.4918
P _{walk}	Move probability of "walk" moves, default to 0.4918
P _{blow}	Move probability of "traverse" moves, default to 0.0082
burnin	number of iterations treated as burn-in. These iterations are not recorded in the chain.
thin	thinning parameter. Determines the interval in which values are recorded.
startValue	Matrix with start values
consoleUpdates	Intervall in which the sampling progress is printed to the console
message	logical determines whether the sampler's progress should be printed

Details

The probability of "hop" moves is 1 minus the sum of all other probabilities.

Value

Object of class bayesianOutput.

Author(s)

Stefan Paul

References

Christen, J. Andres, and Colin Fox. "A general purpose sampling algorithm for continuous distributions (the t-walk)." *Bayesian Analysis* 5.2 (2010): 263-281.

updateProposalGenerator

To update settings of an existing proposal generator

Description

To update settings of an existing proposal generator

Usage

```
updateProposalGenerator(proposal, chain = NULL, message = F, eps = 1e-10,
  manualScaleAdjustment = 1)
```

Arguments

proposal	an object of class proposalGenerator
chain	a chain to create the covariance matrix from (optional)
message	whether to print an updating message
eps	numeric tolerance for covariance
manualScaleAdjustment	optional adjustment for the covariance scale (multiplicative)

Details

The this function can be applied in 2 ways 1) update the covariance given an MCMC chain, and 2) update the proposal generator after parameters have been changed

violinPlot

Violin Plot

Description

Function to plot classic violin plots, as well as "half violin plots" (density plots).

Usage

```
violinPlot(x, at, .range = 1, add = FALSE, horizontal = TRUE,
  which = "both", relToAt = "above", plotQBox = TRUE, plotMed = TRUE,
  col = "orangered", border = "black", lwd = par("lwd"),
  colQBox = "black", borderQBox = "black", colMed = "white",
  pchMed = 19, res = 500, main = "")
```

Arguments

x	vector of values to plot
at	position of the plot when add is TRUE
.range	maximum height if horizontal or width if vertical of the plot when add is set to FALSE
add	logical, determining whether the plot should be added to an existing plot window
horizontal	logical, determining whether the plot should be horizontal, if FALSE the plot will be vertical
which	string, either "both" for a classic violing plot, or "top" or "bottom" to plot only the half violin
relToAt	string, one of "centered", "above", "below", "left", "right". Determining the relative position to at.
plotQBox	logical, determining whether the quantile box should be plotted
plotMed	logical, determining whether the median should be plotted
col	color of the violin
border	color of the border of the violin
lwd	line width if the border of the violin
colQBox	color of quantile box
borderQBox	color of the border of the quantile box
colMed	color of the median point
pchMed	pch for median point
res	"resolution" of the violin. Determining how many discrete points should be used to calculate the density kernel.
main	header text, only applicable, if 'add' is FALSE

Author(s)

Tankred Ott

VSEM

*Very simple ecosystem model***Description**

A very simple ecosystem model, based on three carbon pools and a basic LUE model

Usage

```
VSEM(pars = c(KEXT = 0.5, LAR = 1.5, LUE = 0.002, GAMMA = 0.4, tauV = 1440,
  tauS = 27370, tauR = 1440, Av = 0.5, Cv = 3, Cs = 15, Cr = 3), PAR,
  C = TRUE)
```

Arguments

pars	a parameter vector with parameters and initial states
PAR	Forcing, photosynthetically active radiation (PAR) MJ /m2 /day
C	switch to choose whether to use the C or R version of the model. C is much faster.

Details

This Very Simple Ecosystem Model (VSEM) is a 'toy' model designed to be very simple but yet bear some resemblance to deterministic processed based ecosystem models (PBM) that are commonly used in forest modelling.

The model determines the accumulation of carbon in the plant and soil from the growth of the plant via photosynthesis and senescence to the soil which respire carbon back to the atmosphere.

The model calculates Gross Primary Productivity (GPP) using a very simple light-use efficiency (LUE) formulation multiplied by light interception. Light interception is calculated via Beer's law with a constant light extinction coefficient operating on Leaf Area Index (LAI).

A parameter (GAMMA) determines the fraction of GPP that is autotrophic respiration. The Net Primary Productivity (NPP) is then allocated to above and below-ground vegetation via a fixed allocation fraction. Carbon is lost from the plant pools to a single soil pool via fixed turnover rates. Heterotrophic respiration in the soil is determined via a soil turnover rate.

The model equations are

– Photosynthesis

$$LAI = LAR * Cv$$

$$GPP = PAR * LUE * (1 - \exp^{-KEXT * LAI})$$

$$NPP = (1 - GAMMA) * GPP$$

– State equations

$$dC_v/dt = A_v * NPP - C_v/tauV$$

$$dC_r/dt = (1.0 - A_v) * NPP - C_r/tauR$$

$$dC_s/dt = C_r/tauR + C_v/tauV - C_s/tauS$$

The model time-step is daily.

