Package 'LaplacesDemon'

October 12, 2022

Version 16.1.6

Title Complete Environment for Bayesian Inference

Depends R (>= 3.0.0)

Imports parallel, grDevices, graphics, stats, utils

Suggests KernSmooth

ByteCompile TRUE

Description Provides a complete environment for Bayesian inference using a variety of different samplers (see ?LaplacesDemon for an overview).

License MIT + file LICENSE

URL https://github.com/LaplacesDemonR/LaplacesDemon

BugReports https://github.com/LaplacesDemonR/LaplacesDemon/issues

NeedsCompilation no

Author Byron Hall [aut],
Martina Hall [aut],
Statisticat, LLC [aut],
Eric Brown [ctb],
Richard Hermanson [ctb],
Emmanuel Charpentier [ctb],
Daniel Heck [ctb],
Stephane Laurent [ctb],
Quentin F. Gronau [ctb],
Henrik Singmann [cre]

Maintainer Henrik Singmann <singmann+LaplacesDemon@gmail.com>

Repository CRAN

Date/Publication 2021-07-09 14:00:02 UTC

R topics documented:

LaplacesDemon-package *Complete Environment for Bayesian Inference*

Description

Provides a complete environment for Bayesian inference using a variety of different samplers (see ?LaplacesDemon for an overview).

Details

The DESCRIPTION file:

Package:	LaplacesDemon
Version:	16.1.6
Title:	Complete Environment for Bayesian Inference
Authors@R:	c(person("Byron", "Hall", role = "aut"), person("Martina", "Hall", role = "aut"), person(family="Statisticat, L
Depends:	R (>= 3.0.0)
Imports:	parallel, grDevices, graphics, stats, utils
Suggests:	KernSmooth
ByteCompile:	TRUE
Description:	Provides a complete environment for Bayesian inference using a variety of different samplers (see ?LaplacesI
License:	MIT + file LICENSE
URL:	https://github.com/LaplacesDemonR/LaplacesDemon
BugReports:	https://github.com/LaplacesDemonR/LaplacesDemon/issues
Author:	Byron Hall [aut], Martina Hall [aut], Statisticat, LLC [aut], Eric Brown [ctb], Richard Hermanson [ctb], Emm
Maintainer:	Henrik Singmann <singmann+LaplacesDemon@gmail.com>

Index of help topics:

Gelman.Diagnostic	Gelman and Rubin's MCMC Convergence Diagnostic
Geweke.Diagnostic	Geweke's Convergence Diagnostic
Hangartner.Diagnostic	Hangartner's Convergence Diagnostic
Heidelberger.Diagnostic	Heidelberger and Welch's MCMC Convergence Diagnostic
IAT	Integrated Autocorrelation Time
Importance	Variable Importance
IterativeQuadrature	Iterative Quadrature
Juxtapose	Juxtapose MCMC Algorithm Inefficiency
KLD	Kullback-Leibler Divergence (KLD)
KS.Diagnostic	Kolmogorov-Smirnov Convergence Diagnostic
LML	Logarithm of the Marginal Likelihood
LPL.interval	Lowest Posterior Loss Interval
LaplaceApproximation	Laplace Approximation
LaplacesDemon	Laplace's Demon
LaplacesDemon-package	Complete Environment for Bayesian Inference
LaplacesDemon.RAM	LaplacesDemon RAM Estimate
Levene.Test	Levene's Test
LossMatrix	Loss Matrix
MCSE	Monte Carlo Standard Error
MISS	Multiple Imputation Sequential Sampling
MinnesotaPrior	Minnesota Prior
Mode	The Mode(s) of a Vector
Model.Spec.Time	Model Specification Time
PMC	Population Monte Carlo
PMC.RAM	PMC RAM Estimate
PosteriorChecks	Posterior Checks
Raftery.Diagnostic	Raftery and Lewis's diagnostic
RejectionSampling	Rejection Sampling
SIR	Sampling Importance Resampling
SensitivityAnalysis	Sensitivity Analysis
Stick	Truncated Stick-Breaking
Thin	Thin
Validate	Holdout Validation
VariationalBayes	Variational Bayes
WAIC	Widely Applicable Information Criterion
as.covar	Proposal Covariance
as.indicator.matrix	Matrix Utility Functions
as.initial.values	Initial Values
as.parm.names	Parameter Names
as.ppc	As Posterior Predictive Check
burnin	Burn-in
caterpillar.plot	Caterpillar Plot
cloglog	The log-log and complementary log-log functions
cond.plot	Conditional Plots
dStick	Truncated Stick-Breaking Prior Distribution
dalaplace	Asymmetric Laplace Distribution: Univariate

dallaplace	Asymmetric Log-Laplace Distribution
daml	Asymmetric Multivariate Laplace Distribution
dbern	Bernoulli Distribution
dcat	Categorical Distribution
dcrmrf	Continuous Relaxation of a Markov Random Field Distribution
ddirichlet	Dirichlet Distribution
de.Finetti.Game	de Finetti's Game
deburn	De-Burn
delicit	Prior Elicitation
demonchoice	Demon Choice Data Set
demonfx	Demon FX Data Set
demonsessions	Demon Sessions Data Set
demonsnacks	Demon Snacks Data Set
demontexas	Demon Space-Time Data Set
dgpd	Generalized Pareto Distribution
dgpois	Generalized Poisson Distribution
dhalfcauchy	Half-Cauchy Distribution
dhalfnorm	Half-Normal Distribution
dhalft	Half-t Distribution
dhs	Horseshoe Distribution
dhuangwand	Huang-Wand Distribution
dhyperg	Hyperprior-g Prior and Zellner's g-Prior
dinvbeta	Inverse Beta Distribution
dinvchisq	(Scaled) Inverse Chi-Squared Distribution
dinvgamma	Inverse Gamma Distribution
dinvgaussian	Inverse Gaussian Distribution
dinvmatrixgamma	Inverse Matrix Gamma Distribution
dinvwishart	Inverse Wishart Distribution
dinvwishartc	Inverse Wishart Distribution: Cholesky Parameterization
dlaplace	Laplace Distribution: Univariate Symmetric
dlaplacem	Mixture of Laplace Distributions
dlaplacep	Laplace Distribution: Precision Parameterization
dlasso	LASSO Distribution
dllaplace	Log-Laplace Distribution: Univariate Symmetric
dlnormp	Log-Normal Distribution: Precision Parameterization
dmatrixgamma	Matrix Gamma Distribution
dmatrixnorm	Matrix Normal Distribution
dmvc	Multivariate Cauchy Distribution
dmvcc	Multivariate Cauchy Distribution: Cholesky Parameterization
dmvcp	Multivariate Cauchy Distribution: Precision Parameterization
dmvcpc	Multivariate Cauchy Distribution: Precision-Cholesky Parameterization

dmvl	Multivariate Laplace Distribution
dmvlc	Multivariate Laplace Distribution: Cholesky Parameterization
dmvn	Multivariate Normal Distribution
dmvnc	Multivariate Normal Distribution: Cholesky Parameterization
dmvnp	Multivariate Normal Distribution: Precision Parameterization
dmvnpc	Multivariate Normal Distribution: Precision-Cholesky Parameterization
dmvpe	Multivariate Power Exponential Distribution
dmvpec	Multivariate Power Exponential Distribution: Cholesky Parameterization
dmvpolya	Multivariate Polya Distribution
dmvt	Multivariate t Distribution
dmvtc	Multivariate t Distribution: Cholesky Parameterization
dmvtp	Multivariate t Distribution: Precision Parameterization
dmvtpc	Multivariate t Distribution: Precision-Cholesky Parameterization
dnorminvwishart	Normal-Inverse-Wishart Distribution
dnormlaplace	Normal-Laplace Distribution: Univariate Asymmetric
dnormm	Mixture of Normal Distributions
dnormp	Normal Distribution: Precision Parameterization
dnormv	Normal Distribution: Variance Parameterization
dnormwishart	Normal-Wishart Distribution
dpareto	Pareto Distribution
dpe	Power Exponential Distribution: Univariate Symmetric
dsdlaplace	Skew Discrete Laplace Distribution: Univariate
dsiw	Scaled Inverse Wishart Distribution
dslaplace	Skew-Laplace Distribution: Univariate
dst	Student t Distribution: Univariate
dstp	Student t Distribution: Precision Parameterization
dtrunc	Truncated Distributions
dwishart	Wishart Distribution
dwishartc	Wishart Distribution: Cholesky Parameterization
dyangberger	Yang-Berger Distribution
interval	Constrain to Interval
is.appeased	Appeased
is.bayesfactor	Logical Check of Classes
is.bayesian	Logical Check of a Bayesian Model
is.constant	Logical Check of a Constant
is.constrained	Logical Check of Constraints
is.data	Logical Check of Data

is.model	Logical Check of a Model
is.proper	Logical Check of Propriety
is.stationary	Logical Check of Stationarity
joint.density.plot	Joint Density Plot
joint.pr.plot	Joint Probability Region Plot
logit	The logit and inverse-logit functions
p.interval	Probability Interval
plot.bmk	Plot Hellinger Distances
plot.demonoid	Plot samples from the output of Laplace's Demon
plot.demonoid.ppc	Plots of Posterior Predictive Checks
plot.importance	Plot Variable Importance
plot.iterquad	Plot the output of 'IterativeQuadrature'
plot.iterquad.ppc	Plots of Posterior Predictive Checks
plot.juxtapose	Plot MCMC Juxtaposition
plot.laplace	Plot the output of 'LaplaceApproximation'
plot.laplace.ppc	Plots of Posterior Predictive Checks
plot.miss	Plot samples from the output of MISS
plot.pmc	Plot samples from the output of PMC
plot.pmc.ppc	Plots of Posterior Predictive Checks
plot.vb	Plot the output of 'VariationalBayes'
plot.vb.ppc	Plots of Posterior Predictive Checks
plotMatrix	Plot a Numerical Matrix
plotSamples	Plot Samples
predict.demonoid	Posterior Predictive Checks
predict.iterquad	Posterior Predictive Checks
predict.laplace	Posterior Predictive Checks
predict.pmc	Posterior Predictive Checks
predict.vb	Posterior Predictive Checks
print.demonoid	Print an object of class 'demonoid' to the screen.
print.heidelberger	Print an object of class 'heidelberger' to the screen.
print.iterquad	Print an object of class 'iterquad' to the screen.
print.laplace	Print an object of class 'laplace' to the screen.
print.miss	Print an object of class 'miss' to the screen.
print.pmc	Print an object of class 'pmc' to the screen.
print.raftery	Print an object of class 'raftery' to the screen.
print.vb	Print an object of class 'vb' to the screen.
server_Listening	Server Listening
summary.demonoid.ppc	Posterior Predictive Check Summary
summary.iterquad.ppc	Posterior Predictive Check Summary
summary.laplace.ppc	Posterior Predictive Check Summary
summary.miss	MISS Summary
summary.pmc.ppc	Posterior Predictive Check Summary
summary.vb.ppc	Posterior Predictive Check Summary

The goal of LaplacesDemon, often referred to as LD, is to provide a complete and self-contained Bayesian environment within R. For example, this package includes dozens of MCMC algorithms, Laplace Approximation, iterative quadrature, variational Bayes, parallelization, big data, PMC, over 100 examples in the "Examples" vignette, dozens of additional probability distributions, numerous MCMC diagnostics, Bayes factors, posterior predictive checks, a variety of plots, elicitation, parameter and variable importance, Bayesian forms of test statistics (such as Durbin-Watson, Jarque-Bera, etc.), validation, and numerous additional utility functions, such as functions for multimodality, matrices, or timing your model specification. Other vignettes include an introduction to Bayesian inference, as well as a tutorial.

No further development of this package is currently being done as the original maintainer has stopped working on the package. Contributions to this package are welcome at https://github.com/LaplacesDemonR/LaplacesDemon.

The main function in this package is the `LaplacesDemon` function, and the best place to start is probably with the LaplacesDemon Tutorial vignette.

Author(s)

NA

Maintainer: NA

`ABB` *Approximate Bayesian Bootstrap*

Description

This function performs multiple imputation (MI) with the Approximate Bayesian Bootstrap (ABB) of Rubin and Schenker (1986).

Usage

```
ABB(X, K=1)
```

Arguments

`X`	This is a vector or matrix of data that must include both observed and missing values. When `X` is a matrix, missing values must occur somewhere in the set, but are not required to occur in each variable.
`K`	This is the number of imputations.

Details

The Approximate Bayesian Bootstrap (ABB) is a modified form of the `BayesianBootstrap` (Rubin, 1981) that is used for multiple imputation (MI). Imputation is a family of statistical methods for replacing missing values with estimates. Introduced by Rubin and Schenker (1986) and Rubin (1987), MI is a family of imputation methods that includes multiple estimates, and therefore includes variability of the estimates.

The data, $\mathbf{X}$, are assumed to be independent and identically distributed (IID), contain both observed and missing values, and its missing values are assumed to be ignorable (meaning enough information is available in the data that the missingness mechanism can be ignored, if the information is used properly) and Missing Completely At Random (MCAR). When ABB is used in conjunction with a propensity score (described below), missing values may be Missing At Random (MAR).

ABB does not add auxiliary information, but performs imputation with two sampling (with replacement) steps. First, $\mathbf{X}^{\star}_{obs}$ is sampled from $\mathbf{X}_{obs}$. Then, $\mathbf{X}^{\star}_{mis}$ is sampled from $\mathbf{X}^{\star}_{obs}$. The result is a sample of the posterior predictive distribution of $(\mathbf{X}_{mis}|\mathbf{X}_{obs})$. The first sampling step is also known as hotdeck imputation, and the second sampling step changes the variance. Since auxiliary information is not included, ABB is appropriate for missing values that are ignorable and MCAR.

Auxiliary information may be included in the process of imputation by introducing a propensity score (Rosenbaum and Rubin, 1983; Rosenbaum and Rubin, 1984), which is an estimate of the probability of missingness. The propensity score is often the result of a binary logit model, where missingness is predicted as a function of other variables. The propensity scores are discretized into quantile-based groups, usually quintiles. Each quintile must have both observed and missing values. ABB is applied to each quintile. This is called within-class imputation. It is assumed that the missing mechanism depends only on the variables used to estimate the propensity score.

With $K = 1$, ABB may be used in MCMC, such as in LaplacesDemon, more commonly along with a propensity score for missingness. MI is performed, despite $K = 1$, because imputation occurs at each MCMC iteration. The practical advantage of this form of imputation is the ease with which it may be implemented. For example, full-likelihood imputation should perform better, but requires a chain to be updated for each missing value.

An example of a limitation of ABB with propensity scores is to consider imputing missing values of income from age in a context where age and income have a positive relationship, and where the highest incomes are missing systematically. ABB with propensity scores should impute these highest missing incomes given the highest observed ages, but is unable to infer beyond the observed data.

ABB has been extended (Parzen et al., 2005) to reduce bias, by introducing a correction factor that is applied to the MI variance estimate. This correction may be applied to output from ABB.

Value

This function returns a list with K components, one for each set of imputations. Each component contains a vector of imputations equal in length to the number of missing values in the data.

ABB does not currently return the mean of the imputations, or the between-imputation variance or within-imputation variance.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Parzen, M., Lipsitz, S.R., and Fitzmaurice, G.M. (2005). "A Note on Reducing the Bias of the Approximate Bayesian Bootstrap Imputation Variance Estimator". *Biometrika*, 92, 4, p. 971–974.

Rosenbaum, P.R. and Rubin, D.B. (1983). "The Central Role of the Propensity Score in Observational Studies for Causal Effects". *Biometrika*, 70, p. 41–55.

Rosenbaum, P.R. and Rubin, D.B. (1984). "Reducing Bias in Observational Studies Using Subclassification in the Propensity Score". *Journal of the American Statistical Association*, 79, p. 516–524.

Rubin, D.B. (1981). "The Bayesian Bootstrap". *Annals of Statistics*, 9, p. 130–134.

Rubin, D.B. (1987). "Multiple Imputation for Nonresponse in Surveys". John Wiley and Sons: New York, NY.

Rubin, D.B. and Schenker, N. (1986). "Multiple Imputation for Interval Estimation from Simple Random Samples with Ignorable Nonresponse". *Journal of the American Statistical Association*, 81, p. 366–374.

See Also

BayesianBootstrap, LaplacesDemon, and MISS.

Examples

```
library(LaplacesDemon)

### Create Data
J <- 10 #Number of variables
m <- 20 #Number of missings
N <- 50 #Number of records
mu <- runif(J, 0, 100)
sigma <- runif(J, 0, 100)
X <- matrix(0, N, J)
for (j in 1:J) X[,j] <- rnorm(N, mu[j], sigma[j])

### Create Missing Values
M1 <- rep(0, N*J)
M2 <- sample(N*J, m)
M1[M2] <- 1
M <- matrix(M1, N, J)
X <- ifelse(M == 1, NA, X)

### Approximate Bayesian Bootstrap
imp <- ABB(X, K=1)

### Replace Missing Values in X (when K=1)
X.imp <- X
X.imp[which(is.na(X.imp))] <- unlist(imp)
X.imp
```

AcceptanceRate *Acceptance Rate*

Description

The Acceptance.Rate function calculates the acceptance rate per chain from a matrix of posterior MCMC samples.

Usage

```
AcceptanceRate(x)
```

Arguments

x	This required argument accepts a $S \times J$ numeric matrix of S posterior samples for J variables, such as `Posterior1` or `Posterior2` from an object of class `demonoid`.

Details

The acceptance rate of an MCMC algorithm is the percentage of iterations in which the proposals were accepted.

Optimal Acceptance Rates
The optimal acceptance rate varies with the number of parameters and by algorithm. Algorithms with componentwise Gaussian proposals have an optimal acceptance rate of 0.44, regardless of the number of parameters. Algorithms that update with multivariate Gaussian proposals tend to have an optimal acceptance rate that ranges from 0.44 for one parameter (one IID Gaussian target distribution) to 0.234 for an infinite number of parameters (IID Gaussian target distributions), and 0.234 is approached quickly as the number of parameters increases. The AHMC, HMC, and THMC algorithms have an optimal acceptance rate of 0.67, except with the algorithm specification `L=1`, where the optimal acceptance rate is 0.574. The target acceptance rate is specified in HMCDA and NUTS, and the recommended rate is 0.65 and 0.60 respectively. Some algorithms have an acceptance rate of 1, such as AGG, ESS, GG, GS (MISS only), SGLD, or Slice.

Global and Local Acceptance Rates
`LaplacesDemon` reports the global acceptance rate for the un-thinned chains. However, componentwise algorithms make a proposal per parameter, and therefore have a local acceptance rate for each parameter. Since only the global acceptance rate is reported, the `AcceptanceRate` function may be used to calculate the local acceptance rates from a matrix of un-thinned posterior samples.

Thinning
Thinned samples tend to have higher local acceptance rates than un-thinned samples. With enough samples and enough thinning, local acceptance rates approach 1. Local acceptance rates do not need to approach the optimal acceptance rates above. Conversely, local acceptance rates do not need to approach 1, because too much information may possibly be discarded by thinning. For more information on thinning, see the `Thin` function.

Diagnostics
The `AcceptanceRate` function may be used to calculate local acceptance rates on a matrix of thinned or un-thinned samples. Any chain with a local acceptance rate that is an outlier may be studied for reasons that may cause the outlier. A local acceptance rate outlier does not violate theory and is often acceptable, but may indicate a potential problem. Only some of the many potential problems include: identifiability, model misspecification, multicollinearity, multimodality, choice of prior distributions, or becoming trapped in a low-probability space. The solution to local acceptance rate outliers tends to be either changing the MCMC algorithm or re-specifying the model or priors. For example, an MCMC algorithm that makes multivariate Gaussian proposals for a large number of parameters may have low global and local acceptance rates when far from the target distributions.

Value

The AcceptanceRate function returns a vector of acceptance rates, one for each chain.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

LaplacesDemon, MISS, PosteriorChecks, and Thin.

Examples

```
library(LaplacesDemon)
AcceptanceRate(matrix(rnorm(5000),1000,5))
```

---

as.covar *Proposal Covariance*

Description

This function returns the most recent covariance matrix or a list of blocking covariance matrices from an object of class demonoid, the most recent covariance matrix from iterquad, laplace, or vb, the most recent covariance matrix from the chain with the lowest deviance in an object of class demonoid.hpc, and a number of covariance matrices of an object of class pmc equal to the number of mixture components. The returned covariance matrix or matrices are intended to be the initial proposal covariance matrix or matrices for future updates. A variance vector from an object of class demonoid or demonoid.hpc is converted to a covariance matrix.

Usage

```
as.covar(x)
```

Arguments

x	This is an object of class demonoid, demonoid.hpc, iterquad, laplace, pmc, or vb.

Details

Unless it is known beforehand how many iterations are required for iterative quadrature, Laplace Approximation, or Variational Bayes to converge, MCMC to appear converged, or the normalized perplexity to stabilize in PMC, multiple updates are necessary. An additional update, however, should not begin with the same proposal covariance matrix or matrices as the original update, because it will have to repeat the work already accomplished. For this reason, the as.covar function may be used at the end of an update to change the previous initial values to the latest values.

The as.covar function is most helpful with objects of class pmc that have multiple mixture components. For more information, see PMC.

Value

The returned value is a matrix (or array in the case of PMC with multiple mixture components) of the latest observed or proposal covariance, which may now be used as an initial proposal covariance matrix or matrices for a future update.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

See Also

IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, and VariationalBayes.

---

as.initial.values *Initial Values*

Description

This function returns the most recent posterior samples from an object of class demonoid or demonoid.hpc, the posterior means of an object of class iterquad, the posterior modes of an object of class laplace or vb, the posterior means of an object of class pmc with one mixture component, or the latest means of the importance sampling distribution of an object of class pmc with multiple mixture components. The returned values are intended to be the initial values for future updates.

Usage

```
as.initial.values(x)
```

Arguments

x	This is an object of class demonoid, demonoid.hpc, iterquad, laplace, pmc, or vb.

Details

Unless it is known beforehand how many iterations are required for IterativeQuadrature, LaplaceApproximation, or VariationalBayes to converge, MCMC in LaplacesDemon to appear converged, or the normalized perplexity to stabilize in PMC, multiple updates are necessary. An additional update, however, should not begin with the same initial values as the original update, because it will have to repeat the work already accomplished. For this reason, the as.initial.values function may be used at the end of an update to change the previous initial values to the latest values.

When using LaplacesDemon.hpc, as.initial.values should be used when the output is of class demonoid.hpc, before the Combine function is used to combine the multiple chains for use with Consort and other functions, because the Combine function returns an object of class demonoid, and the number of chains will become unknown. The Consort function may suggest using as.initial.values, but when applied to an object of class demonoid, it will return the latest values as if there were only one chain.