– VSEM inputs:

PAR Photosynthetically active radiation (PAR) MJ /m2 /day

– VSEM parameters:

KEXT Light extinction coefficient m2 ground area / m2 leaf area

LAR Leaf area ratio m2 leaf area / kg aboveground vegetation

LUE Light-Use Efficiency (kg C MJ-1 PAR)

GAMMA Autotrophic respiration as a fraction of GPP

tauV Longevity of aboveground vegetation days

tauR Longevity of belowground vegetation days

tauS Residence time of soil organic matter d

– VSEM states:

Cv Above-ground vegetation pool kg C / m2

Cr Below-ground vegetation pool kg C / m2

Cs Carbon in organic matter kg C / m2

– VSEM fluxes:

G Gross Primary Productivity kg C /m2 /day

NPP Net Primary Productivity kg C /m2 /day

NEE Net Ecosystem Exchange kg C /m2 /day

Value

a matrix with columns NEE, CV, CR and CS units and explanations see details

Author(s)

David Cameron, R and C implementation by Florian Hartig

See Also

[VSEMgetDefaults](#), [VSEMcreatePAR](#), [VSEMcreateLikelihood](#)

Examples

```
## This example shows how to run and calibrate the VSEM model

library(BayesianTools)

# Create input data for the model
PAR <- VSEMcreatePAR(1:1000)
plot(PAR, main = "PAR (driving the model)", xlab = "Day")

# load reference parameter definition (upper, lower prior)
refPars <- VSEMgetDefaults()
# this adds one additional parameter for the likelihood standard deviation (see below)
refPars[12,] <- c(2, 0.1, 4)
rownames(refPars)[12] <- "error-sd"
head(refPars)

# create some simulated test data
# generally recommended to start with simulated data before moving to real data
referenceData <- VSEM(refPars$best[1:11], PAR) # model predictions with reference parameters
referenceData[,1] = 1000 * referenceData[,1]
# this adds the error - needs to conform to the error definition in the likelihood
obs <- referenceData + rnorm(length(referenceData), sd = refPars$best[12])
oldpar <- par(mfrow = c(2,2))
for (i in 1:4) plotTimeSeries(observed = obs[,i],
                             predicted = referenceData[,i], main = colnames(referenceData)[i])

# Best to program in a way that we can choose easily which parameters to calibrate
parSel = c(1:6, 12)

# here is the likelihood
likelihood <- function(par, sum = TRUE){
  # set parameters that are not calibrated on default values
  x = refPars$best
  x[parSel] = par
  predicted <- VSEM(x[1:11], PAR) # replace here VSEM with your model
  predicted[,1] = 1000 * predicted[,1] # this is just rescaling
  diff <- c(predicted[,1:4] - obs[,1:4]) # difference between observed and predicted
  # univariate normal likelihood. Note that there is a parameter involved here that is fit
  llValues <- dnorm(diff, sd = x[12], log = TRUE)
  if (sum == FALSE) return(llValues)
  else return(sum(llValues))
}

# optional, you can also directly provide lower, upper in the createBayesianSetup, see help
prior <- createUniformPrior(lower = refPars$lower[parSel],
                           upper = refPars$upper[parSel], best = refPars$best[parSel])

bayesianSetup <- createBayesianSetup(likelihood, prior, names = rownames(refPars)[parSel])

# settings for the sampler, iterations should be increased for real applicatoin
```

```

settings <- list(iterations = 2000, nrChains = 2)

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "DEzs", settings = settings)

## Not run:

plot(out)
summary(out)
marginalPlot(out, scale = T)
gelmanDiagnostics(out) # should be below 1.05 for all parameters to demonstrate convergence

# Posterior predictive simulations

# Create a prediction function
createPredictions <- function(par){
  # set the parameters that are not calibrated on default values
  x = refPars$best
  x[parSel] = par
  predicted <- VSEM(x[1:11], PAR) # replace here VSEM with your model
  return(predicted[,1] * 1000)
}

# Create an error function
createError <- function(mean, par){
  return(rnorm(length(mean), mean = mean, sd = par[7]))
}

# plot prior predictive distribution and prior predictive simulations
plotTimeSeriesResults(sampler = out, model = createPredictions, observed = obs[,1],
  error = createError, prior = TRUE, main = "Prior predictive")

# plot posterior predictive distribution and posterior predictive simulations
plotTimeSeriesResults(sampler = out, model = createPredictions, observed = obs[,1],
  error = createError, main = "Posterior predictive")

#####
# Demonstrating the updating of the prior from old posterior
# Note that it is usually more exact to rerun the MCMC
# with all (old and new) data, instead of updating the prior
# because likely some information is lost when approximating the
# Posterior by a multivariate normal

settings <- list(iterations = 5000, nrChains = 2)

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "DEzs", settings = settings)

plot(out)
correlationPlot(out, start = 1000)

newPrior = createPriorDensity(out, method = "multivariate",
  eps = 1e-10,
  lower = refPars$lower[parSel],