Value

The returned value is a vector (or matrix in the case of an object of class demonoid.hpc, or pmc with multiple mixture components) of the latest values, which may now be used as initial values for a future update.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Combine, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, and VariationalBayes.

---

as.parm.names *Parameter Names*

Description

This function creates a vector of parameter names from a list of parameters, and the list may contain any combination of scalars, vectors, matrices, upper-triangular matrices, and arrays.

Usage

```
as.parm.names(x, uppertri=NULL)
```

Arguments

x	This required argument is a list of named parameters. The list may contain scalars, vectors, matrices, and arrays. The value of the named parameters does not matter here, though they are usually set to zero. However, if a missing value occurs, then the associated element is omitted in the output.
uppertri	This optional argument must be a vector with a length equal to the number of named parameters. Each element in uppertri must be either a 0 or 1, where a 1 indicates that an upper triangular matrix will be used for the associated element in the vector of named parameters. Each element of uppertri is associated with a named parameter. The uppertri argument does not function with arrays.

Details

Each model function for IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, or VariationalBayes requires a vector of parameters (specified at first as Initial.Values) and a list of data. One component in the list of data must be named parm.names. Each element of parm.names is a name associated with the corresponding parameter in Initial.Values.

The parm.names vector is easy to program explicitly for a simple model, but can require considerably more programming effort for more complicated models. The as.parm.names function is a utility function designed to minimize programming by the user.

For example, a simple model may only require `parm.names <- c("alpha", "beta[1]", "beta[2]", "sigma")`. A more complicated model may contain hundreds of parameters that are a combination of scalars, vectors, matrices, upper-triangular matrices, and arrays, and is the reason for the `as.parm.names` function. The code for the above is `as.parm.names(list(alpha=0, beta=rep(0,2), sigma=0))`.

In the case of an upper-triangular matrix, simply pass the full matrix to `as.parm.names` and indicate that only the upper-triangular will be used via the `uppertri` argument. For example, `as.parm.names(list(beta=rep(0,J), uppertri=c(0,1))` creates parameter names for a vector of β parameters of length J and an upper-triangular matrix $\mathbf{U}$ of dimension K.

Numerous examples may be found in the accompanying "Examples" vignette.

Value

This function returns a vector of parameter names.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`IterativeQuadrature` `LaplaceApproximation`, `LaplacesDemon`, `PMC`, and `VariationalBayes`.

Examples

```
library(LaplacesDemon)
N <- 100
J <- 5
y <- rnorm(N,0,1)
X <- matrix(runif(N*J,-2,2),N,J)
S <- diag(J)
T <- diag(2)
mon.names <- c("LP","sigma")
parm.names <- as.parm.names(list(log.sigma=0, beta=rep(0,J), S=diag(J),
     T=diag(2)), uppertri=c(0,0,0,1))
MyData <- list(J=J, N=N, S=S, T=T, X=X, mon.names=mon.names,
     parm.names=parm.names, y=y)
MyData
```

---

as.ppc *As Posterior Predictive Check*

Description

This function converts an object of class `demonoid.val` to an object of class `demonoid.ppc`.

Usage

```
as.ppc(x, set=3)
```

Arguments

`x`	This is an object of class `demonoid.val`.
`set`	This is an integer that indicates which list component is to be used. When `set=1`, the modeled data set is used. When `set=2`, the validation data set is used. When `set=3`, both data sets are used.

Details

After using the `Validate` function for holdout validation, it is often suggested to perform posterior predictive checks. The `as.ppc` function converts the output object of `Validate`, which is an object of class `demonoid.val`, to an object of class `demonoid.ppc`. The returned object is the same as if it were created with the `predict.demonoid` function, rather than the `Validate` function.

After this conversion, the user may use posterior predictive checks, as usual, with the `summary.demonoid.ppc` function.

Value

The returned object is an object of class `demonoid.ppc`.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`predict.demonoid`, `summary.demonoid.ppc`, and `Validate`.

`BayesFactor` *Bayes Factor*

Description

This function calculates Bayes factors for two or more fitted objects of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb` that were estimated respectively with the `LaplacesDemon`, `IterativeQuadrature`, `LaplaceApproximation`, `PMC`, or `VariationalBayes` functions, and indicates the strength of evidence in favor of the hypothesis (that each model, $\mathcal{M}_i$, is better than another model, $\mathcal{M}_j$).

Usage

```
BayesFactor(x)
```

Arguments

`x`	This is a list of two or more fitted objects of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb`. The components are named in order beginning with model 1, `M1`, and k models are usually represented as $\mathcal{M}_1, \ldots, \mathcal{M}_k$.

Details

Introduced by Harold Jeffreys, a 'Bayes factor' is a Bayesian alternative to frequentist hypothesis testing that is most often used for the comparison of multiple models by hypothesis testing, usually to determine which model better fits the data (Jeffreys, 1961). Bayes factors are notoriously difficult to compute, and the Bayes factor is only defined when the marginal density of $\mathbf{y}$ under each model is proper (see `is.proper`). However, the Bayes factor is easy to approximate with the Laplace-Metropolis estimator (Lewis and Raftery, 1997) and other methods of approximating the logarithm of the marginal likelihood (for more information, see `LML`).

Hypothesis testing with Bayes factors is more robust than frequentist hypothesis testing, since the Bayesian form avoids model selection bias, evaluates evidence in favor of the null hypothesis, includes model uncertainty, and allows non-nested models to be compared (though of course the model must have the same dependent variable). Also, frequentist significance tests become biased in favor of rejecting the null hypothesis with sufficiently large sample size.

The Bayes factor for comparing two models may be approximated as the ratio of the marginal likelihood of the data in model 1 and model 2. Formally, the Bayes factor in this case is

$$B = \frac{p(\mathbf{y}|\mathcal{M}_1)}{p(\mathbf{y}|\mathcal{M}_2)} = \frac{\int p(\mathbf{y}|\Theta_1, \mathcal{M}_1)p(\Theta_1|\mathcal{M}_1)d\Theta_1}{\int p(\mathbf{y}|\Theta_2, \mathcal{M}_2)p(\Theta_2|\mathcal{M}_2)d\Theta_2}$$

where $p(\mathbf{y}|\mathcal{M}_1)$ is the marginal likelihood of the data in model 1.

The `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, and `VariationalBayes` functions each return the `LML`, the approximate logarithm of the marginal likelihood of the data, in each fitted object of class `iterquad`, `laplace`, `demonoid`, `pmc`, or `vb`. The `BayesFactor` function calculates matrix `B`, a matrix of Bayes factors, where each element of matrix `B` is a comparison of two models. Each Bayes factor is calculated as the exponentiated difference of `LML` of model 1 ($\mathcal{M}_1$) and `LML` of model 2 ($\mathcal{M}_2$), and the hypothesis for each element of matrix `B` is that the model associated with the row is greater than the model associated with the column. For example, element `B[3,2]` is the Bayes factor that model 3 is greater than model 2. The 'Strength of Evidence' aids in the interpretation (Jeffreys, 1961).

A table for the interpretation of the strength of evidence for Bayes factors is available at https://web.archive.org/web/20150214194051/http://www.bayesian-inference.com/bayesfactors.

Each Bayes factor, `B`, is the posterior odds in favor of the hypothesis divided by the prior odds in favor of the hypothesis, where the hypothesis is usually $\mathcal{M}_1 > \mathcal{M}_2$. For example, when `B[3,2]=2`, the data favor $\mathcal{M}_3$ over $\mathcal{M}_2$ with 2:1 odds.

It is also popular to consider the natural logarithm of the Bayes factor. The scale of the logged Bayes factor is the same above and below one, which is more appropriate for visual comparisons. For example, when comparing two Bayes factors at 0.5 and 2, the logarithm of these Bayes factors is -0.69 and 0.69.

Gelman finds Bayes factors generally to be irrelevant, because they compute the relative probabilities of the models conditional on one of them being true. Gelman prefers approaches that measure the distance of the data to each of the approximate models (Gelman et al., 2004, p. 180), such as with posterior predictive checks (see the `predict.iterquad` function regarding iterative quadrature, `predict.laplace` function in the context of Laplace Approximation, `predict.demonoid` function in the context of MCMC, `predict.pmc` function in the context of PMC, or `predict.vb` function in the context of Variational Bayes). Kass et al. (1995) asserts this can be done without assuming one model is the true model.

Value

`BayesFactor` returns an object of class `bayesfactor` that is a list with the following components:

`B` This is a matrix of Bayes factors.

`Hypothesis` This is the hypothesis, and is stated as 'row > column', indicating that the model associated with the row of an element in matrix `B` is greater than the model associated with the column of that element.

`Strength.of.Evidence` This is the strength of evidence in favor of the hypothesis.

`Posterior.Probability` This is a vector of the posterior probability of each model, given flat priors.

Author(s)

Statisticat, LLC.

References

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). "Bayesian Data Analysis, Texts in Statistical Science, 2nd ed.". Chapman and Hall, London.

Jeffreys, H. (1961). "Theory of Probability, Third Edition". Oxford University Press: Oxford, England.

Kass, R.E. and Raftery, A.E. (1995). "Bayes Factors". *Journal of the American Statistical Association*, 90(430), p. 773–795.

Lewis, S.M. and Raftery, A.E. (1997). "Estimating Bayes Factors via Posterior Simulation with the Laplace-Metropolis Estimator". *Journal of the American Statistical Association*, 92, p. 648–655.

See Also

`is.bayesfactor`, `is.proper`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `LML`, `PMC`, `predict.demonoid`, `predict.iterquad`, `predict.laplace`, `predict.pmc`, `predict.vb`, and `VariationalBayes`.

Examples

```
# The following example fits a model as Fit1, then adds a predictor, and
# fits another model, Fit2. The two models are compared with Bayes
# factors.

library(LaplacesDemon)

#############################  Demon Data  ##############################
data(demonsnacks)
J <- 2
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,10]+1)))
X[,2] <- CenterScale(X[,2])

########################  Data List Preparation  ########################
```

```
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

############################  Model Specification  ############################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

############################  Initial Values  ############################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)

########################  Laplace Approximation  ########################
Fit1 <- LaplaceApproximation(Model, Initial.Values, Data=MyData,
     Iterations=10000)
Fit1

############################  Demon Data  ############################
data(demonsnacks)
J <- 3
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(demonsnacks[,c(7,8)]))
X[,2] <- CenterScale(X[,2])
X[,3] <- CenterScale(X[,3])

########################  Data List Preparation  ########################
mon.names <- c("sigma","mu[1]")
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
```

```
PGF <- function(Data) return(c(rnormv(Data$J,0,10), rhalfcauchy(1,5)))
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

############################  Initial Values  ############################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)

#######################  Laplace Approximation  ########################
Fit2 <- LaplaceApproximation(Model, Initial.Values, Data=MyData,
     Iterations=10000)
Fit2

############################  Bayes Factor  ############################
Model.list <- list(M1=Fit1, M2=Fit2)
BayesFactor(Model.list)
```

`BayesianBootstrap` *The Bayesian Bootstrap*

Description

This function performs the Bayesian bootstrap of Rubin (1981), returning either bootstrapped weights or statistics.

Usage

```
BayesianBootstrap(X, n=1000, Method="weights", Status=NULL)
```

Arguments

`X`	This is a vector or matrix of data. When a matrix is supplied, sampling is based on the first column.
`n`	This is the number of bootstrapped replications.
`Method`	When `Method="weights"` (which is the default), a matrix of row weights is returned. Otherwise, a function is accepted. The function specifies the statistic to be bootstrapped. The first argument of the function should be a matrix of data, and the second argument should be a vector of weights.
`Status`	This determines the periodicity of status messages. When `Status=100`, for example, a status message is displayed every 100 replications. Otherwise, `Status` defaults to `NULL`, and status messages are not displayed.

Details

The term, 'bootstrap', comes from the German novel *Adventures of Baron Munchausen* by Rudolph Raspe, in which the hero saves himself from drowning by pulling on his own bootstraps. The idea of the statistical bootstrap is to evaluate properties of an estimator through the empirical, rather than theoretical, CDF.

Rubin (1981) introduced the Bayesian bootstrap. In contrast to the frequentist bootstrap which simulates the sampling distribution of a statistic estimating a parameter, the Bayesian bootstrap simulates the posterior distribution.

The data, **X**, are assumed to be independent and identically distributed (IID), and to be a representative sample of the larger (bootstrapped) population. Given that the data has N rows in one bootstrap replication, the row weights are sampled from a Dirichlet distribution with all N concentration parameters equal to 1 (a uniform distribution over an open standard $N-1$ simplex). The distributions of a parameter inferred from considering many samples of weights are interpretable as posterior distributions on that parameter.

The Bayesian bootstrap is useful for estimating marginal posterior covariance and standard deviations for the posterior modes of LaplaceApproximation, especially when the model dimension (the number of parameters) is large enough that estimating the Hessian matrix of second partial derivatives is too computationally demanding.

Just as with the frequentist bootstrap, inappropriate use of the Bayesian bootstrap can lead to inappropriate inferences. The Bayesian bootstrap violates the likelihood principle, because the evaluation of a statistic of interest depends on data sets other than the observed data set. For more information on the likelihood principle, see https://web.archive.org/web/20150213002158/http://www.bayesian-inference.com/likelihood#likelihoodprinciple.

The BayesianBootstrap function has many uses, including creating test statistics on the population data given the observed data (supported here), imputation (with this variation: ABB), validation, and more.

Value

When Method="weights", this function returns a $N \times n$ matrix of weights, where the number of rows N is equal to the number of rows in X.

For statistics, a matrix or array is returned, depending on the number of dimensions. The replicates are indexed by row in a matrix or in the first dimension of the array.

Author(s)

Bogumil Kaminski, <bkamins@sgh.waw.pl> and Statisticat, LLC.

References

Rubin, D.B. (1981). "The Bayesian Bootstrap". *The Annals of Statistics*, 9(1), p. 130–134.

See Also

ABB, Hessian, LaplaceApproximation, and LaplacesDemon.

Examples

```
library(LaplacesDemon)

#Example 1: Samples
x <- 1:2
BB <- BayesianBootstrap(X=x, n=100, Method="weights"); BB
```

```
#Example 2: Mean, Univariate
x <- 1:2
BB <- BayesianBootstrap(X=x, n=100, Method=weighted.mean); BB

#Example 3: Mean, Multivariate
data(demonsnacks)
BB <- BayesianBootstrap(X=demonsnacks, n=100,
     Method=function(x,w) apply(x, 2, weighted.mean, w=w)); BB

#Example 4: Correlation
dye <- c(1.15, 1.70, 1.42, 1.38, 2.80, 4.70, 4.80, 1.41, 3.90)
efp <- c(1.38, 1.72, 1.59, 1.47, 1.66, 3.45, 3.87, 1.31, 3.75)
X <- matrix(c(dye,efp), length(dye), 2)
colnames(X) <- c("dye","efp")
BB <- BayesianBootstrap(X=X, n=100,
     Method=function(x,w) cov.wt(x, w, cor=TRUE)$cor); BB

#Example 5: Marginal Posterior Covariance
#The following example is commented out due to package build time.
#To run the following example, use the code from the examples in
#the LaplaceApproximation function for the data, model specification
#function, and initial values. Then perform the Laplace
#Approximation as below (with CovEst="Identity" and sir=FALSE) until
#convergence, set the latest initial values, then use the Bayesian
#bootstrap on the data, run the Laplace Approximation again to
#convergence, save the posterior modes, and repeat until S samples
#of the posterior modes are collected. Finally, calculate the
#parameter covariance or standard deviation.

#Fit <- LaplaceApproximation(Model, Initial.Values, Data=MyData,
#     Iterations=1000, Method="SPG",  CovEst="Identity", sir=FALSE)
#Initial.Values <- as.initial.values(Fit)
#S <- 100 #Number of bootstrapped sets of posterior modes (parameters)
#Z <- rbind(Fit$Summary1[,1]) #Bootstrapped parameters collected here
#N <- nrow(MyData$X) #Number of records
#MyData.B <- MyData
#for (s in 1:S) {
#     cat("\nIter:", s, "\n")
#     BB <- BayesianBootstrap(MyData$y, n=N)
#     z <- apply(BB, 2, function(x) sample.int(N, size=1, prob=x))
#     MyData.B$y <- MyData$y[z]
#     MyData.B$X <- MyData$X[z,]
#     Fit <- LaplaceApproximation(Model, Initial.Values, Data=MyData.B,
#          Iterations=1000, Method="SPG", CovEst="Identity", sir=FALSE)
#     Z <- rbind(Z, Fit$Summary1[,1])}
#cov(Z) #Bootstrapped marginal posterior covariance
#sqrt(diag(cov(Z))) #Bootstrapped marginal posterior standard deviations
```

BayesTheorem *Bayes' Theorem*

Description

Bayes' theorem shows the relation between two conditional probabilities that are the reverse of each other. This theorem is named after Reverend Thomas Bayes (1702-1761), and is also referred to as Bayes' law or Bayes' rule (Bayes and Price, 1763). Bayes' theorem expresses the conditional probability, or 'posterior probability', of an event A after B is observed in terms of the 'prior probability' of A, prior probability of B, and the conditional probability of B given A. Bayes' theorem is valid in all common interpretations of probability. This function provides one of several forms of calculations that are possible with Bayes' theorem.

Usage

```
BayesTheorem(PrA, PrBA)
```

Arguments

`PrA`	This required argument is the prior probability of A, or $\Pr(A)$.
`PrBA`	This required argument is the conditional probability of B given A or $\Pr(B\|A)$, and is known as the data, evidence, or likelihood.

Details

Bayes' theorem provides an expression for the conditional probability of A given B, which is equal to

$$\Pr(A|B) = \frac{\Pr(B|A)\Pr(A)}{\Pr(B)}$$

For example, suppose one asks the question: what is the probability of going to Hell, conditional on consorting (or given that a person consorts) with Laplace's Demon. By replacing A with $Hell$ and B with $Consort$, the question becomes

$$\Pr(\text{Hell}|\text{Consort}) = \frac{\Pr(\text{Consort}|\text{Hell})\Pr(\text{Hell})}{\Pr(\text{Consort})}$$

Note that a common fallacy is to assume that $\Pr(A|B) = \Pr(B|A)$, which is called the conditional probability fallacy.

Another way to state Bayes' theorem (and this is the form in the provided function) is

$$\Pr(A_i|B) = \frac{\Pr(B|A_i)\Pr(A_i)}{\Pr(B|A_i)\Pr(A_i) + \ldots + \Pr(B|A_n)\Pr(A_n)}$$

Let's examine our *burning* question, by replacing A_i with Hell or Heaven, and replacing B with Consort

- $\Pr(A_1) = \Pr(\text{Hell})$
- $\Pr(A_2) = \Pr(\text{Heaven})$
- $\Pr(B) = \Pr(\text{Consort})$

- $\Pr(A_1|B) = \Pr(\text{Hell}|\text{Consort})$
- $\Pr(A_2|B) = \Pr(\text{Heaven}|\text{Consort})$
- $\Pr(B|A_1) = \Pr(\text{Consort}|\text{Hell})$
- $\Pr(B|A_2) = \Pr(\text{Consort}|\text{Heaven})$

Laplace's Demon was conjured and asked for some data. He was glad to oblige.

- 6 people consorted out of 9 who went to Hell.
- 5 people consorted out of 7 who went to Heaven.
- 75% of the population goes to Hell.
- 25% of the population goes to Heaven.

Now, Bayes' theorem is applied to the data. Four pieces are worked out as follows

- $\Pr(\text{Consort}|\text{Hell}) = 6/9 = 0.666$
- $\Pr(\text{Consort}|\text{Heaven}) = 5/7 = 0.714$
- $\Pr(\text{Hell}) = 0.75$
- $\Pr(\text{Heaven}) = 0.25$

Finally, the desired conditional probability $\Pr(\text{Hell}|\text{Consort})$ is calculated using Bayes' theorem

- $\Pr(\text{Hell}|\text{Consort}) = \frac{0.666(0.75)}{0.666(0.75)+0.714(0.25)}$
- $\Pr(\text{Hell}|\text{Consort}) = 0.737$

The probability of someone consorting with Laplace's Demon and going to Hell is 73.7%, which is less than the prevalence of 75% in the population. According to these findings, consorting with Laplace's Demon does not increase the probability of going to Hell.

For an introduction to model-based Bayesian inference, see the accompanying vignette entitled "Bayesian Inference" or https://web.archive.org/web/20150206004608/http://www.bayesian-inference.com/bayesian.

Value

The `BayesTheorem` function returns the conditional probability of A given B, known in Bayesian inference as the posterior. The returned object is of class `bayestheorem`.

Author(s)

Statisticat, LLC.

References

Bayes, T. and Price, R. (1763). "An Essay Towards Solving a Problem in the Doctrine of Chances". By the late Rev. Mr. Bayes, communicated by Mr. Price, in a letter to John Canton, M.A. and F.R.S. *Philosophical Transactions of the Royal Statistical Society of London*, 53, p. 370–418.

See Also

IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes.

Examples

```
# Pr(Hell|Consort) =
PrA <- c(0.75,0.25)
PrBA <- c(6/9, 5/7)
BayesTheorem(PrA, PrBA)
```

BigData *Big Data*

Description

This function enables Bayesian inference with data that is too large for computer memory (RAM) with the simplest method: reading in batches of data (where each batch is a section of rows), applying a function to the batch, and combining the results.

Usage

```
BigData(file, nrow, ncol, size=1, Method="add", CPUs=1, Type="PSOCK",
FUN, ...)
```

Arguments

file	This required argument accepts a path and filename that must refer to a .csv file, and that must contain only a numeric matrix without a header, row names, or column names.
nrow	This required argument accepts a scalar integer that indicates the number of rows in the big data matrix.
ncol	This required argument accepts a scalar integer that indicates the number of columns in the big data matrix.
size	This argument accepts a scalar integer that specifies the number of rows of each batch. The last batch is not required to have the same number of rows as the other batches. The largest possible size, and therefore the fewest number of batches, should be preferred.
Method	This argument accepts a scalar string, defaults to "add", and alternatively accepts "rbind". When Method="rbind", the user-specified function FUN is applied to each batch, and results are combined together by rows. For example, if calculating $\mu = \mathbf{X}\beta$ in, say, 10 batches, then the output column vector μ is equal to the number of rows of the big data set.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.

Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".
FUN	This required argument accepts a user-specified function that will be performed on each batch. The first argument in the function must be the data.
...	Additional arguments are used within the user-specified function. Additional arguments often refer to parameters.

Details

Big data is defined loosely here as data that is too large for computer memory (RAM). The BigData function uses the split-apply-combine strategy with a big data set. The unmanageable big data set is split into smaller, manageable pieces (batches), a function is applied to each batch, and results are combined.