```

```

upper = refPars$upper[parSel], start= 1000)

bayesianSetup <- createBayesianSetup(likelihood = likelihood,
                                     prior = newPrior,
                                     names = rownames(refPars)[parSel] )

# check boundaries are correct set
bayesianSetup$prior$sampler() < refPars$lower[parSel]
bayesianSetup$prior$sampler() > refPars$upper[parSel]

# check prior looks similar to posterior
x = bayesianSetup$prior$sampler(2000)
correlationPlot(x, thin = F)

out <- runMCMC(bayesianSetup = bayesianSetup, sampler = "DEzs", settings = settings)

plot(out)
correlationPlot(out)

plotTimeSeriesResults(sampler = out,
                      model = createPredictions,
                      observed = obs[,1],
                      error = createError,
                      prior = F, main = "Posterior predictive")

plotTimeSeriesResults(sampler = out,
                      model = createPredictions,
                      observed = obs[,1],
                      error = createError,
                      prior = T, main = "Prior predictive")

## End(Not run)

par(oldpar)

```

vsemC

C version of the VSEM model

Description

C version of the VSEM model

Usage

```
vsemC(par, PAR)
```

Arguments

par	parameter vector
PAR	Photosynthetically active radiation (PAR) MJ /m2 /day

VSEMcreateLikelihood *Create an example dataset, and from that a likelihood or posterior for the VSEM model*

Description

Create an example dataset, and from that a likelihood or posterior for the VSEM model

Usage

```
VSEMcreateLikelihood(likelihoodOnly = F, plot = F, selection = c(1:6, 12))
```

Arguments

likelihoodOnly	switch to decide whether to create only a likelihood, or a full bayesianSetup with uniform priors.
plot	switch to decide whether data should be plotted
selection	vector containing the indices of the selected parameters

Details

The purpose of this function is to be able to conveniently create a likelihood for the VSEM model for demonstration purposes. The function creates example data → likelihood → BayesianSetup, where the latter is the

Author(s)

Florian Hartig

VSEMcreatePAR *Create a random radiation (PAR) time series*

Description

Create a random radiation (PAR) time series

Usage

```
VSEMcreatePAR(days = 1:(3 * 365))
```

Arguments

days days to calculate the PAR for

Author(s)

David Cameron, R implementation by Florian Hartig

VSEMgetDefaults *returns the default values for the VSEM*

Description

returns the default values for the VSEM

Usage

```
VSEMgetDefaults()
```

Value

a data.frame

WAIC *calculates the WAIC*

Description

calculates the WAIC

Usage

```
WAIC(bayesianOutput, numSamples = 1000, ...)
```

Arguments

bayesianOutput an object of class BayesianOutput
numSamples the number of samples to calculate the WAIC
... optional values to be passed on the the getSample function

Details

The WAIC is constructed as

$$WAIC = -2 * (lppd - p_{WAIC})$$

The lppd (log pointwise predictive density), defined in Gelman et al., 2013, eq. 4 as

$$lppd = \sum_{i=1}^n \log \left(\frac{1}{S} \sum_{s=1}^S p(y_i | \theta^s) \right)$$

The value of p_{WAIC} can be calculated in two ways, the method used is determined by the method argument.

Method 1 is defined as,

$$p_{WAIC1} = 2 \sum_{i=1}^n \left(\log \left(\frac{1}{S} \sum_{s=1}^S p(y_i | \theta^s) \right) - \frac{1}{S} \sum_{s=1}^S \log p(y_i | \theta^s) \right)$$

Method 2 is defined as,

$$p_{WAIC2} = 2 \sum_{i=1}^n V_{s=1}^S (\log p(y_i | \theta^s))$$

where $V_{s=1}^S$ is the sample variance.

Note

The function requires that the likelihood passed on to BayesianSetup contains the option `sum = T/F`, with default F. If set to true, the likelihood for each data point must be returned.

Author(s)

Florian Hartig

References

Gelman, Andrew and Jessica Hwang and Aki Vehtari (2013), "Understanding Predictive Information Criteria for Bayesian Models," http://www.stat.columbia.edu/~gelman/research/unpublished/waic_understand_final.pdf.

Watanabe, S. (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory", Journal of Machine Learning Research, <http://www.jmlr.org/papers/v11/watanabe10a.html>.