Each iteration, the BigData function opens a connection to a big data set and keeps the connection open while the scan function reads in each batch of data (elsewhere, batches are often referred to chunks). A user-specified function is applied to each batch of data, the results are combined together, the connection is closed, and the results are returned.

As an introductory example, suppose a statistician updates a linear regression model, but the design matrix $\mathbf{X}$ is too large for computer memory. Suppose the design matrix has 100 million rows, and the statistician specifies size=1e6. The statistician combines dependent variable $\mathbf{y}$ with design matrix $\mathbf{X}$. Each iteration in IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, or VariationalBayes, the BigData function sequentially reads in one million rows of the combined data $\mathbf{X}$, calculates expectation vector μ, and finally returns the sum of the log-likelihood. The sum of the log-likelihood is added together for all batches, and returned.

There are many limitations with this function.

This function is not fast, in the sense that the entire big data set is processed in batches, each iteration. With iterative methods, this may perform well, albeit slowly.

There are many functions that cannot be performed on batches, though most models in the Examples vignette may easily be updated with big data.

Large matrices of samples are unaddressed, only the data.

Although many (but not all) models may be estimated, many additional functions in this package will not work when applied after the model has updated. Instead, a batch or random sample of data (see the read.matrix function for sampling from big data) should be used in the usual way, in the Data argument, and the Model function coded in the usual way without the BigData function.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface (MPI) is used. Each call to BigData establishes and closes the parallelization, which is costly, and unfortunately results in copious output to the console. With small data sets, parallel processing may be slower, due to computer network communication. With larger data sets, the user should experience a faster run-time.

There have been several alternative approaches suggested for big data.

Huang and Gelman (2005) propose that the user creates batches by sampling from big data, updating a separate Bayesian model on each batch, and combining the results into a consensus posterior. This many-mini-model approach may be faster when feasible, because multiple models may be updated

in parallel, say one per CPU. Such results will work with all functions in this package. With the many-mini-model approach, several methods are proposed for combining posterior samples from batch-level models, such as by using a normal approximation, updating from prior to posterior sequentially (the posterior from the last batch becomes the prior of the next batch), sample from the full posterior via importance sampling from the batched posteriors, and more.

Scott et al. (2013) propose a method that they call Consensus Monte Carlo, which consists of breaking the data down into chunks, calling each chunk a shard, and use a many-mini-model approach as well, but propose their own method of weighting the posteriors back together.

Balakrishnan and Madigan (2006) introduced a Sequential Monte Carlo (SMC) sampler, a refinement of an earlier proposal, that was designed for big data. It makes one pass through the massive data set, after an initial MCMC estimation on a small sample. Each particle is updated for each record, resulting in numerous evaluations per record.

Welling and Teh (2011) proposed a new class of MCMC sampler in which only a random sample of big data is used each iteration. The stochastic gradient Langevin dynamics (SGLD) algorithm is available in the `LaplacesDemon` function.

An important alternative to consider is using the `ff` package, where "ff" stands for fast access file. The `ff` package has been tested successfully with updating a model in `LaplacesDemon`. Once the big data set, say **X**, is an object of class `ff_matrix`, simply include it in the list of data as usual, and modify the `Model` specification function appropriately. For example, change `mu <- tcrossprod(X, t(beta))` to `mu <- tcrossprod(X[], t(beta))`. The `ff` package is not included as a dependency in the `LaplacesDemon` package, so it must be installed and activated.

Value

The `BigData` function returns output that is the result of performing a user-specified function on batches of big data. Output is a matrix, and may have one or more column vectors.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Balakrishnan, S. and Madigan, D. (2006). "A One-Pass Sequential Monte Carlo Method for Bayesian Analysis of Massive Datasets". *Bayesian Analysis*, 1(2), p. 345–362.

Huang, Z. and Gelman, A. (2005) "Sampling for Bayesian Computation with Large Datasets". *SSRN eLibrary*.

Scott, S.L., Blocker, A.W. and Bonassi, F.V. (2013). "Bayes and Big Data: The Consensus Monte Carlo Algorithm". In *Bayes 250*.

Welling, M. and Teh, Y.W. (2011). "Bayesian Learning via Stochastic Gradient Langevin Dynamics". *Proceedings of the 28th International Conference on Machine Learning (ICML)*, p. 681–688.

See Also

`IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `LaplacesDemon.RAM`, `PMC`, `PMC.RAM`, `read.matrix`, and `VariationalBayes`.

Examples

```
### Below is an example of a linear regression model specification
### function in which BigData reads in a batch of 1,000 records of
### Data$N records from a data set that is too large to fully open
### in memory. The example simulates on 10,000 records, which is
### not big data; it's just a toy example. The data set is file X.csv,
### and the first column of matrix X is the dependent variable y. The
### user supplies a function to BigData along with parameters beta and
### sigma. When each batch of 1,000 records is read in,
### mu = XB is calculated, and then the LL is calculated as
### y ~ N(mu, sigma^2). These results are added together from all
### batches, and returned as LL.

library(LaplacesDemon)
N <- 10000
J <- 10 #Number of predictors, including the intercept
X <- matrix(1,N,J)
for (j in 2:J) {X[,j] <- rnorm(N,runif(1,-3,3),runif(1,0.1,1))}
beta.orig <- runif(J,-3,3)
e <- rnorm(N,0,0.1)
y <- as.vector(tcrossprod(beta.orig, X) + e)
mon.names <- c("LP","sigma")
parm.names <- as.parm.names(list(beta=rep(0,J), log.sigma=0))
PGF <- function(Data) return(c(rnormv(Data$J,0,0.01),
     log(rhalfcauchy(1,1))))
MyData <- list(J=J, PGF=PGF, N=N, mon.names=mon.names,
     parm.names=parm.names) #Notice that X and y are not included here
filename <- tempfile("X.csv")
write.table(cbind(y,X), filename, sep=",", row.names=FALSE,
  col.names=FALSE)

Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[1:Data$J]
     sigma <- exp(parm[Data$J+1])
     ### Log(Prior Densities)
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     LL <- BigData(file=filename, nrow=Data$N, ncol=Data$J+1, size=1000,
          Method="add", CPUs=1, Type="PSOCK",
          FUN=function(x, beta, sigma) sum(dnorm(x[,1], tcrossprod(x[,-1],
               t(beta)), sigma, log=TRUE)), beta, sigma)
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=c(LP,sigma),
          yhat=0,#rnorm(length(mu), mu, sigma),
          parm=parm)
     return(Modelout)
     }
```

```
### From here, the user may update the model as usual.
```

`Blocks` *Blocks*

Description

The `Blocks` function returns a list of N blocks of parameters, for use with some MCMC algorithms in the `LaplacesDemon` function. Blocks may be created either sequentially, or from a hierarchical clustering of the posterior correlation matrix.

Usage

```
Blocks(Initial.Values, N, PostCor=NULL)
```

Arguments

`Initial.Values`	This required argument is a vector of initial values.
`N`	This optional argument indicates the desired number of blocks. If omitted, then the truncated square root of the number of initial values is used. If a posterior correlation matrix is supplied to `PostCor`, then `N` may be a scalar, or have length two. If `N` has length two, then the first element indicates the minimum number of blocks, and the second element indicates the maximum number of blocks, and the number of blocks is the maximum of the mean silhouette width for each hierarchical cluster solution.
`PostCor`	This optional argument defaults to `NULL`, in which case sequential blocking is performed. If a posterior correlation matrix is supplied, then blocks are created based on hierarchical clustering.

Details

Usually, there is more than one target distribution in MCMC, in which case it must be determined whether it is best to sample from target distributions individually, in groups, or all at once. Blockwise sampling (also called block updating) refers to splitting a multivariate vector into groups called blocks, and each block is sampled separately. A block may contain one or more parameters.

Parameters are usually grouped into blocks such that parameters within a block are as correlated as possible, and parameters between blocks are as independent as possible. This strategy retains as much of the parameter correlation as possible for blockwise sampling, as opposed to component-wise sampling where parameter correlation is ignored. The `PosteriorChecks` function can be used on the output of previous runs to find highly correlated parameters. See examples below.

Advantages of blockwise sampling are that a different MCMC algorithm may be used for each block (or parameter, for that matter), creating a more specialized approach (though different algorithms by block are not supported here), the acceptance of a newly proposed state is likely to be higher than sampling from all target distributions at once in high dimensions, and large proposal covariance matrices can be reduced in size, which is most helpful again in high dimensions.

Disadvantages of blockwise sampling are that correlations probably exist between parameters between blocks, and each block is updated while holding the other blocks constant, ignoring these correlations of parameters between blocks. Without simultaneously taking everything into account, the algorithm may converge slowly or never arrive at the proper solution. However, there are instances when it may be best when everything is not taken into account at once, such as in state-space models. Also, as the number of blocks increases, more computation is required, which slows the algorithm. In general, blockwise sampling allows a more specialized approach at the expense of accuracy, generalization, and speed. Blockwise sampling is offered in the following algorithms: Adaptive-Mixture Metropolis (AMM), Adaptive Metropolis-within-Gibbs (AMWG), Automated Factor Slice Sampler (AFSS), Elliptical Slice Sampler (ESS), Hit-And-Run Metropolis (HARM), Metropolis-within-Gibbs (MWG), Random-Walk Metropolis (RWM), Robust Adaptive Metropolis (RAM), Slice Sampler (Slice), and the Univariate Eigenvector Slice Sampler (UESS).

Large-dimensional models often require blockwise sampling. For example, with thousands of parameters, a componentwise algorithm must evaluate the model specification function once per parameter per iteration, resulting in an algorithm that may take longer than is acceptable to produce samples. Algorithms that require derivatives, such as the family of Hamiltonian Monte Carlo (HMC), require even more evaluations of the model specification function per iteration, and quickly become too costly in large dimensions. Finally, algorithms with multivariate proposals often have difficulty producing an accepted proposal in large-dimensional models. The most practical solution is to group parameters into N blocks, and each iteration the algorithm evaluates the model specification function N times, each with a reduced set of parameters.

The `Blocks` function performs either a sequential assignment of parameters to blocks when posterior correlation is not supplied, or uses hierarchical clustering to create blocks based on posterior correlation. If posterior correlation is supplied, then the user may specify a range of the number of blocks to consider, and the optimal number of blocks is considered to be the maximum of the mean silhouette width of each hierarchical clustering. Silhouette width is calculated as per the `cluster` package. Hierarchical clustering is performed on the distance matrix calculated from the dissimilarity matrix (1 - abs(PostCor)) of the posterior correlation matrix. With sequential assignment, the number of parameters per block is approximately equal. With hierarchical clustering, the number of parameters per block may vary widely. Creating blocks from hierarchical clustering performs well in practice, though there are many alternative methods the user may consider outside of this function, such as using factor analysis, model-based clustering, or other methods.

Aside from sequentially-assigned blocks, or blocks based on posterior correlation, it is also common to group parameters with similar uses, such as putting regression effects parameters into one block, and autocorrelation parameters into another block. Another popular way to group parameters into blocks is by time-period for some time-series models. These alternative blocking strategies are unsupported in the `Blocks` function, and best left to user discretion.

Some MCMC algorithms that accept blocked parameters also require blocked variance-covariance matrices. The `Blocks` function does not return these matrices, because it may not be necessary, or when it is, the user may prefer identity matrices, scaled identity matrices, or matrices with explicitly-defined elements.

If the user is looking for a place to begin with blockwise sampling, then the recommended, default approach (when blocked parameters by time-period are not desired in a time-series) is to begin with a trial run of the adaptive, unblocked HARM algorithm (since covariance matrices are not required) for the purposes of obtaining a posterior correlation matrix. Next, create blocks with the `Blocks` function based on the posterior correlation matrix obtained from the trial run. Finally, run the desired, blocked algorithm with the newly created blocks (and possibly user-specified covariance

matrices), beginning where the trial run ended.

If hierarchical clustering is used, then it is important to note that hierarchical clustering has no idea that the user intends to perform blockwise sampling in MCMC. If hierarchical clustering returns numerous small blocks, then the user may consider combining some or all of those blocks. For example, if several 1-parameter blocks are returned, then blockwise sampling will equal componentwise sampling for those blocks, which will iterate slower. Conversely, if hierarchical clustering returns one or more big blocks, each with enough parameters that multivariate sampling will have difficulty getting an accepted proposal, or an accepted proposal that moves more than a small amount, then the user may consider subdividing these big blocks into smaller, more manageable blocks, though with the understanding that more posterior correlation is unaccounted for.

Value

The `Blocks` function returns an object of class `blocks`, which is a list. Each component of the list is a block of parameters, and parameters are indicated by their position in the initial values vector.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`LaplacesDemon` and `PosteriorChecks`.

Examples

```
library(LaplacesDemon)

### Create the default number of sequentially assigned blocks:
Initial.Values <- rep(0,1000)
MyBlocks <- Blocks(Initial.Values)
MyBlocks

### Or, a pre-specified number of sequentially assigned blocks:
#Initial.Values <- rep(0,1000)
#MyBlocks <- Blocks(Initial.Values, N=20)

### If scaled diagonal covariance matrices are desired:
#VarCov <- list()
#for (i in 1:length(MyBlocks))
#  VarCov[[i]] <- diag(length(MyBlocks[[i]]))*2.38^2/length(MyBlocks[[i]])

### Or, determine the number of blocks in the range of 2 to 50 from
### hierarchical clustering on the posterior correlation matrix of an
### object, say called Fit, output from LaplacesDemon:
#MyBlocks <- Blocks(Initial.Values, N=c(2,50),
#  PostCor=cor(Fit$Posterior1))
#lapply(MyBlocks, length) #See the number of parameters per block

### Or, create a pre-specified number of blocks from hierarchical
### clustering on the posterior correlation matrix of an object,
```

```
### say called Fit, output from LaplacesDemon:
#MyBlocks <- Blocks(Initial.Values, N=20, PostCor=cor(Fit$Posterior1))

### Posterior correlation from a previous trial run could be obtained
### with either method below (though cor() will be fastest because
### additional checks are not calculated for the parameters):
#rho <- cor(Fit$Posterior1)
#rho <- PosteriorChecks(Fit)$Posterior.Correlation
```

`BMK.Diagnostic` *BMK Convergence Diagnostic*

Description

Given a matrix of posterior samples from MCMC, the `BMK.Diagnostic` function calculates Hellinger distances between consecutive batches for each chain. This is useful for monitoring convergence of MCMC chains.

Usage

```
BMK.Diagnostic(X, batches=10)
```

Arguments

`X`	This required argument accepts a matrix of posterior samples or an object of class `demonoid`, in which case it uses the posterior samples in `X$Posterior1`.
`batches`	This is the number of batches on which the convergence diagnostic will be calculated. The `batches` argument defaults to 10.

Details

Hellinger distance is used to quantify dissimilarity between two probability distributions. It is based on the Hellinger integral, introduced by Hellinger (1909). Traditionally, Hellinger distance is bound to the interval [0,1], though another popular form occurs in the interval $[0,\sqrt{2}]$. A higher value of Hellinger distance is associated with more dissimilarity between the distributions.

Convergence is assumed when Hellinger distances are below a threshold, indicating that posterior samples are similar between consecutive batches. If all Hellinger distances beyond a given batch of samples is below the threshold, then `burnin` is suggested to occur immediately before the first batch of satisfactory Hellinger distances.

As an aid to interpretation, consider a matrix of 1,000 posterior samples from three chains: `beta[1]`, `beta[2]`, and `beta[3]`. With 10 batches, the column names are: 100, 200, ..., 900. A Hellinger distance for the chain `beta[1]` at 100 is the Hellinger distance between two batches: samples 1-100, and samples 101:200.

A benefit to using `BMK.Diagnostic` is that the resulting Hellinger distances may easily be plotted with the `plotMatrix` function, allowing the user to see quickly which consecutive batches of which chains were dissimilar. This makes it easier to find problematic chains.

The BMK.Diagnostic is calculated automatically in the LaplacesDemon function, and is one of the criteria in the Consort function regarding the recommendation of when to stop updating the Markov chain Monte Carlo (MCMC) sampler in LaplacesDemon.

For more information on the related topics of burn-in and stationarity, see the burnin and is.stationary functions, and the accompanying vignettes.

Value

The BMK.Diagnostic function returns an object of class bmk that is a $J \times B$ matrix of Hellinger distances between consecutive batches for J parameters of posterior samples. The number of columns, B is equal to the number of batches minus one.

The BMK.Diagnostic function is similar to the bmkconverge function in package BMK.

References

Boone, E.L., Merrick, J.R. and Krachey, M.J. (2013). "A Hellinger Distance Approach to MCMC Diagnostics". *Journal of Statistical Computation and Simulation*, in press.

Hellinger, E. (1909). "Neue Begrundung der Theorie quadratischer Formen von unendlichvielen Veranderlichen" (in German). *Journal fur die reine und angewandte Mathematik*, 136, p. 210–271.

See Also

burnin, Consort, is.stationary, and LaplacesDemon.

Examples

```
library(LaplacesDemon)
N <- 1000 #Number of posterior samples
J <- 10 #Number of parameters
Theta <- matrix(runif(N*J),N,J)
colnames(Theta) <- paste("beta[", 1:J, "]", sep="")
for (i in 2:N) {Theta[i,1] <- Theta[i-1,1] + rnorm(1)}
HD <- BMK.Diagnostic(Theta, batches=10)
plot(HD, title="Hellinger distance between batches")
```

---

burnin *Burn-in*

Description

The burnin function estimates the duration of burn-in in iterations for one or more Markov chains. "Burn-in" refers to the initial portion of a Markov chain that is not stationary and is still affected by its initial value.

Usage

```
burnin(x, method="BMK")
```

Arguments

x	This is a vector or matrix of posterior samples for which a the number of burn-in iterations will be estimated.
method	This argument defaults to "BMK", in which case stationarity is estimated with the BMK.Diagnostic function. Alternatively, the Geweke.Diagnostic function may be used when method="Geweke" or the KS.Diagnostic function may be used when method="KS".

Details

Burn-in is a colloquial term for the initial iterations in a Markov chain prior to its convergence to the target distribution. During burn-in, the chain is not considered to have "forgotten" its initial value.

Burn-in is not a theoretical part of MCMC, but its use is the norm because of the need to limit the number of posterior samples due to computer memory. If burn-in were retained rather than discarded, then more posterior samples would have to be retained. If a Markov chain starts anywhere close to the center of its target distribution, then burn-in iterations do not need to be discarded.

In the LaplacesDemon function, stationarity is estimated with the BMK.Diagnostic function on all thinned posterior samples of each chain, beginning at cumulative 10% intervals relative to the total number of samples, and the lowest number in which all chains are stationary is considered the burn-in.

The term, "burn-in", originated in electronics regarding the initial testing of component failure at the factory to eliminate initial failures (Geyer, 2011). Although "burn-in' has been the standard term for decades, some are referring to these as "warm-up" iterations.

Value

The burnin function returns a vector equal in length to the number of MCMC chains in x, and each element indicates the maximum iteration in burn-in.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Geyer, C.J. (2011). "Introduction to Markov Chain Monte Carlo". In S Brooks, A Gelman, G Jones, and M Xiao-Li (eds.), "Handbook of Markov Chain Monte Carlo", p. 3–48. Chapman and Hall, Boca Raton, FL.

See Also

BMK.Diagnostic, deburn, Geweke.Diagnostic, KS.Diagnostic, and LaplacesDemon.

Examples

```
library(LaplacesDemon)
x <- rnorm(1000)
burnin(x)
```

`caterpillar.plot` *Caterpillar Plot*

Description

A caterpillar plot is a horizontal plot of 3 quantiles of selected distributions. This may be used to produce a caterpillar plot of posterior samples (parameters and monitored variables) from an object either of class `demonoid`, `demonoid.hpc`, `iterquad`, `laplace`, `pmc`, `vb`, or a matrix.

Usage

```
caterpillar.plot(x, Parms=NULL, Title=NULL)
```

Arguments

`x`	This required argument is an object of class `demonoid`, codedemonoid.hpc, `iterquad`, `laplace`, `pmc`, `vb`, or a $S \times J$ matrix of S samples and J variables. For an object of class `demonoid`, the distributions of the stationary posterior summary (`Summary2`) will be attempted first, and if missing, then the parameters of all posterior samples (`Summary1`) will be plotted. For an object of class `demonoid.hpc`, stationarity may differ by chain, so all posterior samples (`Summary1`) are used. For an object of class `laplace` or `vb`, the distributions in the posterior summary, `Summary`, are plotted according to the posterior draws, sampled with sampling importance resampling in the `SIR` function. When a generic matrix is supplied, unimodal 95% HPD intervals are estimated with the `p.interval` function.
`Parms`	This argument accepts a vector of quoted strings to be matched for selecting parameters and monitored variables for plotting (though all parameters are selected when a generic matrix is supplied). This argument defaults to `NULL` and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the `grep` function. For example, if the user specifies `Parms=c("eta", "tau")`, and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string `eta` is within `beta`. Since `grep` is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
`Title`	This argument accepts a title for the plot.

Details

Caterpillar plots are popular plots in Bayesian inference for summarizing the quantiles of posterior samples. A caterpillar plot is similar to a horizontal boxplot, though without quartiles, making it easier for the user to study more distributions in a single plot. The following quantiles are plotted as a line for each parameter: 0.025 and 0.975, with the exception of a generic matrix, where unimodal 95% HPD intervals are estimated (for more information, see `p.interval`). A vertical, gray line is included at zero. For all but class `demonoid.hpc`, the median appears as a black dot, and the quantile line is black. For class `demonoid.hpc`, the color of the median and quantile line differs by chain; the first chain is black and additional chains appear beneath.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, p.interval, SIR, and VariationalBayes.

Examples

```
#An example is provided in the LaplacesDemon function.
```

CenterScale *Centering and Scaling*

Description

This function either centers and scales a continuous variable and provides options for binary variables, or returns an untransformed variable from a centered and scaled variable.

Usage

```
CenterScale(x, Binary="none", Inverse=FALSE, mu, sigma, Range, Min)
```

Arguments

x	This is a vector to be centered and scaled, or to be untransformed if Inverse=TRUE.
Binary	This argument indicates how binary variables will be treated, and defaults to "none", which keeps the original scale, or transforms the variable to the 0-1 range, if not already there. With "center", it will center the binary variable by subtracting the mean. With "center0", it centers the binary variable at zero, recoding a 0 to -0.5, and a 1 to 0.5. Finally, "centerscale" will center and scale the binary variable, subtracting the mean and dividing by two standard deviations.
Inverse	Logical. If TRUE, then a centered and scaled variable x will be transformed to its original, un-centered and un-scaled state. This defaults to FALSE.
mu, sigma, Range, Min	These arguments are required only when Inverse=TRUE, where mu is the mean, sigma is the standard deviation, Range is the range, and Min is the minimum of the original x. Range and Min are used only when Binary="none" or Binary="center0".

Details

Gelman (2008) recommends centering and scaling continuous predictors to facilitate MCMC convergence and enable comparisons between coefficients of centered and scaled continuous predictors with coefficients of untransformed binary predictors. A continuous predictor is centered and scaled as follows: `x.cs <- (x - mean(x)) / (2*sd(x))`. This is an improvement over the usual practice of standardizing predictors, which is `x.z <- (x - mean(x)) / sd(x)`, where coefficients cannot be validly compared between binary and continuous predictors.

In MCMC, such as in LaplacesDemon, a centered and scaled predictor often results in a higher effective sample size (ESS), and therefore the chain mixes better. Centering and scaling is a method of re-parameterization to improve mixing.

Griffin and Brown (2013) also assert that the user may not want to scale predictors that are measured on the same scale, since scaling in this case may increase noisy, low signals. In this case, centering (without scaling) is recommended. To center a predictor, subtract its mean.

Value

The `CenterScale` function returns a centered and scaled vector, or the untransformed vector.

References

Gelman, A. (2008). "Scaling Regression Inputs by Dividing by Two Standard Devations". *Statistics in Medicine*, 27, p. 2865–2873.

Griffin, J.E. and Brown, P.J. (2013) "Some Priors for Sparse Regression Modelling". *Bayesian Analysis*, 8(3), p. 691–702.

See Also

ESS, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, and PMC.

Examples

```
### See the LaplacesDemon function for an example in use.
library(LaplacesDemon)
x <- rnorm(100,10,1)
x.cs <- CenterScale(x)
x.orig <- CenterScale(x.cs, Inverse=TRUE, mu=mean(x), sigma=sd(x))
```

Combine *Combine Demonoid Objects*

Description

This function combines objects of class `demonoid`.

Usage

```
Combine(x, Data, Thinning=1)
```

Arguments

x	This is a list of objects of class demonoid, and this list may be an object of class demonoid.hpc.
Data	This is the data, and must be identical to the data used to create the demonoid objects with LaplacesDemon.
Thinning	This is the amount of thinning to apply to the posterior samples after appending them together. Thinning defaults to 1, in which case all samples are retained. For example, in the case of, say, Thinning=10, then only every 10th sample would be retained. When combining parallel chains, Thinning is often left to its default. When combining consecutive updates, Thinning is usually applied, with the value equal to the number of objects of class demonoid. For more information on thinning, see the Thin function.

Details

The purpose of the Combine function is to enable a user to combine objects of class demonoid for one of three reasons. First, parallel chains from LaplacesDemon.hpc may be combined after convergence is assessed with Gelman.Diagnostic. Second, consecutive updates of single chains from LaplacesDemon or parallel chains from LaplacesDemon.hpc may be combined when the computer has insufficient random-access memory (RAM) for the user to update once with enough iterations. Third, consecutive single-chain or parallel-chain updates may be combined when it seems that the logarithm of the joint posterior distribution, LP, seems to be oscillating up and down, which is described in more detail below.

The most common use regards the combination of parallel chains output from LaplacesDemon.hpc. Typically, a user with parallel chains examines them graphically with the caterpillar.plot and plot (actually, plot.demonoid) functions, and assesses convergence with the Gelman.Diagnostic function. Thereafter, the parallel chain output in the object of class demonoid.hpc should be combined into a single object of class demonoid, before doing posterior predictive checks and making inferences. In this case, the Thinning argument usually is recommended to remain at its default.

It is also common with a high-dimensional model (a model with a large number of parameters) to need more posterior samples than allowed by the random-access memory (RAM) of the computer. In this case, it is best to use the LaplacesDemon.RAM function to estimate the amount of RAM that a given model will require with a given number of iterations, and then update LaplacesDemon almost as much as RAM allows, and save the output object of class demonoid. Then, the user is advised to continue onward with a consecutive update (after using as.initial.values and anything else appropriate to prepare for the consecutive update). Suppose a user desires to update a gigantic model with thousands of parameters, and with the aid of LaplacesDemon.RAM, estimates that they can safely update only 100,000 iterations, and that 150,000 iterations would exceed RAM and crash the computer. The patient user can update several consecutive models, each with retaining only 1,000 thinned posterior samples, and combine them later with the Combine function, by placing multiple objects into a list, as described below. In this way, it is possible for a user to update models that otherwise far exceed computer RAM.

Less commonly, multiple updates of single-chain objects should be combined into a single object of class demonoid. This is most useful in complicated models that are run for large numbers of iterations, where it may be suspected that stationarity has been achieved, but that thinning is insufficient, and the samples may be combined and thinned. If followed, then these suggestions may continue seemingly to infinity, and the unnormalized logarithm of the joint posterior density, LP, may seem

to oscillate, sometimes improving and getting higher, and getting lower during other updates. For this purpose, the prior covariance matrix of the last model is retained (rather than combining them). This may be an unpleasant surprise for combining parallel updates, so be aware of it.

In these cases, which usually involve complicated models with high autocorrelation in the chains, the user may opt to use parallel processing with the LaplacesDemon.hpc function, or may use the LaplacesDemon function as follows. The user should save (meaning, not overwrite) each object of class demonoid, place multiple objects into a list, and use the Combine function to combine these objects.

For example, suppose a user names the object Fit, as in the LaplacesDemon example. Now, rather than overwriting object Fit, object Fit is renamed, after updating a million iterations, to Fit1. As suggested by Consort, another million iterations are used, but now to create object Fit2. Further suppose this user specified Thinning=1000 in LaplacesDemon, meaning that the million iterations are thinned by 1,000, so only 1,000 iterations are retained in each object, Fit1 and Fit2. In this case, Combine combines the information in Fit1 and Fit2, and returns an object the user names Fit3. Fit3 has only 1,000 iterations, which is the result of appending the iterations in Fit1 and Fit2, and thinning by 2. If 2,000,000 iterations were updated from the beginning, and were thinned by 2,000, then the same information exists now in Fit3. The Consort function can now be applied to Fit3, to see if stationarity is found. If not, then more objects of class demonoid can be collected and combined.

Value

This function returns an object of class demonoid. For more information on an object of class demonoid, see the LaplacesDemon function.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

caterpillar.plot, Gelman.Diagnostic, LaplacesDemon, LaplacesDemon.hpc, and Thin.

---

cond.plot *Conditional Plots*

Description

This function provides several styles of conditional plots with base graphics.

Usage

```
cond.plot(x, y, z, Style="smoothscatter")
```

Arguments

x	This required argument accepts a numeric vector.
y	This argument accepts a numeric vector, and is only used with some styles.
z	This required argument accepts a discrete vector.
Style	This argument specifies the style of plot, and accepts "boxplot", "densover" (density overlay), "hist", "scatter", or "smoothscatter".

Details

The cond.plot function provides simple conditional plots with base graphics. All plot styles are conditional upon z. Up to nine conditional plots are produced in a panel.

Plots include:

boxplot: y ~ x | z densover: f(x | z) hist: x | z scatter: x, y | z smoothscatter: x, y | z

The cond.plot function is not intended to try to compete with some of the better graphics packages, but merely to provide simple functionality.

Value

Conditional plots are returned.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

joint.density.plot and joint.pr.plot.

Examples

```
library(LaplacesDemon)
x <- rnorm(1000)
y <- runif(1000)
z <- rcat(1000, rep(1/4,4))
cond.plot(x, y, z, Style="smoothscatter")
```

Consort *Consort with Laplace's Demon*

Description

This may be used to consort with Laplace's Demon regarding an object of class demonoid. Laplace's Demon will offer suggestions.

Usage

```
Consort(object)
```

Arguments

`object`	This required argument is an object of class `demonoid`. For more information, see the `LaplacesDemon` function.

Details

First, `Consort` calls `print.demonoid`, which prints most of the components to the screen from the supplied object of class `demonoid`.

Second, Laplace's Demon considers a combination of five conditions when making the largest part of its suggestion. These conditions are: the algorithm, acceptance rate, MCSE, ESS, and stationarity. Other things are considered as well, such as the recommended thinning value is used to suggest a new number of iterations, how fast the algorithm is expected to be, and if the condition of diminishing adaptation (also called the vanishing adaptation condition) was met (for an adaptive algorithm). Diminishing adaptation occurs only when the absolute value of the proposed variances trends downward (toward zero) over the course of all adaptations. When an algorithm is adaptive and it does not have diminishing adaptations, the `Consort` function will suggest a different adaptive algorithm. The `Periodicity` argument is suggested to be set equal to the value of `Rec.Thinning`.

Appeasement applies only when all parameters are continuous.The `Hangartner.Diagnostic` should be considered for discrete parameters.

Appeasement Conditions

- Algorithm: The final algorithm must be non-adaptive, so that the Markov property holds. This is conservative. A user may have an adaptive (non-final) algorithm in which adaptations in the latest update are stationary, or no longer diminishing. Laplace's Demon is unaware of previous updates, and conservatively interprets this as failing to meet the condition of diminishing adaptation, when the output may be satisfactory. On the other hand, if the adaptive algorithm has essentially stopped adapting, and if there is a non-adaptive version, then the user should consider switching to the non-adaptive algorithm. User discretion is advised.
- Acceptance Rate: The acceptance rate is considered satisfactory if it is within the interval [15%,50%] for most algorithms. Some algorithms have different recommended intervals.
- MCSE: The Monte Carlo Standard Error (MCSE) is considered satisfactory for each target distribution if it is less than 6.27% of the standard deviation of the target distribution. This allows the true mean to be within 5% of the area under a Gaussian distribution around the estimated mean. The `MCSE` function is used. Toft et al. (2007) propose a stricter criterion of 5%. The criterion of 6.27% for this stopping rule is arbitrary, and may be too lenient or strict, depending on the needs of the user. Nonetheless, it has performed well, and this type of stopping rule has been observed to perform better than MCMC convergence diagnostics (Flegal et al., 2008).
- ESS: The effective sample size (ESS) is considered satisfactory for each target distribution if it is at least 100, which is usually enough to describe 95% probability intervals (see `p.interval` and `LPL.interval` for more information). The `ESS` function is used. When this criterion is unmet, the name of the worst mixing chain in Summary1 appears.
- Stationarity: Each target distribution is considered satisfactory if it is estimated to be stationary with the `BMK.Diagnostic` function.

Bear in mind that the MCSE, ESS, and stationarity criteria are all univariate measures applied to each marginal posterior distribution. Multivariate forms are not included. By chance alone

due to multiple independent tests, 5% of these diagnostics should indicate non-convergence when 'convergence' exists. In contrast, even one non-convergent nuisance parameter is associated with non-convergence in all other parameters. Assessing convergence is difficult.

If all five conditions are satisfactory, then Laplace's Demon is appeased. Otherwise, Laplace's Demon will suggest and supply R code that is ready to be copy/pasted and executed.

To visualize the MCSE-based stopping rule, run the following code:

```
x <- seq(from=-3, to=3, by=0.1); plot(x, dnorm(x,0,1), type="l"); abline(v=-0.0627);
abline(v=0.0627); abline(v=2*-0.0627, col="red"); abline(v=2*0.0627, col="red")
```

The black vertical lines show the standard error, and the red vertical lines show the 95% interval.

If the user has an object of class `demonoid.hpc`, then the `Consort` function may be still be applied, but a particular chain in the object must be specified as a component in a list. For example, with an object called `Fit` and a goal of consorting over the second chain, the code would be: `Consort(Fit[[2]])`.

The Demonic Suggestion is usually very helpful, but should not be followed blindly. Do not let it replace critical thinking. For example, `Consort` may find that diminishing adaptation is unmet, and recommend a different algorithm. However, the user may be convinced that the current algorithm is best, and believe instead that MCMC found a local solution, and is leaving it to find the global solution, in which case adaptations may increase again. Diminishing adaptation may have occurred in a previous run, and is not found in the current run because adaptation is essentially finished. If either of these is true, then it may be best to ignore the newly suggested algorithm, and continue with the current algorithm. The suggested code may be helpful, but it is merely a suggestion.

If achieving the appeasement of Laplace's Demon is difficult, consider ignoring the MCSE criterion and terminate when all other criteria have been met, placing special emphasis on ESS.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Flegal, J.M., Haran, M., and Jones, G.L. (2008). "Markov chain Monte Carlo: Can We Trust the Third Significant Figure?". *Statistical Science*, 23, p. 250–260.

Toft, N., Innocent, G., Gettinby, G., and Reid, S. (2007). "Assessing the Convergence of Markov Chain Monte Carlo Methods: An Example from Evaluation of Diagnostic Tests in Absence of a Gold Standard". *Preventive Veterinary Medicine*, 79, p. 244–256.

See Also

`BMK.Diagnostic`, `ESS`, `Hangartner.Diagnostic`, `LaplacesDemon`, `LaplacesDemon.hpc`, `LPL.interval`, `MCSE`, and `p.interval`.

CSF *Cumulative Sample Function*

Description

The Cumulative Sample Function (CSF) is a visual MCMC diagnostic in which the user may select a measure (such as a variable, summary statistic, or other diagnostic), and observe a plot of how the measure changes over cumulative posterior samples from MCMC, such as the output of LaplacesDemon. This may be considered to be a generalized extension of the cumuplot in the coda package, which is a more restrictive form of the cusum diagnostic introduced by Yu and Myckland (1998).

Yu and Myckland (1998) suggest that CSF plots should be examined after traditional trace plots seem convergent, and assert that faster mixing chains (which are more desirable) result in CSF plots that are more 'hairy' (as opposed to smooth), though this is subjective and has been debated. The LaplacesDemon package neither supports nor contradicts the suggestion of mixing and 'hairiness', but suggests that CSF plots may be used to provide additional information about a chain. For example, a user may decide on a practical burnin given when a conditional mean obtains a certain standard error.

Usage

```
CSF(x, name, method="Quantiles", quantiles=c(0.025,0.500,0.975), output=FALSE)
```

Arguments

x	This is a vector of posterior samples from MCMC.
name	This is an optional name for vector x, and is input as a quoted string, such as name="theta".
method	This is a measure that will be observed over the course of cumulative samples of x. It defaults to method="Quantiles", and optional methods include: "ESS", "Geweke.Diagnostic", "HPD", "is.stationary", "Kurtosis", "MCSE", "MCSE.bm", "MCSE.sv", "Mean", "Mode", "N.Modes", "Precision", "Quantiles", and "Skewness".
quantiles	This optional argument applies only when method="Quantiles", in which case this vector indicates the probabilities that will be observed. It defaults to the median and 95% probability interval bounds (see p.interval for more information).
output	Logical. If output=TRUE, then the results of the measure over the course of the cumulative samples will be output as an object, either a vector or matrix, depending on the method argument. The output argument defaults to FALSE.

Details

When method="ESS", the effective sample size (ESS) is observed as a function of the cumulative samples of x. For more information, see the ESS function.

When method="Geweke.Diagnostic", the Z-score output of the Geweke diagnostic is observed as a function of the cumulative samples of x. For more information, see the Geweke.Diagnostic function.

When method="HPD", the Highest Posterior Density (HPD) interval is observed as a function of the cumulative samples of x. For more information, see the p.interval function.

When method="is.stationary", stationarity is logically tested and the result is observed as a function of the cumulative samples of x. For more information, see the is.stationary function.

When method="Kurtosis", kurtosis is observed as a function of the cumulative samples of x.

When method="MCSE", the Monte Carlo Standard Error (MCSE) estimated with the IMPS method is observed as a function of the cumulative samples of x. For more information, see the MCSE function.

When method="MCSE.bm", the Monte Carlo Standard Error (MCSE) estimated with the batch.means method is observed as a function of the cumulative samples of x. For more information, see the MCSE function.

When method="MCSE.sv", the Monte Carlo Standard Error (MCSE) estimated with the sample.variance method is observed as a function of the cumulative samples of x. For more information, see the MCSE function.

When method="Mean", the mean is observed as a function of the cumulative samples of x.

When method="Mode", the estimated mode is observed as a function of the cumulative samples of x. For more information, see the Mode function.

When method="N.Modes", the estimated number of modes is observed as a function of the cumulative samples of x. For more information, see the Modes function.

When method="Precision", the precision (inverse variance) is observed as a function of the cumulative samples of x.

When method="Quantiles", the quantiles selected with the quantiles argument are observed as a function of the cumulative samples of x.

When method="Skewness", skewness is observed as a function of the cumulative samples of x.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Yu, B. and Myckland, P. (1997). "Looking at Markov Samplers through Cusum Path Plots: A Simple Diagnostic Idea". *Statistics and Computing*, 8(3), p. 275–286.

See Also

burnin, ESS, Geweke.Diagnostic, is.stationary, LaplacesDemon, MCSE, Mode, Modes, and p.interval.

Examples

```
#Commented-out because of run-time for package builds
#library(LaplacesDemon)
#x <- rnorm(1000)
#CSF(x, method="ESS")
#CSF(x, method="Geweke.Diagnostic")
#CSF(x, method="HPD")
#CSF(x, method="is.stationary")
#CSF(x, method="Kurtosis")
#CSF(x, method="MCSE")
#CSF(x, method="MCSE.bm")
#CSF(x, method="MCSE.sv")
#CSF(x, method="Mean")
#CSF(x, method="Mode")
#CSF(x, method="N.Modes")
#CSF(x, method="Precision")
#CSF(x, method="Quantiles")
#CSF(x, method="Skewness")
```

---

data.demonchoice *Demon Choice Data Set*

Description

This data set is for discrete choice models and consists of the choice of commuting route to school: arterial, two-lane, or freeway. There were 151 Pennsylvania commuters who started from a residential complex in State College, PA, and commute to downtown State College.

Usage

```
data(demonchoice)
```

Format

This data frame contains 151 rows of individual choices and 9 columns. The following data dictionary describes each variable or column.

`Choice` This is the route choice: four-lane arterial (35 MPH speed limit), two-lane highway (35 MPH speed limit, with one lane in each direction), or a limited-access four-lane freeway (55 MPH speed liimit.)

`HH.Income` This is an ordinal variable of annual household income of the commuter in USD. There are four categories: 1 is less than 20,000 USD, 2 is 20,000-29,999 USD, 3 is 30,000-39,999 USD, and 4 is 40,000 USD or greater.

`Vehicle.Age` This is the age in years of the vehicle of the commuter.

`Stop.Signs.Arterial` This is the number of stop signs along the arterial route.

`Stop.Signs.Two.Lane` This is the number of stop signs along the two-lane route.

`Stop.Signs.Freeway` This is the number of stop signs along the freeway route.

`Distance.Arterial` This is distance in miles of the arterial route.

`Distance.Two.Lane` This is the distance in miles of the two-lane route.

`Distance.Freeway` This is the distance in miles of the freeway route.

Source

Washington, S., Congdon, P., Karlaftis, M., and Mannering, F. (2009). "Bayesian Multinomial Logit: Theory and Route Choice Example". Transportation Research Record, 2136, p. 28–36.

`data.demonfx` *Demon FX Data Set*

Description

This data set consists of daily currency pair prices from 2010 through 2014. Each currency pair has a close, high, and low price.

Usage

```
data(demonfx)
```

Format

This data frame contains 1,301 rows as time-periods (with row names) and 39 columns of currency pair prices. The following data dictionary describes each time-series or column.

`EURUSD.Close` This is the currency pair closing price.

`EURUSD.High` This is the currency pair high price.

`EURUSD.Low` This is the currency pair low price.

`USDJPY.Close` This is the currency pair closing price.

`USDJPY.High` This is the currency pair high price.

`USDJPY.Low` This is the currency pair low price.

`USDCHF.Close` This is the currency pair closing price.

`USDCHF.High` This is the currency pair high price.

`USDCHF.Low` This is the currency pair low price.

`GBPUSD.Close` This is the currency pair closing price.

`GBPUSD.High` This is the currency pair high price.

`GBPUSD.Low` This is the currency pair low price.

`USDCAD.Close` This is the currency pair closing price.

`USDCAD.High` This is the currency pair high price.

`USDCAD.Low` This is the currency pair low price.

`EURGBP.Close` This is the currency pair closing price.

`EURGBP.High` This is the currency pair high price.

`EURGBP.Low` This is the currency pair low price.

`EURJPY.Close` This is the currency pair closing price.

`EURJPY.High` This is the currency pair high price.

`EURJPY.Low` This is the currency pair low price.

`EURCHF.Close` This is the currency pair closing price.

`EURCHF.High` This is the currency pair high price.

`EURCHF.Low` This is the currency pair low price.

`AUDUSD.Close` This is the currency pair closing price.

`AUDUSD.High` This is the currency pair high price.

`AUDUSD.Low` This is the currency pair low price.

`GBPJPY.Close` This is the currency pair closing price.

`GBPJPY.High` This is the currency pair high price.

`GBPJPY.Low` This is the currency pair low price.

`CHFJPY.Close` This is the currency pair closing price.

`CHFJPY.High` This is the currency pair high price.

`CHFJPY.Low` This is the currency pair low price.

`GBPCHF.Close` This is the currency pair closing price.

`GBPCHF.High` This is the currency pair high price.

`GBPCHF.Low` This is the currency pair low price.

`NZDUSD.Close` This is the currency pair closing price.

`NZDUSD.High` This is the currency pair high price.

`NZDUSD.Low` This is the currency pair low price.

Source

https://www.global-view.com/forex-trading-tools/forex-history/index.html

---

`data.demonsessions` *Demon Sessions Data Set*

Description

These are the monthly number of user sessions at https://web.archive.org/web/20141224051720/http://www.bayesian-inference.com/index by continent. Additional data may be added in the future.

Usage

```
data(demonsessions)
```

Format

This data frame contains 26 rows (with row names) and 6 columns. The following data dictionary describes each variable or column.

`Africa` This is the African continent.

`Americas` This is North and South America.

`Asia` This is the Asian continent.

`Europe` This is Europe as a continent.

`Oceania` This is Oceania, such as Australia.

`Not.Set` This includes sessions in which the continent was not set, or is unknown.

Source

https://web.archive.org/web/20141224051720/http://www.bayesian-inference.com/index

---

`data.demonsnacks` *Demon Snacks Data Set*

Description

Late one night, after witnessing Laplace's Demon in action, I followed him back to what seemed to be his lair. Minutes later, he left again. I snuck inside and saw something labeled 'Demon Snacks'. Hurriedly, I recorded the 39 items, each with a name and 10 nutritional attributes.

Usage

```
data(demonsnacks)
```

Format

This data frame contains 39 rows (with row names) and 10 columns. The following data dictionary describes each variable or column.