See Also

[DIC](#), [MAP](#), [marginalLikelihood](#)

Examples

```
bayesianSetup <- createBayesianSetup(likelihood = testDensityNormal,  
                                     prior = createUniformPrior(lower = rep(-10,2),  
                                                                upper = rep(10,2)))  
  
# likelihood density needs to have option sum = FALSE  
  
testDensityNormal(c(1,1,1), sum = FALSE)  
bayesianSetup$likelihood$density(c(1,1,1), sum = FALSE)  
bayesianSetup$likelihood$density(matrix(rep(1,9), ncol = 3), sum = FALSE)  
  
# running MCMC  
  
out = runMCMC(bayesianSetup = bayesianSetup)  
  
WAIC(out)
```

Index

AM, [3](#), [54](#)
applySettingsDefault, [4](#), [59](#)

BayesianTools, [5](#)
BayesianTools-package (BayesianTools), [5](#)
bridgesample, [5](#)

checkBayesianSetup, [7](#), [11](#)
convertCoda, [8](#)
correctThin, [8](#)
correlationPlot, [9](#), [52](#), [57](#), [68](#)
createBayesianSetup, [5](#), [7](#), [10](#), [13](#), [15–17](#),
[19](#), [23](#), [24](#), [53](#), [59](#), [60](#), [62](#), [63](#)
createBetaPrior, [13](#), [17](#), [19](#), [23](#), [24](#)
createBreakMat, [14](#), [55](#)
createLikelihood, [11](#), [14](#)
createMcmcSamplerList, [15](#), [60](#)
createMixWithDefaults, [16](#)
createPosterior, [16](#)
createPrior, [10](#), [11](#), [13](#), [17](#), [19](#), [23](#), [24](#)
createPriorDensity, [13](#), [17](#), [18](#), [23](#), [24](#), [60](#)
createProposalGenerator, [19](#), [53](#)
createSmcSamplerList, [22](#)
createTruncatedNormalPrior, [13](#), [17](#), [19](#),
[22](#), [24](#)
createUniformPrior, [13](#), [17](#), [19](#), [23](#), [23](#)

DE, [5](#), [24](#), [27](#), [60](#)
DEzs, [5](#), [26](#), [26](#), [60](#)
DIC, [8](#), [28](#), [49](#), [50](#), [78](#)
DR, [29](#)
DRAM, [29](#)
DREAM, [5](#), [30](#), [34](#), [60](#)
DREAMzs, [5](#), [32](#), [32](#), [60](#)

gelmanDiagnostics, [34](#)
generateParallelExecuter, [35](#)
generateTestDensityMultiNormal, [36](#),
[65–67](#)
getCredibleIntervals, [37](#), [40](#), [41](#)

getDharmaResiduals, [38](#)
getMetropolisDefaultSettings, [38](#)
getPanels, [39](#)
getPossibleSamplerTypes, [39](#)
getPredictiveDistribution, [37](#), [40](#), [41](#)
getPredictiveIntervals, [37](#), [40](#), [40](#)
getSample, [9](#), [28](#), [34](#), [41](#), [43](#), [49](#), [51](#), [58](#), [60](#), [68](#)
getVolume, [43](#)
GOF, [44](#)

histMarginal, [45](#)

likelihoodAR1, [15](#), [46](#)
likelihoodIidNormal, [15](#), [46](#)
logSumExp, [47](#)

M, [47](#)
MAP, [8](#), [28](#), [48](#), [50](#), [78](#)
marginalLikelihood, [6](#), [28](#), [49](#), [49](#), [78](#)
marginalPlot, [10](#), [51](#), [57](#), [68](#)
mcmcMultipleChains, [52](#)
Metropolis, [5](#), [53](#), [60](#)
metropolisRatio, [55](#)

plot, [57](#), [58](#)
plotHist, [55](#)
plotSensitivity, [56](#)
plotTimeSeries, [10](#), [56](#), [68](#)
plotTimeSeriesResiduals, [57](#)
plotTimeSeriesResults, [57](#), [58](#)

rescale, [59](#)
runMCMC, [4](#), [5](#), [59](#)

sampleEquallySpaced, [61](#)
sampleMetropolis, [62](#)
setupStartProposal, [62](#)
smcSampler, [5](#), [60](#), [63](#)
stopParallel, [64](#)

testDensityBanana, [36](#), [65](#), [66](#), [67](#)

testDensityInfinity, [65](#)
testDensityMultiNormal, [66](#)
testDensityNormal, [66](#)
testLinearModel, [36](#), [65](#), [67](#)
tracePlot, [10](#), [52](#), [57](#), [67](#)
Twalk, [5](#), [60](#), [68](#)

updateProposalGenerator, [20](#), [69](#)

violinPlot, [70](#)
VSEM, [71](#)
vsemC, [75](#)
VSEMcreateLikelihood, [72](#), [76](#)
VSEMcreatePAR, [72](#), [76](#)
VSEMgetDefaults, [72](#), [77](#)

WAIC, [28](#), [49](#), [50](#), [77](#)