`Serving.Size` This is serving size in grams.

`Calories` This is the number of calories.

`Total.Fat` This is total fat in grams.

`Saturated.Fat` This is saturated fat in grams.

`Cholesterol` This is cholesterol in milligrams.

`Sodium` This is sodium in milligrams.

`Total.Carbohydrate` This is the total carbohydrates in grams.

`Dietary.Fiber` This is dietary fiber in grams.

`Sugars` This is sugar in grams.

`Protein` This is protein in grams.

Source

This data was obtained from the lair of Laplace's Demon!

`data.demontexas` *Demon Space-Time Data Set*

Description

This data set is for space-time models that require latitude and longitude, or coordinates. This data set consists of the minimum, mean, and maximum temperatures in Texas for 13 months.

Usage

```
data(demontexas)
```

Format

This data frame contains 369 rows of sites in Texas and 43 columns. The following data dictionary describes each variable or column.

`Elevation` This is the elevation of the site.

`Latitude` This is the latitude of the site.

`Longitude` This is the longitude of the site.

`Gulf` This is a gulf indicator of the site.

`Max1` This is the maximum temperature in month 1.

`Max2` This is the maximum temperature in month 2.

`Max3` This is the maximum temperature in month 3.

`Max4` This is the maximum temperature in month 4.

`Max5` This is the maximum temperature in month 5.

`Max6` This is the maximum temperature in month 6.

`Max7` This is the maximum temperature in month 7.

`Max8` This is the maximum temperature in month 8.

`Max9` This is the maximum temperature in month 9.

`Max10` This is the maximum temperature in month 10.

`Max11` This is the maximum temperature in month 11.

`Max12` This is the maximum temperature in month 12.

`Max13` This is the maximum temperature in month 13.

`Mean1` This is the mean temperature in month 1.

`Mean2` This is the mean temperature in month 2.

`Mean3` This is the mean temperature in month 3.

`Mean4` This is the mean temperature in month 4.

`Mean5` This is the mean temperature in month 5.

`Mean6` This is the mean temperature in month 6.

`Mean7` This is the mean temperature in month 7.

`Mean8` This is the mean temperature in month 8.

`Mean9` This is the mean temperature in month 9.

`Mean10` This is the mean temperature in month 10.

`Mean11` This is the mean temperature in month 11.

`Mean12` This is the mean temperature in month 12.

`Mean13` This is the mean temperature in month 13.

`Min1` This is the minimum temperature in month 1.

`Min2` This is the minimum temperature in month 2.

`Min3` This is the minimum temperature in month 3.

`Min4` This is the minimum temperature in month 4.

`Min5` This is the minimum temperature in month 5.

`Min6` This is the minimum temperature in month 6.

`Min7` This is the minimum temperature in month 7.

`Min8` This is the minimum temperature in month 8.

`Min9` This is the minimum temperature in month 9.

`Min10` This is the minimum temperature in month 10.

`Min11` This is the minimum temperature in month 11.

`Min12` This is the minimum temperature in month 12.

`Min13` This is the minimum temperature in month 13.

Source

http://www.stat.ufl.edu/~winner/datasets.html

`de.Finetti.Game` *de Finetti's Game*

Description

The `de.Finetti.Game` function estimates the interval of a subjective probability regarding a possible event in the near future.

Usage

```
de.Finetti.Game(width)
```

Arguments

`width`	This is the maximum acceptable width of the interval for the returned subjective probability. The user must specify a width between 0 and 1.

Details

This function is a variation on the game introduced by de Finetti, who is one of the main developers of subjective probability, along with Ramsey and Savage. In the original context, de Finetti proposed a gamble regarding life on Mars one billion years ago.

The frequentist interpretation of probability defines the probability of an event as the limit of its relative frequency in a large number of trials. Frequentist inference is undefined, for example, when there are no trials from which to calculate a probability. By defining probability relative to frequencies of physical events, frequentists attempt to objectify probability. However, de Finetti asserts that the frequentist (or objective) interpretation always reduces to a subjective interpretation of probability, because probability is a human construct and does not exist independently of humans in nature. Therefore, probability is a degree of belief, and is called subjective or personal probability.

Value

The `de.Finetti.Game` function returns a vector of length two. The respective elements are the lower and upper bounds of the subjective probability of the participant regarding the possible event in the near future.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`elicit`

`deburn` *De-Burn*

Description

The `deburn` function discards or removes a user-specified number of burn-in iterations from an object of class `demonoid`.

Usage

```
deburn(x, BurnIn=0)
```

Arguments

`x`	This is an object of class `demonoid`.
`BurnIn`	This argument defaults to `BurnIn=0`, and accepts an integer that indicates the number of iterations to discard as burn-in.

Details

Documentation for the burnin function provides an introduction to the concept of burn-in as it relates to Markov chains.

The deburn function discards a number of the first posterior samples, as specified by the BurnIn argument. Stationarity is not checked, because it is assumed the user has a reason for using the deburn function, rather than using the results from the object of class demonoid. Therefore, the posterior samples in Posterior1 and Posterior2 are identical, as are Summary1 and Summary2.

Value

The deburn function returns an object of class demonoid.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

burnin and LaplacesDemon.

Examples

```
### Assuming the user has Fit which is an object of class demonoid:
#library(LaplacesDemon)
#Fit2 <- deburn(Fit, BurnIn=100)
```

---

dist.Asymmetric.Laplace

Asymmetric Laplace Distribution: Univariate

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, asymmetric Laplace distribution with location parameter location, scale parameter scale, and asymmetry or skewness parameter kappa.

Usage

```
dalaplace(x, location=0, scale=1, kappa=1, log=FALSE)
palaplace(q, location=0, scale=1, kappa=1)
qalaplace(p, location=0, scale=1, kappa=1)
ralaplace(n, location=0, scale=1, kappa=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
location	This is the location parameter μ.
scale	This is the scale parameter λ, which must be positive.
kappa	This is the asymmetry or skewness parameter κ, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\kappa\sqrt{2}}{\lambda(1+\kappa^2)} \exp(-|\theta-\mu|\frac{\sqrt{2}}{\lambda}\kappa^{|\theta-\mu|}|\theta-\mu|)$
- Inventor: Kotz, Kozubowski, and Podgorski (2001)
- Notation 1: $\theta \sim \mathcal{AL}(\mu, \lambda, \kappa)$
- Notation 2: $p(\theta) = \mathcal{AL}(\theta|\mu, \lambda, \kappa)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\lambda > 0$
- Parameter 3: skewness parameter $\kappa > 0$
- Mean: $E(\theta) = \mu + \lambda\frac{1/\kappa - \kappa}{\sqrt{2}}$
- Variance: $var(\theta) = \lambda^2\frac{1+\kappa^4}{2\kappa^2}$
- Mode: $mode(\theta) = \mu$

The asymmetric Laplace of Kotz, Kozubowski, and Podgorski (2001), also referred to as AL, is an extension of the univariate, symmetric Laplace distribution to allow for skewness. It is parameterized according to three parameters: location parameter μ, scale parameter λ, and asymmetry or skewness parameter κ. The special case of $\kappa = 1$ is the symmetric Laplace distribution. Values of κ in the intervals $(0, 1)$ and $(1, \infty)$, correspond to positive (right) and negative (left) skewness, respectively. The AL distribution is leptokurtic, and its kurtosis ranges from 3 to 6 as κ ranges from 1 to infinity. The skewness of the AL has been useful in engineering and finance. As an example, the AL distribution has been used as a replacement for Gaussian-distributed GARCH residuals. There is also an extension to the asymmetric multivariate Laplace distribution.

The asymmetric Laplace distribution is demonstrated in Kozubowski and Podgorski (2001) to be well-suited for financial modeling, specifically with currency exchange rates.

These functions are similar to those in the VGAM package.

Value

dalaplace gives the density, palaplace gives the distribution function, qalaplace gives the quantile function, and ralaplace generates random deviates.

References

Kotz, S., Kozubowski, T.J., and Podgorski, K. (2001). "The Laplace Distribution and Generalizations: a Revisit with Applications to Communications, Economics, Engineering, and Finance". Boston: Birkhauser.

Kozubowski, T.J. and Podgorski, K. (2001). "Asymmetric Laplace Laws and Modeling Financial Data". *Mathematical and Computer Modelling*, 34, p. 1003-1021.

See Also

dlaplace and dallaplace

Examples

```
library(LaplacesDemon)
x <- dalaplace(1,0,1,1)
x <- palaplace(1,0,1,1)
x <- qalaplace(0.5,0,1,1)
x <- ralaplace(100,0,1,1)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dalaplace(x,0,1,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dalaplace(x,0,1,1), type="l", col="green")
lines(x, dalaplace(x,0,1,5), type="l", col="blue")
legend(1, 0.9, expression(paste(mu==0, ", ", lambda==1, ", ", kappa==0.5),
     paste(mu==0, ", ", lambda==1, ", ", kappa==1),
     paste(mu==0, ", ", lambda==1, ", ", kappa==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Asymmetric.Log.Laplace

Asymmetric Log-Laplace Distribution

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, asymmetric, log-Laplace distribution with location parameter μ, scale parameter λ, and asymmetry or skewness parameter κ.

Usage

```
dallaplace(x, location=0, scale=1, kappa=1, log=FALSE)
pallaplace(q, location=0, scale=1, kappa=1)
qallaplace(p, location=0, scale=1, kappa=1)
rallaplace(n, location=0, scale=1, kappa=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
location	This is the location parameter μ.
scale	This is the scale parameter λ, which must be positive.
kappa	This is the asymmetry or skewness parameter κ, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density 1: $p(\theta) = \exp(-\mu)\frac{(\sqrt{(2)}\kappa/\lambda)(\sqrt{(2)}/\lambda\kappa)}{(\sqrt{(2)}\kappa/\lambda)+(\sqrt{(2)}/(\lambda\kappa))}\exp(-(\frac{\sqrt{(2)}\kappa}{\lambda})+1), \quad \theta \geq \exp(\mu)$
- Density 2: $p(\theta) = \exp(-\mu)\frac{(\sqrt{(2)}\kappa/\lambda)(\sqrt{(2)}/(\lambda\kappa))}{(\sqrt{(2)}\kappa/\lambda)+(\sqrt{(2)}/(\lambda\kappa))}\exp(\frac{\sqrt{(2)}(\log(\theta)-\mu)}{\lambda\kappa}-(\log(\theta)-\mu)), \quad \theta < \exp(\mu)$
- Inventor: Pierre-Simon Laplace
- Notation 1: $\theta \sim \mathcal{ALL}(\mu, \lambda, \kappa)$
- Notation 2: $p(\theta) = \mathcal{ALL}(\theta|\mu, \lambda, \kappa)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\lambda > 0$
- Mean: $E(\theta) =$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) =$

The univariate, asymmetric log-Laplace distribution is derived from the Laplace distribution. Multivariate and symmetric versions also exist.

These functions are similar to those in the VGAM package.

Value

dallaplace gives the density, pallaplace gives the distribution function, qallaplace gives the quantile function, and rallaplace generates random deviates.

References

Kozubowski, T. J. and Podgorski, K. (2003). "Log-Laplace Distributions". *International Mathematical Journal*, 3, p. 467–495.

See Also

dalaplace, dexp, dlaplace, dlaplacep, dllaplace, dmvl, dnorm, dnormp, dnormv.

Examples

```
library(LaplacesDemon)
x <- dallaplace(1,0,1,1)
x <- pallaplace(1,0,1,1)
x <- qallaplace(0.5,0,1,1)
x <- rallaplace(100,0,1,1)

#Plot Probability Functions
x <- seq(from=0.1, to=10, by=0.1)
plot(x, dallaplace(x,0,1,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dallaplace(x,0,1,1), type="l", col="green")
lines(x, dallaplace(x,0,1,5), type="l", col="blue")
legend(5, 0.9, expression(paste(mu==0, ", ", lambda==1, ", ", kappa==0.5),
     paste(mu==0, ", ", lambda==1, ", ", kappa==1),
     paste(mu==0, ", ", lambda==1, ", ", kappa==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Asymmetric.Multivariate.Laplace

Asymmetric Multivariate Laplace Distribution

Description

These functions provide the density and random generation for the asymmetric multivariate Laplace distribution with location and skew parameter μ and covariance Σ.

Usage

```
daml(x, mu, Sigma, log=FALSE)
raml(n, mu, Sigma)
```

Arguments

x	This is a $N \times K$ matrix of data, or a vector of length K.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the location and skew parameter μ. This may be a $N \times K$ matrix, or a vector of length K.
Sigma	This is the $K \times K$ positive-definite covariance matrix Σ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \frac{2\exp(\theta\Omega\theta)}{(2\pi)^{k/2}|\Sigma|^{0}.5} \frac{\theta\Omega\theta}{2+\mu\Omega\mu}^{(2-k)/4} K_{(2-k)/2}(\sqrt{(2+\mu\Omega\mu)(\theta\Omega\theta)})$
- Inventor: Kotz, Kozubowski, and Podgorski (2003)
- Notation 1: $\theta \sim \mathcal{AL}_K(\mu, \Sigma)$
- Notation 2: $p(\theta) = \mathcal{AL}_K(\theta|\mu, \Sigma)$
- Parameter 1: location-skew parameter μ
- Parameter 2: positive-definite covariance matrix Σ
- Mean: Unknown
- Variance: Unknown
- Mode: $mode(\theta) = \mu$

The asymmetric multivariate Laplace distribution of Kotz, Kozubowski, and Podgorski (2003) is a multivariate extension of the univariate, asymmetric Laplace distribution. It is parameterized according to two parameters: location-skew parameter μ and positive-definite covariance matrix Σ. Location and skew occur in the same parameter. When $\mu = 0$, the density is the (symmetric) multivariate Laplace of Anderson (1992). As each location deviates from zero, the marginal distribution becomes more skewed. Since location and skew are combined, it is appropriate for zero-centered variables, such as a matrix of centered and scaled dependent variables in cluster analysis, factor analysis, multivariate regression, or multivariate time-series.

The asymmetric multivariate Laplace distribution is also discussed earlier in Kozubowski and Podgorski (2001), and is well-suited for financial modeling via multivariate regression, specifically with currency exchange rates. Cajigas and Urga (2005) fit residuals in a multivariate GARCH model with the asymmetric multivariate Laplace distribution, regarding stocks and bonds. They find that it "overwhelmingly outperforms" normality.

Value

`daml` gives the density, and `raml` generates random deviates.

References

Anderson, D.N. (1992). "A Multivariate Linnik Distribution". *Statistical Probability Letters*, 14, p. 333–336.

Cajigas, J.P. and Urga, G. (2005) "Dynamic Conditional Correlation Models with Asymmetric Laplace Innovations". Centre for Economic Analysis: Cass Business School.

Kotz, S., Kozubowski, T.J., and Podgorski, K. (2003). "An Asymmetric Multivariate Laplace Distribution". Working Paper.

Kozubowski, T.J. and Podgorski, K. (2001). "Asymmetric Laplace Laws and Modeling Financial Data". *Mathematical and Computer Modelling*, 34, p. 1003–1021.

See Also

`dalaplace` and `dmvl`

Examples

```
library(LaplacesDemon)
x <- daml(c(1,2,3), c(0,1,2), diag(3))
X <- raml(1000, c(0,1,2), diag(3))
joint.density.plot(X[,1], X[,2], color=FALSE)
```

`dist.Bernoulli` *Bernoulli Distribution*

Description

These functions provide the density, distribution function, quantile function, and random generation for the Bernoulli distribution.

Usage

```
dbern(x, prob, log=FALSE)
pbern(q, prob, lower.tail=TRUE, log.p=FALSE)
qbern(p, prob, lower.tail=TRUE, log.p=FALSE)
rbern(n, prob)
```

Arguments

`x, q`	These are each a vector of quantiles.
`p`	This is a vector of probabilities.
`n`	This is the number of observations. If `length(n) > 1`, then the length is taken to be the number required.
`prob`	This is the probability of success on each trial.
`log, log.p`	Logical. if `TRUE`, probabilities p are given as $\log(p)$.
`lower.tail`	Logical. if `TRUE` (default), probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = p^{\theta}(1-p)^{1-\theta}, \theta = 0, 1$
- Inventor: Jacob Bernoulli
- Notation 1: $\theta \sim \mathcal{BERN}(p)$
- Notation 2: $p(\theta) = \mathcal{BERN}(\theta|p)$
- Parameter 1: probability parameter $0 \leq p \leq 1$
- Mean: $E(\theta) = p$
- Variance: $var(\theta) = \frac{p}{1-p}$

- Mode: $mode(\theta) =$

The Bernoulli distribution is a binomial distribution with $n = 1$, and one instance of a Bernoulli distribution is called a Bernoulli trial. One coin flip is a Bernoulli trial, for example. The categorical distribution is the generalization of the Bernoulli distribution for variables with more than two discrete values. The beta distribution is the conjugate prior distribution of the Bernoulli distribution. The geometric distribution is the number of Bernoulli trials needed to get one success.

Value

`dbern` gives the density, `pbern` gives the distribution function, `qbern` gives the quantile function, and `rbern` generates random deviates.

See Also

`dbinom`

Examples

```
library(LaplacesDemon)
dbern(1, 0.7)
rbern(10, 0.5)
```

---

dist.Categorical *Categorical Distribution*

Description

This is the density and random deviates function for the categorical distribution with probabilities parameter p.

Usage

```
dcat(x, p, log=FALSE)
qcat(pr, p, lower.tail=TRUE, log.pr=FALSE)
rcat(n, p)
```

Arguments

`x`	This is a vector of discrete data with k discrete categories, and is of length n. This function also accepts x after it has been converted to an $n \times k$ indicator matrix, such as with the `as.indicator.matrix` function.
`n`	This is the number of observations, which must be a positive integer that has length 1. When `p` is supplied to `rcat` as a matrix, `n` must equal the number of rows in `p`.
`p`	This is a vector of length k or $n \times k$ matrix of probabilities. The `qcat` function requires a vector.
`pr`	This is a vector of probabilities, or log-probabilities.

`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.
`log.pr`	Logical. if `TRUE`, probabilities pr are given as $\log(pr)$.
`lower.tail`	Logical. if `TRUE` (default), probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Discrete Univariate
- Density: $p(\theta) = \sum \theta p$
- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{CAT}(p)$
- Notation 2: $p(\theta) = \mathcal{CAT}(\theta|p)$
- Parameter 1: probabilities p
- Mean: $E(\theta)$ = Unknown
- Variance: $var(\theta)$ = Unknown
- Mode: $mode(\theta)$ = Unknown

Also called the discrete distribution, the categorical distribution describes the result of a random event that can take on one of k possible outcomes, with the probability p of each outcome separately specified. The vector p of probabilities for each event must sum to 1. The categorical distribution is often used, for example, in the multinomial logit model. The conjugate prior is the Dirichlet distribution.

Value

`dcat` gives the density and `rcat` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`as.indicator.matrix`, `ddirichlet`, and `dmultinom`.

Examples

```
library(LaplacesDemon)
dcat(x=1, p=c(0.3,0.3,0.4))
rcat(n=10, p=c(0.1,0.3,0.6))
```

`dist.ContinuousRelaxation` *Continuous Relaxation of a Markov Random Field Distribution*

Description

This is the density function and random generation from the continuous relaxation of a Markov random field (MRF) distribution.

Usage

```
dcrmrf(x, alpha, Omega, log=FALSE)
rcrmrf(n, alpha, Omega)
```

Arguments

`x`	This is a vector of length k.
`n`	This is the number of random deviates to generate.
`alpha`	This is a vector of length k of shape parameters.
`Omega`	This is the $k \times k$ precision matrix Ω.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:
$$p(\theta) \propto \exp(-\frac{1}{2}\theta^T \Omega^{-1} \theta) \prod_i (1 + \exp(\theta_i + alpha_i))$$
- Inventor: Zhang et al. (2012)
- Notation 1: $\theta \sim \mathcal{CRMRF}(\alpha, \Omega)$
- Notation 2: $p(\theta) = \mathcal{CRMRF}(\theta|\alpha, \Omega)$
- Parameter 1: shape vector α
- Parameter 2: positive-definite $k \times k$ matrix Ω
- Mean: $E(\theta)$
- Variance: $var(\theta)$
- Mode: $mode(\theta)$

It is often easier to solve or optimize a problem with continuous variables rather than a problem that involves discrete variables. A continuous variable may also have a gradient, contour, and curvature that may be useful for optimization or sampling. Continuous MCMC samplers are far more common.

Zhang et al. (2012) introduced a generalized form of the Gaussian integral trick from statistical physics to transform a discrete variable so that it may be estimated with continuous variables. An

auxiliary Gaussian variable is added to a discrete Markov random field (MRF) so that discrete dependencies cancel out, allowing the discrete variable to be summed away, and leaving a continuous problem. The resulting continuous representation of the problem allows the model to be updated with a continuous MCMC sampler, and may benefit from a MCMC sampler that uses derivatives. Another advantage of continuous MCMC is that stationarity of discrete Markov chains is problematic to assess.

A disadvantage of solving a discrete problem with continuous parameters is that the continuous solution requires more parameters.

Value

`dcrmrf` gives the density and `rcrmrf` generates random deviates.

References

Zhang, Y., Ghahramani, Z., Storkey, A.J., and Sutton, C.A. (2012). "Continuous Relaxations for Discrete Hamiltonian Monte Carlo". *Advances in Neural Information Processing Systems*, 25, p. 3203–3211.

See Also

`dmvn`

Examples

```
library(LaplacesDemon)
x <- dcrmrf(rnorm(5), rnorm(5), diag(5))
x <- rcrmrf(10, rnorm(5), diag(5))
```

---

dist.Dirichlet *Dirichlet Distribution*

Description

This is the density function and random generation from the Dirichlet distribution.

Usage

```
ddirichlet(x, alpha, log=FALSE)
rdirichlet(n, alpha)
```

Arguments

`x`	This is a vector containing a single deviate or matrix containing one random deviate per row. Each vector, or matrix row, must sum to 1.
`n`	This is the number of random deviates to generate.
`alpha`	This is a vector or matrix of shape parameters.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\gamma(\alpha_1 + \ldots + \alpha_k)}{\gamma\alpha_1 \ldots \gamma\alpha_k} \theta_1^{(\alpha[1]-1)} \ldots \theta_k^{(\alpha[k]-1)}, \quad \theta_1, \ldots, \theta_k > 0, \quad \sum_{j=1}^{k} \theta_j = 1$$

- Inventor: Johann Peter Gustav Lejeune Dirichlet (1805-1859)
- Notation 1: $\theta \sim \text{Dirichlet}(\alpha_1, \ldots, \alpha_k)$
- Notation 2: $p(\theta) = \text{Dirichlet}(\theta|\alpha_1, \ldots, \alpha_k)$
- Notation 3: $\theta \sim \mathcal{DIR}(\alpha_1, \ldots, \alpha_k)$
- Notation 4: $p(\theta) = \mathcal{DIR}(\theta|\alpha_1, \ldots, \alpha_k)$
- Parameter: 'prior sample sizes' $\alpha_j > 0, \alpha_0 = \sum_{j=1}^{k} \alpha_j$
- Mean: $E(\theta_j) = \frac{\alpha_j}{\alpha_0}$
- Variance: $var(\theta_j) = \frac{\alpha_j(\alpha_0 - \alpha_j)}{\alpha_0^2(\alpha_0+1)}$
- Covariance: $cov(\theta_i, \theta_j) = -\frac{\alpha_i\alpha_j}{\alpha_0^2(\alpha_0+1)}$
- Mode: $mode(\theta_j) = \frac{\alpha_j - 1}{\alpha_0 - k}$

The Dirichlet distribution is the multivariate generalization of the univariate beta distribution. Its probability density function returns the belief that the probabilities of k rival events are θ_j given that each event has been observed $\alpha_j - 1$ times.

The Dirichlet distribution is commonly used as a prior distribution in Bayesian inference. The Dirichlet distribution is the conjugate prior distribution for the parameters of the categorical and multinomial distributions.

A very common special case is the symmetric Dirichlet distribution, where all of the elements in parameter vector α have the same value. Symmetric Dirichlet distributions are often used as vague or weakly informative Dirichlet prior distributions, so that one component is not favored over another. The single value that is entered into all elements of α is called the concentration parameter.

Value

`ddirichlet` gives the density and `rdirichlet` generates random deviates.

See Also

`dbeta`, `dcat`, `dmvpolya`, `dmultinom`, and `TransitionMatrix`.

Examples

```
library(LaplacesDemon)
x <- ddirichlet(c(.1,.3,.6), c(1,1,1))
x <- rdirichlet(10, c(1,1,1))
```

`dist.Generalized.Pareto` *Generalized Pareto Distribution*

Description

These are the density and random generation functions for the generalized Pareto distribution.

Usage

```
dgpd(x, mu, sigma, xi, log=FALSE)
rgpd(n, mu, sigma, xi)
```

Arguments

`x`	This is a vector of data.
`n`	This is a positive scalar integer, and is the number of observations to generate randomly.
`mu`	This is a scalar or vector location parameter μ. When ξ is non-negative, μ must not be greater than $\mathbf{x}$. When ξ is negative, μ must be less than $\mathbf{x} + \sigma/\xi$.
`sigma`	This is a positive-only scalar or vector of scale parameters σ.
`xi`	This is a scalar or vector of shape parameters ξ.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{1}{\sigma}(1 + \xi\mathbf{z})^(- 1/\xi + 1)$ where $\mathbf{z} = \frac{\theta - \mu}{\sigma}$
- Inventor: Pickands (1975)
- Notation 1: $\theta \sim \mathcal{GPD}(\mu, \sigma, \xi)$
- Notation 2: $p(\theta) \sim \mathcal{GPD}(\theta|\mu, \sigma, \xi)$
- Parameter 1: location μ, where $\mu \leq \theta$ when $\xi \geq 0$, and $\mu \geq \theta + \sigma/\xi$ when $\xi < 0$
- Parameter 2: scale $\sigma > 0$
- Parameter 3: shape ξ
- Mean: $\mu + \frac{\sigma}{1-\xi}$ when $\xi < 1$
- Variance: $\frac{\sigma^2}{(1-\xi)^2(1-2\xi)}$ when $\xi < 0.5$
- Mode:

The generalized Pareto distribution (GPD) is a more flexible extension of the Pareto (`dpareto`) distribution. It is equivalent to the exponential distribution when both $\mu = 0$ and $\xi = 0$, and it is equivalent to the Pareto distribution when $\mu = \sigma/\xi$ and $\xi > 0$.

The GPD is often used to model the tails of another distribution, and the shape parameter ξ relates to tail-behavior. Distributions with tails that decrease exponentially are modeled with shape $\xi = 0$. Distributions with tails that decrease as a polynomial are modeled with a positive shape parameter. Distributions with finite tails are modeled with a negative shape parameter.

Value

dgpd gives the density, and rgpd generates random deviates.

References

Pickands J. (1975). "Statistical Inference Using Extreme Order Statistics". *The Annals of Statistics*, 3, p. 119–131.

See Also

dpareto

Examples

```
library(LaplacesDemon)
x <- dgpd(0,0,1,0,log=TRUE)
x <- rgpd(10,0,1,0)
```

---

dist.Generalized.Poisson

Generalized Poisson Distribution

Description

The density function is provided for the univariate, discrete, generalized Poisson distribution with location parameter λ and scale parameter ω.

Usage

```
dgpois(x, lambda=0, omega=0, log=FALSE)
```

Arguments

x	This is a vector of quantiles.
lambda	This is the parameter λ.
omega	This is the parameter ω, which should be in the interval [0,1) for positive counts.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Discrete Univariate
- Density: $p(\theta) = (1-\omega)\lambda \frac{[(1-\omega)\lambda+\omega\theta]^{\theta-1}}{\theta!} \exp -[(1-\omega)\lambda+\omega\theta]$
- Inventor: Consul (1989) and Ntzoufras et al. (2005)
- Notation 1: $\theta \sim \mathrm{GP}(\lambda, \omega)$
- Notation 2: $p(\theta) = \mathrm{GP}(\theta|\lambda, \omega)$

- Parameter 1: location parameter λ
- Parameter 2: scale parameter $\omega \in [0, 1)$
- Mean: $E(\theta) = \lambda$
- Variance: $var(\theta) = \lambda(1 - \omega)^{-2}$

The generalized Poisson distribution (Consul, 1989) is also called the Lagrangian Poisson distribution. The simple Poisson distribution is a special case of the generalized Poisson distribution. The generalized Poisson distribution is used in generalized Poisson regression as an extension of Poisson regression that accounts for overdispersion.

The `dgpois` function is parameterized according to Ntzoufras et al. (2005), which is easier to interpret and estimates better with MCMC.

Valid values for omega are in the interval [0,1) for positive counts. For $\omega = 0$, the generalized Poisson reduces to a simple Poisson with mean λ. Note that it is possible for $\omega < 0$, but this implies underdispersion in count data, which is uncommon. The `dgpois` function returns warnings or errors, so ω should be non-negative here.

The dispersion index (DI) is a variance-to-mean ratio, and is $DI = (1 - \omega)^{-2}$. A simple Poisson has DI=1. When DI is far from one, the assumption that the variance equals the mean of a simple Poisson is violated.

Value

`dgpois` gives the density.

References

Consul, P. (1989). '"Generalized Poisson Distribution: Properties and Applications". Marcel Decker: New York, NY.

Ntzoufras, I., Katsis, A., and Karlis, D. (2005). "Bayesian Assessment of the Distribution of Insurance Claim Counts using Reversible Jump MCMC", *North American Actuarial Journal*, 9, p. 90–108.

See Also

`dnbinom` and `dpois`.

Examples

```
library(LaplacesDemon)
y <- rpois(100, 5)
lambda <- rpois(100, 5)
x <- dgpois(y, lambda, 0.5)

#Plot Probability Functions
x <- seq(from=0, to=20, by=1)
plot(x, dgpois(x,1,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dlaplace(x,1,0.6), type="l", col="green")
lines(x, dlaplace(x,1,0.7), type="l", col="blue")
legend(2, 0.9, expression(paste(lambda==1, ", ", omega==0.5),
```

```
paste(lambda==1, ", ", omega==0.6), paste(lambda==1, ", ", omega==0.7)),
lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.HalfCauchy` *Half-Cauchy Distribution*

Description

These functions provide the density, distribution function, quantile function, and random generation for the half-Cauchy distribution.

Usage

```
dhalfcauchy(x, scale=25, log=FALSE)
phalfcauchy(q, scale=25)
qhalfcauchy(p, scale=25)
rhalfcauchy(n, scale=25)
```

Arguments

`x,q`	These are each a vector of quantiles.
`p`	This is a vector of probabilities.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`scale`	This is the scale parameter α, which must be positive.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{2\alpha}{\pi(\theta^2+\alpha^2)}, \quad \theta > 0$
- Inventor: Derived from Cauchy
- Notation 1: $\theta \sim \mathcal{HC}(\alpha)$
- Notation 2: $p(\theta) = \mathcal{HC}(\theta|\alpha)$
- Parameter 1: scale parameter $\alpha > 0$
- Mean: $E(\theta)$ = does not exist
- Variance: $var(\theta)$ = does not exist
- Mode: $mode(\theta) = 0$

The half-Cauchy distribution with scale $\alpha = 25$ is a recommended, default, weakly informative prior distribution for a scale parameter. Otherwise, the scale, α, is recommended to be set to be just a little larger than the expected standard deviation, as a weakly informative prior distribution on a standard deviation parameter.

The Cauchy distribution is known as a pathological distribution because its mean and variance are undefined, and it does not satisfy the central limit theorem.

Value

dhalfcauchy gives the density, phalfcauchy gives the distribution function, qhalfcauchy gives the quantile function, and rhalfcauchy generates random deviates.

See Also

dcauchy

Examples

```
library(LaplacesDemon)
x <- dhalfcauchy(1,25)
x <- phalfcauchy(1,25)
x <- qhalfcauchy(0.5,25)
x <- rhalfcauchy(1,25)

#Plot Probability Functions
x <- seq(from=0, to=20, by=0.1)
plot(x, dhalfcauchy(x,1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dhalfcauchy(x,5), type="l", col="green")
lines(x, dhalfcauchy(x,10), type="l", col="blue")
legend(2, 0.9, expression(alpha==1, alpha==5, alpha==10),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.HalfNormal *Half-Normal Distribution*

Description

These functions provide the density, distribution function, quantile function, and random generation for the half-normal distribution.

Usage

```
dhalfnorm(x, scale=sqrt(pi/2), log=FALSE)
phalfnorm(q, scale=sqrt(pi/2), lower.tail=TRUE, log.p=FALSE)
qhalfnorm(p, scale=sqrt(pi/2), lower.tail=TRUE, log.p=FALSE)
rhalfnorm(n, scale=sqrt(pi/2))
```

Arguments

x,q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
scale	This is the scale parameter σ, which must be positive.

log,log.p	Logical. If log=TRUE, then the logarithm of the density or result is returned.
lower.tail	Logical. If lower.tail=TRUE (default), probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{2\sigma}{\pi} \exp(-\frac{\theta^2\sigma^2}{\pi}), \quad \theta \geq 0$
- Inventor: Derived from the normal or Gaussian
- Notation 1: $\theta \sim \mathcal{HN}(\sigma)$
- Notation 2: $p(\theta) = \mathcal{HN}(\theta|\sigma)$
- Parameter 1: scale parameter $\sigma > 0$
- Mean: $E(\theta) = \frac{1}{\sigma}$
- Variance: $var(\theta) = \frac{\pi-2}{2\sigma^2}$
- Mode: $mode(\theta) = 0$

The half-normal distribution is recommended as a weakly informative prior distribution for a scale parameter that may be useful as an alternative to the half-Cauchy, half-t, or vague gamma.

Value

dhalfnorm gives the density, phalfnorm gives the distribution function, qhalfnorm gives the quantile function, and rhalfnorm generates random deviates.

See Also

dnorm, dnormp, and dnormv.

Examples

```
library(LaplacesDemon)
x <- dhalfnorm(1)
x <- phalfnorm(1)
x <- qhalfnorm(0.5)
x <- rhalfnorm(10)

#Plot Probability Functions
x <- seq(from=0.1, to=20, by=0.1)
plot(x, dhalfnorm(x,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dhalfnorm(x,0.5), type="l", col="green")
lines(x, dhalfnorm(x,1), type="l", col="blue")
legend(2, 0.9, expression(sigma==0.1, sigma==0.5, sigma==1),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Halft` *Half-t Distribution*

Description

These functions provide the density, distribution function, quantile function, and random generation for the half-t distribution.

Usage

```
dhalft(x, scale=25, nu=1, log=FALSE)
phalft(q, scale=25, nu=1)
qhalft(p, scale=25, nu=1)
rhalft(n, scale=25, nu=1)
```

Arguments

`x,q`	These are each a vector of quantiles.
`p`	This is a vector of probabilities.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`scale`	This is the scale parameter α, which must be positive.
`nu`	This is the scalar degrees of freedom parameter, which is usually represented as ν.
`log`	Logical. If `log=TRUE` then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = (1 + \frac{1}{\nu}(\theta/\alpha)^2)^{-(\nu+1)/2}, \quad \theta \geq 0$
- Inventor: Derived from the Student t
- Notation 1: $\theta \sim \mathcal{HT}(\alpha, \nu)$
- Notation 2: $p(\theta) = \mathcal{HT}(\theta|\alpha, \nu)$
- Parameter 1: scale parameter $\alpha > 0$
- Parameter 2: degrees of freedom parameter ν
- Mean: $E(\theta)$ = unknown
- Variance: $var(\theta)$ = unknown
- Mode: $mode(\theta) = 0$

The half-t distribution is derived from the Student t distribution, and is useful as a weakly informative prior distribution for a scale parameter. It is more adaptable than the default recommended half-Cauchy, though it may also be more difficult to estimate due to its additional degrees of freedom parameter, ν. When $\nu = 1$, the density is proportional to a proper half-Cauchy distribution.

When $\nu = -1$, the density becomes an improper, uniform prior distribution. For more information on propriety, see `is.proper`.

Wand et al. (2011) demonstrated that the half-t distribution may be represented as a scale mixture of inverse-gamma distributions. This representation is useful for conjugacy.

Value

`dhalft` gives the density, `phalft` gives the distribution function, `qhalft` gives the quantile function, and `rhalft` generates random deviates.

References

Wand, M.P., Ormerod, J.T., Padoan, S.A., and Fruhwirth, R. (2011). "Mean Field Variational Bayes for Elaborate Distributions". *Bayesian Analysis*, 6: p. 847–900.

See Also

`dhalfcauchy`, `dst`, `dt`, `dunif`, and `is.proper`.

Examples

```
library(LaplacesDemon)
x <- dhalft(1,25,1)
x <- phalft(1,25,1)
x <- qhalft(0.5,25,1)
x <- rhalft(10,25,1)

#Plot Probability Functions
x <- seq(from=0.1, to=20, by=0.1)
plot(x, dhalft(x,1,-1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dhalft(x,1,0.5), type="l", col="green")
lines(x, dhalft(x,1,500), type="l", col="blue")
legend(2, 0.9, expression(paste(alpha==1, ", ", nu==-1),
     paste(alpha==1, ", ", nu==0.5), paste(alpha==1, ", ", nu==500)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Horseshoe *Horseshoe Distribution*

Description

This is the density function and random generation from the horseshoe distribution.

Usage

```
dhs(x, lambda, tau, log=FALSE)
rhs(n, lambda, tau)
```

Arguments

n	This is the number of draws from the distribution.
x	This is a location vector at which to evaluate density.
lambda	This vector is a positive-only local parameter λ.
tau	This scalar is a positive-only global parameter τ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Multivariate Scale Mixture
- Density: (see below)
- Inventor: Carvalho et al. (2008)
- Notation 1: $\theta \sim \mathcal{HS}(\lambda, \tau)$
- Notation 2: $p(\theta) = \mathcal{HS}(\theta|\lambda, \tau)$
- Parameter 1: local scale $\lambda > 0$
- Parameter 2: global scale $\tau > 0$
- Mean: $E(\theta)$
- Variance: $var(\theta)$
- Mode: $mode(\theta)$

The horseshoe distribution (Carvalho et al., 2008) is a heavy-tailed mixture distribution that can be considered a variance mixture, and it is in the family of multivariate scale mixtures of normals.

The horseshoe distribution was proposed as a prior distribution, and recommended as a default choice for shrinkage priors in the presence of sparsity. Horseshoe priors are most appropriate in large-p models where dimension reduction is necessary to avoid overly complex models that predict poorly, and also perform well in estimating a sparse covariance matrix via Cholesky decomposition (Carvalho et al., 2009).

When the number of parameters in variable selection is assumed to be sparse, meaning that most elements are zero or nearly zero, a horseshoe prior is a desirable alternative to the Laplace-distributed parameters in the LASSO, or the parameterization in ridge regression. When the true value is far from zero, the horseshoe prior leaves the parameter unshrunk. Yet, the horseshoe prior is accurate in shrinking parameters that are truly zero or near-zero. Parameters near zero are shrunk more than parameters far from zero. Therefore, parameters far from zero experience less shrinkage and are closer to their true values. The horseshoe prior is valuable in discriminating signal from noise.

By replacing the Laplace-distributed parameters in LASSO with horseshoe-distributed parameters and including a global scale, the result is called horseshoe regression.

Value

dhs gives the density and rhs generates random deviates.

References

Carvalho, C.M., Polson, N.G., and Scott, J.G. (2008). "The Horseshoe Estimator for Sparse Signals". *Discussion Paper 2008-31*. Duke University Department of Statistical Science.

Carvalho, C.M., Polson, N.G., and Scott, J.G. (2009). "Handling Sparsity via the Horseshoe". *Journal of Machine Learning Research*, 5, p. 73–80.

See Also

dlaplace

Examples

```
library(LaplacesDemon)
x <- rnorm(100)
lambda <- rhalfcauchy(100, 5)
tau <- 5
x <- dhs(x, lambda, tau, log=TRUE)
x <- rhs(100, lambda=lambda, tau=tau)
plot(density(x))
```

---

dist.HuangWand *Huang-Wand Distribution*

Description

These are the density and random generation functions for the Huang-Wand prior distribution for a covariance matrix.

Usage

```
dhuangwand(x, nu=2, a, A, log=FALSE)
dhuangwandc(x, nu=2, a, A, log=FALSE)
rhuangwand(nu=2, a, A)
rhuangwandc(nu=2, a, A)
```

Arguments

x	This is a $k \times k$ positive-definite covariance matrix Σ for dhuangwand, or the Cholesky factor **U** of the covariance matrix for dhuangwandc.
nu	This is a scalar degrees of freedom parameter ν. The default is nu=2, which is an uninformative prior, resulting in marginal uniform distributions on the correlation matrix.
a	This is a positive-only vector of scale parameters a of length k.
A	This is a positive-only vector of of scale hyperparameters A of length k. Larger values result in a more uninformative prior. A default, uninformative prior is A=rep(1e6,k).
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \mathcal{W}^{-1}_{\nu+k-1}(2\nu diag(1/a))\mathcal{G}^{-1}(1/2, 1/A^2)$
- Inventor: Huang and Wand (2013)
- Notation 1: $\theta \sim \mathcal{HW}_\nu(\mathbf{a}, \mathbf{A})$
- Notation 2: $p(\theta) \sim \mathcal{HW}_\nu(\theta|\mathbf{a}, \mathbf{A})$
- Parameter 1: degrees of freedom ν
- Parameter 2: scale $a > 0$
- Parameter 3: scale $A > 0$
- Mean:
- Variance:
- Mode:

Huang and Wand (2013) proposed a prior distribution for a covariance matrix that uses a hierarchical inverse Wishart. This is a more flexible alternative to the inverse Wishart distribution, and the Huang-Wand prior retains conjugacy. The Cholesky parameterization is also provided here.

The Huang-Wand prior distribution alleviates two main limitations of an inverse Wishart distribution. First, the uncertainty in the diagonal variances of a covariance matrix that is inverse Wishart distributed is represented with only one degrees of freedom parameter, which may be too restrictive. The Huang-Wand prior overcomes this limitation. Second, the inverse Wishart distribution imposes a dependency between variance and correlation. The Huang-Wand prior lessens, but does not fully remove, this dependency.

The standard deviations of a Huang-Wand distributed covariance matrix are half-t distributed, as $\mathcal{HT}(\nu, \mathbf{A})$. This is in accord with modern assumptions about distributions of scale parameters, and is also useful for sparse covariance matrices.

The `rhuangwand` function allows either `a` or `A` to be missing. When `a` is missing, the covariance matrix is generated from the hyperparameters. When `A` is missing, the covariance matrix is generated from the parameters.

Value

`dhuangwand` and `dhuangwandc` give the density, and `rhuangwand` and `rhuangwandc` generate random deviates.

References

Huang, A., Wand, M., et al. (2013), "Simple Marginally Noninformative Prior Distributions for Covariance Matrices". *Bayesian Analysis*, 8, p. 439–452.

See Also

`dhalft` and `dinvwishart`

Examples

```
library(LaplacesDemon)
dhuangwand(diag(3), nu=2, a=runif(3), A=rep(1e6,3), log=TRUE)
rhuangwand(nu=2, A=rep(1e6, 3)) #Missing a
rhuangwand(nu=2, a=runif(3)) #Missing A
```

---

dist.Inverse.Beta *Inverse Beta Distribution*

Description

This is the density function and random generation from the inverse beta distribution.

Usage

```
dinvbeta(x, a, b, log=FALSE)
rinvbeta(n, a, b)
```

Arguments

n	This is the number of draws from the distribution.
x	This is a location vector at which to evaluate density.
a	This is the scalar shape parameter α.
b	This is the scalar shape parameter β
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\theta^{\alpha-1}(1+\theta)^{-\alpha-\beta}}{\beta(\alpha,\beta)}$
- Inventor: Dubey (1970)
- Notation 1: $\theta \sim \mathcal{B}^{-1}(\alpha, \beta)$
- Notation 2: $p(\theta) = \mathcal{B}^{-1}(\theta|\alpha, \beta)$
- Parameter 1: shape $\alpha > 0$
- Parameter 2: shape $\beta > 0$
- Mean: $E(\theta) = \frac{\alpha}{\beta-1}$, for $\beta > 1$
- Variance: $var(\theta) = \frac{\alpha(\alpha+\beta-1)}{(\beta-1)^2(\beta-2)}$
- Mode: $mode(\theta) = \frac{\alpha-1}{\beta+1}$

The inverse-beta, also called the beta prime distribution, applies to variables that are continuous and positive. The inverse beta is the conjugate prior distribution of a parameter of a Bernoulli distribution expressed in odds.

The inverse-beta distribution has also been extended to the generalized beta prime distribution, though it is not (yet) included here.

Value

`dinvbeta` gives the density and `rinvbeta` generates random deviates.

References

Dubey, S.D. (1970). "Compound Gamma, Beta and F Distributions". *Metrika*, 16, p. 27–31.

See Also

`dbeta`

Examples

```
library(LaplacesDemon)
x <- dinvbeta(5:10, 2, 3)
x <- rinvbeta(10, 2, 3)

#Plot Probability Functions
x <- seq(from=0.1, to=20, by=0.1)
plot(x, dinvbeta(x,2,2), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dinvbeta(x,2,3), type="l", col="green")
lines(x, dinvbeta(x,3,2), type="l", col="blue")
legend(2, 0.9, expression(paste(alpha==2, ", ", beta==2),
     paste(alpha==2, ", ", beta==3), paste(alpha==3, ", ", beta==2)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Inverse.ChiSquare

(Scaled) Inverse Chi-Squared Distribution

Description

This is the density function and random generation for the (scaled) inverse chi-squared distribution.

Usage

```
dinvchisq(x, df, scale, log=FALSE)
rinvchisq(n, df, scale=1/df)
```

Arguments

`x`	This is a vector of quantiles.
`n`	This is the number of observations. If `length(n) > 1`, then the length is taken to be the number required.
`df`	This is the degrees of freedom parameter, usually represented as ν.
`scale`	This is the scale parameter, usually represented as λ.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density:

$$p(\theta) = \frac{\nu/2^{\nu/2}}{\Gamma(\nu/2)}\lambda^{\nu}\frac{1}{\theta}^{\nu/2+1}\exp(-\frac{\nu\lambda^2}{2\theta}), \theta \geq 0$$

- Inventor: Derived from the chi-squared distribution
- Notation 1: $\theta \sim \chi^{-2}(\nu, \lambda)$
- Notation 2: $p(\theta) = \chi^{-2}(\theta|\nu, \lambda)$
- Parameter 1: degrees of freedom parameter $\nu > 0$
- Parameter 2: scale parameter λ
- Mean: $E(\theta)$ = unknown
- Variance: $var(\theta)$ = unknown
- Mode: $mode(\theta) =$

The inverse chi-squared distribution, also called the inverted chi-square distribution, is the multiplicate inverse of the chi-squared distribution. If x has the chi-squared distribution with ν degrees of freedom, then $1/x$ has the inverse chi-squared distribution with ν degrees of freedom, and ν/x has the inverse chi-squared distribution with ν degrees of freedom.

These functions are similar to those in the GeoR package.

Value

`dinvchisq` gives the density and `rinvchisq` generates random deviates.

See Also

`dchisq`

Examples

```
library(LaplacesDemon)
x <- dinvchisq(1,1,1)
x <- rinvchisq(10,1)

#Plot Probability Functions
x <- seq(from=0.1, to=5, by=0.01)
plot(x, dinvchisq(x,0.5,1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dinvchisq(x,1,1), type="l", col="green")
lines(x, dinvchisq(x,5,1), type="l", col="blue")
legend(3, 0.9, expression(paste(nu==0.5, ", ", lambda==1),
     paste(nu==1, ", ", lambda==1), paste(nu==5, ", ", lambda==1)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Inverse.Gamma` *Inverse Gamma Distribution*

Description

This is the density function and random generation from the inverse gamma distribution.

Usage

```
dinvgamma(x, shape=1, scale=1, log=FALSE)
rinvgamma(n, shape=1, scale=1)
```

Arguments

`n`	This is the number of draws from the distribution.
`x`	This is the scalar location to evaluate density.
`shape`	This is the scalar shape parameter α, which defaults to one.
`scale`	This is the scalar scale parameter β, which defaults to one.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\beta^\alpha}{\Gamma(\alpha)}\theta^{-(\alpha+1)}\exp(-\frac{\beta}{\theta}), \quad \theta > 0$
- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{G}^{-1}(\alpha, \beta)$
- Notation 2: $p(\theta) = \mathcal{G}^{-1}(\theta|\alpha, \beta)$
- Parameter 1: shape $\alpha > 0$
- Parameter 2: scale $\beta > 0$
- Mean: $E(\theta) = \frac{\beta}{\alpha-1}$, for $\alpha > 1$
- Variance: $var(\theta) = \frac{\beta^2}{(\alpha-1)^2(\alpha-2)}, \alpha > 2$
- Mode: $mode(\theta) = \frac{\beta}{\alpha+1}$

The inverse-gamma is the conjugate prior distribution for the normal or Gaussian variance, and has been traditionally specified as a vague prior in that application. The density is always finite; its integral is finite if $\alpha > 0$. Prior information decreases as $\alpha, \beta \to 0$.

These functions are similar to those in the `MCMCpack` package.

Value

`dinvgamma` gives the density and `rinvgamma` generates random deviates. The parameterization is consistent with the Gamma Distribution in the stats package.

See Also

dgamma, dnorm, dnormp, and dnormv.

Examples

```
library(LaplacesDemon)
x <- dinvgamma(4.3, 1.1)
x <- rinvgamma(10, 3.3)

#Plot Probability Functions
x <- seq(from=0.1, to=20, by=0.1)
plot(x, dinvgamma(x,1,1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dinvgamma(x,1,0.6), type="l", col="green")
lines(x, dinvgamma(x,0.6,1), type="l", col="blue")
legend(2, 0.9, expression(paste(alpha==1, ", ", beta==1),
     paste(alpha==1, ", ", beta==0.6), paste(alpha==0.6, ", ", beta==1)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Inverse.Gaussian *Inverse Gaussian Distribution*

Description

This is the density function and random generation from the inverse gaussian distribution.

Usage

```
dinvgaussian(x, mu, lambda, log=FALSE)
rinvgaussian(n, mu, lambda)
```

Arguments

n	This is the number of draws from the distribution.
x	This is the scalar location to evaluate density.
mu	This is the mean parameter, μ.
lambda	This is the inverse-variance parameter, λ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\lambda}{(2\pi\theta^3)^{1/2}} \exp(-\frac{\lambda(\theta-\mu)^2}{2\mu^2\theta}), \theta > 0$
- Inventor: Schrodinger (1915)
- Notation 1: $\theta \sim \mathcal{N}^{-1}(\mu, \lambda)$

- Notation 2: $p(\theta) = \mathcal{N}^{-1}(\theta|\mu, \lambda)$
- Parameter 1: shape $\mu > 0$
- Parameter 2: scale $\lambda > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \frac{\mu^3}{\lambda}$
- Mode: $mode(\theta) = \mu((1 + \frac{9\mu^2}{4\lambda^2})^{1/2} - \frac{3\mu}{2\lambda})$

The inverse-Gaussian distribution, also called the Wald distribution, is used when modeling dependent variables that are positive and continuous. When $\lambda \to \infty$ (or variance to zero), the inverse-Gaussian distribution becomes similar to a normal (Gaussian) distribution. The name, inverse-Gaussian, is misleading, because it is not the inverse of a Gaussian distribution, which is obvious from the fact that θ must be positive.

Value

`dinvgaussian` gives the density and `rinvgaussian` generates random deviates.

References

Schrodinger E. (1915). "Zur Theorie der Fall-und Steigversuche an Teilchenn mit Brownscher Bewegung". *Physikalische Zeitschrift*, 16, p. 289–295.

See Also

`dnorm`, `dnormp`, and `dnormv`.

Examples

```
library(LaplacesDemon)
x <- dinvgaussian(2, 1, 1)
x <- rinvgaussian(10, 1, 1)

#Plot Probability Functions
x <- seq(from=1, to=20, by=0.1)
plot(x, dinvgaussian(x,1,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dinvgaussian(x,1,1), type="l", col="green")
lines(x, dinvgaussian(x,1,5), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==1, ", ", sigma==0.5),
     paste(mu==1, ", ", sigma==1), paste(mu==1, ", ", sigma==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Inverse.Matrix.Gamma` *Inverse Matrix Gamma Distribution*

Description

This function provides the density for the inverse matrix gamma distribution.

Usage

```
dinvmatrixgamma(X, alpha, beta, Psi, log=FALSE)
```

Arguments

`X`	This is a $k \times k$ positive-definite covariance matrix.
`alpha`	This is a scalar shape parameter (the degrees of freedom), α.
`beta`	This is a scalar, positive-only scale parameter, β.
`Psi`	This is a $k \times k$ positive-definite scale matrix.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate Matrix
- Density: $p(\theta) = \frac{|\Psi|^\alpha}{\beta^{k\alpha}\Gamma_k(\alpha)}|\theta|^{-\alpha-(k+1)/2}\exp(tr(-\frac{1}{\beta}\Psi\theta^{-1}))$
- Inventors: Unknown
- Notation 1: $\theta \sim \mathcal{IMG}_k(\alpha, \beta, \Psi)$
- Notation 2: $p(\theta) = \mathcal{IMG}_k(\theta|\alpha, \beta, \Psi)$
- Parameter 1: shape $\alpha > 2$
- Parameter 2: scale $\beta > 0$
- Parameter 3: positive-definite $k \times k$ scale matrix Ψ
- Mean:
- Variance:
- Mode:

The inverse matrix gamma (IMG), also called the inverse matrix-variate gamma, distribution is a generalization of the inverse gamma distribution to positive-definite matrices. It is a more general and flexible version of the inverse Wishart distribution (`dinvwishart`), and is a conjugate prior of the covariance matrix of a multivariate normal distribution (`dmvn`) and matrix normal distribution (`dmatrixnorm`).

The compound distribution resulting from compounding a matrix normal with an inverse matrix gamma prior over the covariance matrix is a generalized matrix t-distribution.

The inverse matrix gamma distribution is identical to the inverse Wishart distribution when $\alpha = \nu/2$ and $\beta = 2$.

Value

dinvmatrixgamma gives the density.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dinvgamma dmatrixnorm, dmvn, and dinvwishart

Examples

```
library(LaplacesDemon)
k <- 10
dinvmatrixgamma(X=diag(k), alpha=(k+1)/2, beta=2, Psi=diag(k), log=TRUE)
dinvwishart(Sigma=diag(k), nu=k+1, S=diag(k), log=TRUE)
```

---

dist.Inverse.Wishart *Inverse Wishart Distribution*

Description

These functions provide the density and random number generation for the inverse Wishart distribution.

Usage

```
dinvwishart(Sigma, nu, S, log=FALSE)
rinvwishart(nu, S)
```

Arguments

Sigma	This is the symmetric, positive-definite $k \times k$ matrix Σ.
nu	This is the scalar degrees of freedom, ν.
S	This is the symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2^{\nu k/2} \pi^{k(k-1)/4} \prod_{i=1}^{k} \Gamma(\frac{\nu+1-i}{2}))^{-1} |\mathbf{S}|^{nu/2} |\Omega|^{-(nu-k-1)/2} \exp(-\frac{1}{2} tr(\mathbf{S}\Omega^{-1}))$
- Inventor: John Wishart (1928)
- Notation 1: $\Sigma \sim \mathcal{W}_{\nu}^{-1}(\mathbf{S}^{-1})$
- Notation 2: $p(\Sigma) = \mathcal{W}_{\nu}^{-1}(\Sigma|\mathbf{S}^{-1})$

- Parameter 1: degrees of freedom ν
- Parameter 2: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Mean: $E(\Sigma) = \frac{\mathbf{S}}{\nu-k-1}$
- Variance:
- Mode: $mode(\Sigma) = \frac{\mathbf{S}}{\nu+k+1}$

The inverse Wishart distribution is a probability distribution defined on real-valued, symmetric, positive-definite matrices, and is used as the conjugate prior for the covariance matrix, Σ, of a multivariate normal distribution. The inverse-Wishart density is always finite, and the integral is always finite. A degenerate form occurs when $\nu < k$.

When applicable, the alternative Cholesky parameterization should be preferred. For more information, see dinvwishartc.

The inverse Wishart prior lacks flexibility, having only one parameter, ν, to control the variability for all $k(k+1)/2$ elements. Popular choices for the scale matrix $\mathbf{S}$ include an identity matrix or sample covariance matrix. When the model sample size is small, the specification of the scale matrix can be influential.

The inverse Wishart distribution has a dependency between variance and correlation, although its relative for a precision matrix (inverse covariance matrix), the Wishart distribution, does not have this dependency. This relationship becomes weaker with more degrees of freedom.

Due to these limitations (lack of flexibility, and dependence between variance and correlation), alternative distributions have been developed. Alternative distributions that are available here include Huang-Wand (dhuangwand), inverse matrix gamma (dinvmatrixgamma), Scaled Inverse Wishart (dsiw), and Yang-Berger (dyangberger).

These functions are parameterized as per Gelman et al. (2004).

Value

`dinvwishart` gives the density and `rinvwishart` generates random deviates.

References

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). "Bayesian Data Analysis, Texts in Statistical Science, 2nd ed.". Chapman and Hall, London.

Wishart, J. (1928). "The Generalised Product Moment Distribution in Samples from a Normal Multivariate Population". *Biometrika*, 20A(1-2), p. 32–52.

See Also

dhuangwand, dinvmatrixgamma, dinvwishartc, dmvn, dsiw, dwishart, and dyangberger.

Examples

```
library(LaplacesDemon)
x <- dinvwishart(matrix(c(2,-.3,-.3,4),2,2), 3, matrix(c(1,.1,.1,1),2,2))
x <- rinvwishart(3, matrix(c(1,.1,.1,1),2,2))
```

`dist.Inverse.Wishart.Cholesky` *Inverse Wishart Distribution: Cholesky Parameterization*

Description

These functions provide the density and random number generation for the inverse Wishart distribution with the Cholesky parameterization.

Usage

```
dinvwishartc(U, nu, S, log=FALSE)
rinvwishartc(nu, S)
```

Arguments

`U`	This is the upper-triangular $k \times k$ matrix for the Cholesky factor $\mathbf{U}$ of covariance matrix Σ.
`nu`	This is the scalar degrees of freedom, ν.
`S`	This is the symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2^{\nu k/2}\pi^{k(k-1)/4}\prod_{i=1}^{k}\Gamma(\frac{\nu+1-i}{2}))^{-1}|\mathbf{S}|^{nu/2}|\Omega|^{-(nu-k-1)/2}\exp(-\frac{1}{2}tr(\mathbf{S}\Omega^{-1}))$
- Inventor: John Wishart (1928)
- Notation 1: $\Sigma \sim \mathcal{W}_{\nu}^{-1}(\mathbf{S}^{-1})$
- Notation 2: $p(\Sigma) = \mathcal{W}_{\nu}^{-1}(\Sigma|\mathbf{S}^{-1})$
- Parameter 1: degrees of freedom ν
- Parameter 2: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Mean: $E(\Sigma) = \frac{\mathbf{S}}{\nu-k-1}$
- Variance:
- Mode: $mode(\Sigma) = \frac{\mathbf{S}}{\nu+k+1}$

The inverse Wishart distribution is a probability distribution defined on real-valued, symmetric, positive-definite matrices, and is used as the conjugate prior for the covariance matrix, Σ, of a multivariate normal distribution. In this parameterization, Σ has been decomposed to the upper-triangular Cholesky factor $\mathbf{U}$, as per `chol`. The inverse-Wishart density is always finite, and the integral is always finite. A degenerate form occurs when $\nu < k$.

In practice, $\mathbf{U}$ is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with `dinvwishart`, `dinvwishartc`

must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the covariance matrix to positive-semidefiniteness, which overall is slower. Compared with `rinvwishart`, `rinvwishartc` must additionally calculate a Cholesky decomposition, and is therefore slower.

The inverse Wishart prior lacks flexibility, having only one parameter, ν, to control the variability for all $k(k+1)/2$ elements. Popular choices for the scale matrix $\mathbf{S}$ include an identity matrix or sample covariance matrix. When the model sample size is small, the specification of the scale matrix can be influential.

The inverse Wishart distribution has a dependency between variance and correlation, although its relative for a precision matrix (inverse covariance matrix), the Wishart distribution, does not have this dependency. This relationship becomes weaker with more degrees of freedom.

Due to these limitations (lack of flexibility, and dependence between variance and correlation), alternative distributions have been developed. Alternative distributions that are available here include the inverse matrix gamma (dinvmatrixgamma), Scaled Inverse Wishart (dsiw) and Huang-Wand (dhuangwand). Huang-Wand is recommended.

Value

`dinvwishartc` gives the density and `rinvwishartc` generates random deviates.

References

Wishart, J. (1928). "The Generalised Product Moment Distribution in Samples from a Normal Multivariate Population". *Biometrika*, 20A(1-2), p. 32–52.

See Also

chol, Cov2Prec, dhuangwand, dinvmatrixgamma, dmvn, dmvnc, dmvtc, dsiw, dwishart, dwishartc, and dyangbergerc.

Examples

```
library(LaplacesDemon)
Sigma <- matrix(c(2,-.3,-.3,4),2,2)
U <- chol(Sigma)
x <- dinvwishartc(U, 3, matrix(c(1,.1,.1,1),2,2))
x <- rinvwishartc(3, matrix(c(1,.1,.1,1),2,2))
```

---

dist.Laplace *Laplace Distribution: Univariate Symmetric*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, symmetric, Laplace distribution with location parameter μ and scale parameter λ.

Usage

```
dlaplace(x, location=0, scale=1, log=FALSE)
plaplace(q, location=0, scale=1)
qlaplace(p, location=0, scale=1)
rlaplace(n, location=0, scale=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
location	This is the location parameter μ.
scale	This is the scale parameter λ, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{1}{2\lambda} \exp(-\frac{|\theta-\mu|}{\lambda})$
- Inventor: Pierre-Simon Laplace (1774)
- Notation 1: $\theta \sim \text{Laplace}(\mu, \lambda)$
- Notation 2: $\theta \sim \mathcal{L}(\mu, \lambda)$
- Notation 3: $p(\theta) = \text{Laplace}(\theta|\mu, \lambda)$
- Notation 4: $p(\theta) = \mathcal{L}(\theta|\mu, \lambda)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\lambda > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = 2\lambda^2$
- Mode: $mode(\theta) = \mu$

The Laplace distribution (Laplace, 1774) is also called the double exponential distribution, because it looks like two exponential distributions back to back with respect to location μ. It is also called the "First Law of Laplace", just as the normal distribution is referred to as the "Second Law of Laplace". The Laplace distribution is symmetric with respect to μ, though there are asymmetric versions of the Laplace distribution. The PDF of the Laplace distribution is reminiscent of the normal distribution; however, whereas the normal distribution is expressed in terms of the squared difference from the mean μ, the Laplace density is expressed in terms of the absolute difference from the mean, μ. Consequently, the Laplace distribution has fatter tails than the normal distribution. It has been argued that the Laplace distribution fits most things in nature better than the normal distribution.

There are many extensions to the Laplace distribution, such as the asymmetric Laplace, asymmetric log-Laplace, Laplace (re-parameterized for precision), log-Laplace, multivariate Laplace, and skew-Laplace, among many more.

These functions are similar to those in the VGAM package.

Value

dlaplace gives the density, plaplace gives the distribution function, qlaplace gives the quantile function, and rlaplace generates random deviates.

References

Laplace, P. (1774). "Memoire sur la Probabilite des Causes par les Evenements." l'Academie Royale des Sciences, 6, 621–656. English translation by S.M. Stigler in 1986 as "Memoir on the Probability of the Causes of Events" in *Statistical Science*, 1(3), p. 359–378.

See Also

dalaplace, dallaplace, dexp, dlaplacep, dllaplace, dmvl, dnorm, dnormp, dnormv, dsdlaplace, and dslaplace.

Examples

```
library(LaplacesDemon)
x <- dlaplace(1,0,1)
x <- plaplace(1,0,1)
x <- qlaplace(0.5,0,1)
x <- rlaplace(100,0,1)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dlaplace(x,0,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dlaplace(x,0,1), type="l", col="green")
lines(x, dlaplace(x,0,2), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", lambda==0.5),
     paste(mu==0, ", ", lambda==1), paste(mu==0, ", ", lambda==2)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Laplace.Mixture *Mixture of Laplace Distributions*

Description

These functions provide the density, cumulative, and random generation for the mixture of univariate Laplace distributions with probability p, location μ and scale σ.

Usage

```
dlaplacem(x, p, location, scale, log=FALSE)
plaplacem(q, p, location, scale)
rlaplacem(n, p, location, scale)
```

Arguments

x,q	This is vector of values at which the density will be evaluated.
p	This is a vector of length M of probabilities for M components. The sum of the vector must be one.
n	This is the number of observations, which must be a positive integer that has length 1.
location	This is a vector of length M that is the location parameter μ.
scale	This is a vector of length M that is the scale parameter σ, which must be positive.
log	Logical. If TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \sum p_i \mathcal{L}(\mu_i, \sigma_i)$
- Inventor: Unknown
- Notation 1: $\theta \sim \mathcal{L}(\mu, \sigma)$
- Notation 2: $p(\theta) = \mathcal{L}(\theta|\mu, \sigma)$
- Parameter 1: location parameters μ
- Parameter 2: scale parameters $\sigma > 0$
- Mean: $E(\theta) = \sum p_i \mu_i$
- Variance:
- Mode:

A mixture distribution is a probability distribution that is a combination of other probability distributions, and each distribution is called a mixture component, or component. A probability (or weight) exists for each component, and these probabilities sum to one. A mixture distribution (though not these functions here in particular) may contain mixture components in which each component is a different probability distribution. Mixture distributions are very flexible, and are often used to represent a complex distribution with an unknown form. When the number of mixture components is unknown, Bayesian inference is the only sensible approach to estimation.

A Laplace mixture distribution is a combination of Laplace probability distributions.

One of many applications of Laplace mixture distributions is the Laplace Mixture Model (LMM).

Value

dlaplacem gives the density, plaplacem returns the CDF, and rlaplacem generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

ddirichlet and dlaplace.

Examples

```
library(LaplacesDemon)
p <- c(0.3,0.3,0.4)
mu <- c(-5, 1, 5)
sigma <- c(1,2,1)
x <- seq(from=-10, to=10, by=0.1)
plot(x, dlaplacem(x, p, mu, sigma, log=FALSE), type="l") #Density
plot(x, plaplacem(x, p, mu, sigma), type="l") #CDF
plot(density(rlaplacem(10000, p, mu, sigma))) #Random Deviates
```

`dist.Laplace.Precision`

Laplace Distribution: Precision Parameterization

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, symmetric, Laplace distribution with location parameter μ and precision parameter τ, which is the inverse of the usual scale parameter, λ.

Usage

```
dlaplacep(x, mu=0, tau=1, log=FALSE)
plaplacep(q, mu=0, tau=1)
qlaplacep(p, mu=0, tau=1)
rlaplacep(n, mu=0, tau=1)
```

Arguments

`x, q`	These are each a vector of quantiles.
`p`	This is a vector of probabilities.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`mu`	This is the location parameter μ.
`tau`	This is the precision parameter τ, which must be positive.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\tau}{2} \exp(-\tau|\theta - \mu|)$
- Inventor: Pierre-Simon Laplace (1774)
- Notation 1: $\theta \sim \text{Laplace}(\mu, \tau^{-1})$
- Notation 2: $\theta \sim \mathcal{L}(\mu, \tau^{-1})$

- Notation 3: $p(\theta) = \text{Laplace}(\mu, \tau^{-1})$
- Notation 4: $p(\theta) = \mathcal{L}(\theta|\mu, \tau^{-1})$
- Parameter 1: location parameter μ
- Parameter 2: precision parameter $\tau > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = 2\tau^{-2}$
- Mode: $mode(\theta) = \mu$

The Laplace distribution is also called the double exponential distribution, because it looks like two exponential distributions back to back with respect to location μ. It is also called the "First Law of Laplace", just as the normal distribution is referred to as the "Second Law of Laplace". The Laplace distribution is symmetric with respect to μ, though there are asymmetric versions of the Laplace distribution. The PDF of the Laplace distribution is reminiscent of the normal distribution; however, whereas the normal distribution is expressed in terms of the squared difference from the mean μ, the Laplace density is expressed in terms of the absolute difference from the mean, μ. Consequently, the Laplace distribution has fatter tails than the normal distribution. It has been argued that the Laplace distribution fits most things in nature better than the normal distribution. Elsewhere, there are a large number of extensions to the Laplace distribution, including asymmetric versions and multivariate versions, among many more. These functions provide the precision parameterization for convenience and familiarity in Bayesian inference.

Value

`dlaplacep` gives the density, `plaplacep` gives the distribution function, `qlaplacep` gives the quantile function, and `rlaplacep` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dalaplace`, `dexp`, `dlaplace`, `dmvl`, `dnorm`, `dnormp`, and `dnormv`.

Examples

```
library(LaplacesDemon)
x <- dlaplacep(1,0,1)
x <- plaplacep(1,0,1)
x <- qlaplacep(0.5,0,1)
x <- rlaplacep(100,0,1)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dlaplacep(x,0,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dlaplacep(x,0,1), type="l", col="green")
lines(x, dlaplacep(x,0,2), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", tau==0.5),
```

```
paste(mu==0, ", ", tau==1), paste(mu==0, ", ", tau==2)),
lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.LASSO` *LASSO Distribution*

Description

These functions provide the density and random generation for the Bayesian LASSO prior distribution.

Usage

```
dlasso(x, sigma, tau, lambda, a=1, b=1, log=FALSE)
rlasso(n, sigma, tau, lambda, a=1, b=1)
```

Arguments

`x`	This is a location vector of length J at which to evaluate density.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`sigma`	This is a positive-only scalar hyperparameter σ, which is also the residual standard deviation.
`tau`	This is a positive-only vector of hyperparameters, τ, of length J regarding local sparsity.
`lambda`	This is a positive-only scalar hyperhyperparameter, λ, of global sparsity.
`a, b`	These are positive-only scalar hyperhyperhyperparameters for gamma distributed λ.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Multivariate Scale Mixture
- Density: $p(\theta) \sim \mathcal{N}_k(0, \sigma^2 diag(\tau^2))(\frac{1}{sigma^2})\mathcal{EXP}(\frac{\lambda^2}{2})\mathcal{G}(a,b)$
- Inventor: Parks and Casella (2008)
- Notation 1: $\theta \sim \mathcal{LASSO}(\sigma, \tau, \lambda, a, b)$
- Notation 2: $p(\theta) = \mathcal{LASSO}(\theta|\sigma, \tau, \lambda, a, b)$
- Parameter 1: hyperparameter global scale $\sigma > 0$
- Parameter 2: hyperparameter local scale $\tau > 0$
- Parameter 3: hyperhyperparameter global scale $\lambda > 0$
- Parameter 4: hyperhyperhyperparameter scale $a > 0$
- Parameter 5: hyperhyperhyperparameter scale $b > 0$
- Mean: $E(\theta)$

- Variance:
- Mode:

The Bayesian LASSO distribution (Parks and Casella, 2008) is a heavy-tailed mixture distribution that can be considered a variance mixture, and it is in the family of multivariate scale mixtures of normals.

The LASSO distribution was proposed as a prior distribution, as a Bayesian version of the frequentist LASSO, introduced by Tibshirani (1996). It is applied as a shrinkage prior in the presence of sparsity for J regression effects. LASSO priors are most appropriate in large-dimensional models where dimension reduction is necessary to avoid overly complex models that predict poorly.

The Bayesian LASSO results in regression effects that are a compromise between regression effects in the frequentist LASSO and ridge regression. The Bayesian LASSO applies more shrinkage to weak regression effects than ridge regression.

The Bayesian LASSO is an alternative to horseshoe regression and ridge regression.

Value

`dlasso` gives the density and `rlasso` generates random deviates.

References

Park, T. and Casella, G. (2008). "The Bayesian Lasso". *Journal of the American Statistical Association*, 103, p. 672–680.

Tibshirani, R. (1996). "Regression Shrinkage and Selection via the Lasso". *Journal of the Royal Statistical Society*, Series B, 58, p. 267–288.

See Also

`dhs`

Examples

```
library(LaplacesDemon)
x <- rnorm(100)
sigma <- rhalfcauchy(1, 5)
tau <- rhalfcauchy(100, 5)
lambda <- rhalfcauchy(1, 5)
x <- dlasso(x, sigma, tau, lambda, log=TRUE)
x <- rlasso(length(tau), sigma, tau, lambda)
```

---

dist.Log.Laplace *Log-Laplace Distribution: Univariate Symmetric*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, symmetric, log-Laplace distribution with location parameter `location` and scale parameter `scale`.

Usage

```
dllaplace(x, location=0, scale=1, log=FALSE)
pllaplace(q, location=0, scale=1)
qllaplace(p, location=0, scale=1)
rllaplace(n, location=0, scale=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
location	This is the location parameter μ.
scale	This is the scale parameter λ, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density 1: $p(\theta) = \frac{(\sqrt{2}/\lambda)^2}{2(\sqrt{2}/\lambda)} \exp(-(\sqrt{2}/\lambda)(\theta - \mu)), \theta \geq \exp(\mu)$
- Density 2: $p(\theta) = \frac{(\sqrt{2}/\lambda)^2}{2(\sqrt{2}/\lambda)} \exp((\sqrt{2}/\lambda)(\theta - \mu)), \theta < \exp(\mu)$
- Inventor: Pierre-Simon Laplace
- Notation 1: $\theta \sim \mathcal{LL}(\mu, \lambda)$
- Notation 2: $p(\theta) = \mathcal{LL}(\theta|\mu, \lambda)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\lambda > 0$
- Mean: $E(\theta) =$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) =$

The univariate, symmetric log-Laplace distribution is derived from the Laplace distribution. Multivariate and asymmetric versions also exist.

These functions are similar to those in the VGAM package.

Value

dllaplace gives the density, pllaplace gives the distribution function, qllaplace gives the quantile function, and rllaplace generates random deviates.

References

Kozubowski, T. J. and Podgorski, K. (2003). "Log-Laplace Distributions". *International Mathematical Journal*, 3, p. 467–495.

See Also

dalaplace, dallaplace, dexp, dlaplace, dlaplacep, dmvl, dnorm, dnormp, and dnormv.

Examples

```
library(LaplacesDemon)
x <- dllaplace(1,0,1)
x <- pllaplace(1,0,1)
x <- qllaplace(0.5,0,1)
x <- rllaplace(100,0,1)

#Plot Probability Functions
x <- seq(from=0.1, to=20, by=0.1)
plot(x, dllaplace(x,0,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dllaplace(x,0,0.5), type="l", col="green")
lines(x, dllaplace(x,0,1.5), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", lambda==0.1),
     paste(mu==0, ", ", lambda==0.5), paste(mu==0, ", ", lambda==1.5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Log.Normal.Precision

Log-Normal Distribution: Precision Parameterization

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate log-normal distribution with mean μ and precision τ.

Usage

```
dlnormp(x, mu, tau=NULL, var=NULL, log=FALSE)
plnormp(q, mu, tau, lower.tail=TRUE, log.p=FALSE)
qlnormp(p, mu, tau, lower.tail=TRUE, log.p=FALSE)
rlnormp(n, mu, tau=NULL, var=NULL)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the mean parameter μ.
tau	This is the precision parameter τ, which must be positive. Tau and var cannot be used together

`var`	This is the variance parameter, which must be positive. Tau and var cannot be used together
`log, log.p`	Logical. If `TRUE`, then probabilities p are given as $\log(p)$.
`lower.tail`	Logical. If `TRUE` (default), then probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \sqrt{\frac{\tau}{2\pi}} \frac{1}{\theta} \exp(-\frac{\tau}{2}(\log(\theta - \mu))^2)$
- Inventor: Carl Friedrich Gauss or Abraham De Moivre
- Notation 1: $\theta \sim \text{Log}-\mathcal{N}(\mu, \tau^{-1})$
- Notation 2: $p(\theta) = \text{Log}-\mathcal{N}(\theta|\mu, \tau^{-1})$
- Parameter 1: mean parameter μ
- Parameter 2: precision parameter $\tau > 0$
- Mean: $E(\theta) = \exp(\mu + \tau^{-1}/2)$
- Variance: $var(\theta) = (\exp(\tau^{-1}) - 1)\exp(2\mu + \tau^{-1})$
- Mode: $mode(\theta) = \exp(\mu - \tau^{-1})$

The log-normal distribution, also called the Galton distribution, is applied to a variable whose logarithm is normally-distributed. The distribution is usually parameterized with mean and variance, or in Bayesian inference, with mean and precision, where precision is the inverse of the variance. In contrast, `Base R` parameterizes the log-normal distribution with the mean and standard deviation. These functions provide the precision parameterization for convenience and familiarity.

A flat distribution is obtained in the limit as $\tau \to 0$.

These functions are similar to those in `base R`.

Value

`dlnormp` gives the density, `plnormp` gives the distribution function, `qlnormp` gives the quantile function, and `rlnormp` generates random deviates.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`dnorm`, `dnormp`, `dnormv`, and `prec2var`.

Examples

```
library(LaplacesDemon)
x <- dlnormp(1,0,1)
x <- plnormp(1,0,1)
x <- qlnormp(0.5,0,1)
x <- rlnormp(100,0,1)
```

```
#Plot Probability Functions
x <- seq(from=0.1, to=3, by=0.01)
plot(x, dlnormp(x,0,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dlnormp(x,0,1), type="l", col="green")
lines(x, dlnormp(x,0,5), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", tau==0.1),
     paste(mu==0, ", ", tau==1), paste(mu==0, ", ", tau==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Matrix.Gamma` *Matrix Gamma Distribution*

Description

This function provides the density for the matrix gamma distribution.

Usage

```
dmatrixgamma(X, alpha, beta, Sigma, log=FALSE)
```

Arguments

`X`	This is a $k \times k$ positive-definite precision matrix.
`alpha`	This is a scalar shape parameter (the degrees of freedom), α.
`beta`	This is a scalar, positive-only scale parameter, β.
`Sigma`	This is a $k \times k$ positive-definite scale matrix.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate Matrix
- Density: $p(\theta) = \frac{|\Sigma|^{-\alpha}}{\beta^{k\alpha}\Gamma_k(\alpha)}|\theta|^{\alpha-(k+1)/2}\exp(tr(-\frac{1}{\beta}\Sigma^{-1}\theta))$
- Inventors: Unknown
- Notation 1: $\theta \sim \mathcal{MG}_k(\alpha, \beta, \Sigma)$
- Notation 2: $p(\theta) = \mathcal{MG}_k(\theta|\alpha, \beta, \Sigma)$
- Parameter 1: shape $\alpha > 2$
- Parameter 2: scale $\beta > 0$
- Parameter 3: positive-definite $k \times k$ scale matrix Σ
- Mean:
- Variance:

- Mode:

The matrix gamma (MG), also called the matrix-variate gamma, distribution is a generalization of the gamma distribution to positive-definite matrices. It is a more general and flexible version of the Wishart distribution (dwishart), and is a conjugate prior of the precision matrix of a multivariate normal distribution (dmvnp) and matrix normal distribution (dmatrixnorm).

The compound distribution resulting from compounding a matrix normal with a matrix gamma prior over the precision matrix is a generalized matrix t-distribution.

The matrix gamma distribution is identical to the Wishart distribution when $\alpha = \nu/2$ and $\beta = 2$.

Value

dmatrixgamma gives the density.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dgamma dmatrixnorm, dmvnp, and dwishart

Examples

```
library(LaplacesDemon)
k <- 10
dmatrixgamma(X=diag(k), alpha=(k+1)/2, beta=2, Sigma=diag(k), log=TRUE)
dwishart(Omega=diag(k), nu=k+1, S=diag(k), log=TRUE)
```

---

dist.Matrix.Normal *Matrix Normal Distribution*

Description

These functions provide the density and random number generation for the matrix normal distribution.

Usage

```
dmatrixnorm(X, M, U, V, log=FALSE)
rmatrixnorm(M, U, V)
```

Arguments

X	This is data or parameters in the form of a matrix with n rows and k columns.
M	This is mean matrix with n rows and k columns.
U	This is a $n \times n$ positive-definite scale matrix.
V	This is a $k \times k$ positive-definite scale matrix.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate Matrix
- Density: $p(\theta) = \frac{\exp(-0.5tr[V^{-1}(X-M)'U^{-1}(X-M)])}{(2\pi)^{nk/2}|V|^{n/2}|U|^{k/2}}$
- Inventors: Unknown
- Notation 1: $\theta \sim \mathcal{MN}_{n\times k}(M, U, V)$
- Notation 2: $p(\theta) = \mathcal{MN}_{n\times k}(\theta|M, U, V)$
- Parameter 1: location $n \times k$ matrix M
- Parameter 2: positive-definite $n \times n$ scale matrix U
- Parameter 3: positive-definite $k \times k$ scale matrix V
- Mean: $E(\theta) = M$
- Variance: Unknown
- Mode: Unknown

The matrix normal distribution is also called the matrix Gaussian, matrix-variate normal, or matrix-variate Gaussian distribution. It is a generalization of the multivariate normal distribution to matrix-valued random variables.

An example of the use of a matrix normal distribution is multivariate regression, in which there is a $j \times k$ matrix of regression effects of j predictors for k dependent variables. For univariate regression, having only one dependent variable, the j regression effects may be multivariate normally distributed. For multivariate regression, this multivariate normal distribution may be extended to a matrix normal distribution to account for relationships of the regression effects across k dependent variables. In this example, the matrix normal distribution is the conjugate prior distribution for these regression effects.

The matrix normal distribution has two covariance matrices, one for the rows and one for the columns. When U is diagonal, the rows are independent. When V is diagonal, the columns are independent.

Value

`dmatrixnorm` gives the density and `rmatrixnorm` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dinvmatrixgamma`, `dmatrixgamma`, and `dmvn`.

Examples

```
library(LaplacesDemon)
N <- 10
K <- 4
U <- as.positive.definite(matrix(rnorm(N*N),N,N))
V <- as.positive.definite(matrix(rnorm(K*K),K,K))
```

```
x <- dmatrixnorm(matrix(0,N,K), matrix(0,N,K), U, V)
X <- rmatrixnorm(matrix(0,N,K), U, V)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

`dist.Multivariate.Cauchy`
Multivariate Cauchy Distribution

Description

These functions provide the density and random number generation for the multivariate Cauchy distribution.

Usage

```
dmvc(x, mu, S, log=FALSE)
rmvc(n=1, mu, S)
```

Arguments

`x`	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in scale matrix $\mathbf{S}$.
`n`	This is the number of random draws.
`mu`	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution It must be of length k, as defined above.
`S`	This is a $k \times k$ positive-definite scale matrix $\mathbf{S}$.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma[(1+k)/2]}{\Gamma(1/2)1^{k/2}\pi^{k/2}|\Sigma|^{1/2}[1+(\theta-\mu)^{\mathrm{T}}\Sigma^{-1}(\theta-\mu)]^{(1+k)/2}}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MC}_k(\mu, \Sigma)$
- Notation 2: $p(\theta) = \mathcal{MC}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ scale matrix Σ
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = undefined$

- Mode: $mode(\theta) = \mu$

The multivariate Cauchy distribution is a multidimensional extension of the one-dimensional or univariate Cauchy distribution. The multivariate Cauchy distribution is equivalent to a multivariate t distribution with 1 degree of freedom. A random vector is considered to be multivariate Cauchy-distributed if every linear combination of its components has a univariate Cauchy distribution.

The Cauchy distribution is known as a pathological distribution because its mean and variance are undefined, and it does not satisfy the central limit theorem.

Value

dmvc gives the density and rmvc generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dcauchy, dinvwishart, dmvcp, dmvt, and dmvtp.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Sigma <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
f <- dmvc(cbind(x,y,z), mu, Sigma)

X <- rmvc(1000, rep(0,2), diag(2))
X <- X[rowSums((X >= quantile(X, probs=0.025)) &
     (X <= quantile(X, probs=0.975)))==2,]
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Cauchy.Cholesky

Multivariate Cauchy Distribution: Cholesky Parameterization

Description

These functions provide the density and random number generation for the multivariate Cauchy distribution, given the Cholesky parameterization.

Usage

```
dmvcc(x, mu, U, log=FALSE)
rmvcc(n=1, mu, U)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in scale matrix $\mathbf{S}$.
n	This is the number of random draws.
mu	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution It must be of length k, as defined above.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor $\mathbf{U}$ of the positive-definite scale matrix $\mathbf{S}$.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma[(1+k)/2]}{\Gamma(1/2)1^{k/2}\pi^{k/2}|\Sigma|^{1/2}[1+(\theta-\mu)^{\mathrm{T}}\Sigma^{-1}(\theta-\mu)]^{(1+k)/2}}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MC}_k(\mu, \Sigma)$
- Notation 2: $p(\theta) = \mathcal{MC}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ scale matrix Σ
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) = \mu$

The multivariate Cauchy distribution is a multidimensional extension of the one-dimensional or univariate Cauchy distribution. The multivariate Cauchy distribution is equivalent to a multivariate t distribution with 1 degree of freedom. A random vector is considered to be multivariate Cauchy-distributed if every linear combination of its components has a univariate Cauchy distribution.

The Cauchy distribution is known as a pathological distribution because its mean and variance are undefined, and it does not satisfy the central limit theorem.

In practice, $\mathbf{U}$ is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with dmvc, dmvcc must additionally matrix-multiply the Cholesky back to the scake matrix, but it does not have to check for or correct the scale matrix to positive-definiteness, which overall is slower. Compared with rmvc, rmvcc is faster because the Cholesky decomposition has already been performed.

Value

dmvcc gives the density and rmvcc generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

chol, dcauchy, dinvwishartc, dmvcpc, dmvtc, and dmvtpc.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Sigma <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
U <- chol(Sigma)
f <- dmvcc(cbind(x,y,z), mu, U)

X <- rmvcc(1000, rep(0,2), diag(2))
X <- X[rowSums((X >= quantile(X, probs=0.025)) &
     (X <= quantile(X, probs=0.975)))==2,]
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Cauchy.Precision

Multivariate Cauchy Distribution: Precision Parameterization

Description

These functions provide the density and random number generation for the multivariate Cauchy distribution. These functions use the precision parameterization.

Usage

```
dmvcp(x, mu, Omega, log=FALSE)
rmvcp(n=1, mu, Omega)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in precision matrix Ω.
n	This is the number of random draws.
mu	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution. It must be of length k, as defined above.
Omega	This is a $k \times k$ positive-definite precision matrix Ω.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma((1+k)/2)}{\Gamma(1/2)1^{k/2}\pi^{k/2}}|\Omega|^{1/2}(1+(\theta-\mu)^T\Omega(\theta-\mu))^{-(1+k)/2}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MC}_k(\mu, \Omega^{-1})$
- Notation 2: $p(\theta) = \mathcal{MC}_k(\theta|\mu, \Omega^{-1})$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = undefined$
- Mode: $mode(\theta) = \mu$

The multivariate Cauchy distribution is a multidimensional extension of the one-dimensional or univariate Cauchy distribution. A random vector is considered to be multivariate Cauchy-distributed if every linear combination of its components has a univariate Cauchy distribution. The multivariate Cauchy distribution is equivalent to a multivariate t distribution with 1 degree of freedom.

The Cauchy distribution is known as a pathological distribution because its mean and variance are undefined, and it does not satisfy the central limit theorem.

It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision parameterization for convenience and familiarity. It is easier to calculate a multivariate Cauchy density with the precision parameterization, because a matrix inversion can be avoided.

This distribution has a mean parameter vector μ of length k, and a $k \times k$ precision matrix Ω, which must be positive-definite.

Value

`dmvcp` gives the density and `rmvcp` generates random deviates.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`dcauchy`, `dmvc`, `dmvt`, `dmvtp`, and `dwishart`.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Omega <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
f <- dmvcp(cbind(x,y,z), mu, Omega)

X <- rmvcp(1000, rep(0,2), diag(2))
X <- X[rowSums((X >= quantile(X, probs=0.025)) &
     (X <= quantile(X, probs=0.975)))==2,]
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Cauchy.Precision.Cholesky

Multivariate Cauchy Distribution: Precision-Cholesky Parameterization

---

Description

These functions provide the density and random number generation for the multivariate Cauchy distribution. These functions use the precision and Cholesky parameterization.

Usage

```
dmvcpc(x, mu, U, log=FALSE)
rmvcpc(n=1, mu, U)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in precision matrix Ω.
n	This is the number of random draws.
mu	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution. It must be of length k, as defined above.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor $\mathbf{U}$ of the positive-definite precision matrix Ω.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma((1+k)/2)}{\Gamma(1/2)1^{k/2}\pi^{k/2}}|\Omega|^{1/2}(1+(\theta-\mu)^T\Omega(\theta-\mu))^{-(1+k)/2}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MC}_k(\mu, \Omega^{-1})$
- Notation 2: $p(\theta) = \mathcal{MC}_k(\theta|\mu, \Omega^{-1})$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) = \mu$

The multivariate Cauchy distribution is a multidimensional extension of the one-dimensional or univariate Cauchy distribution. A random vector is considered to be multivariate Cauchy-distributed if every linear combination of its components has a univariate Cauchy distribution. The multivariate Cauchy distribution is equivalent to a multivariate t distribution with 1 degree of freedom.

The Cauchy distribution is known as a pathological distribution because its mean and variance are undefined, and it does not satisfy the central limit theorem.

It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision parameterization for convenience and familiarity. It is easier to calculate a multivariate Cauchy density with the precision parameterization, because a matrix inversion can be avoided.

This distribution has a mean parameter vector μ of length k, and a $k \times k$ precision matrix Ω, which must be positive-definite. The precision matrix is replaced with the upper-triangular Cholesky factor, as in `chol`.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, Cholesky parameterization is faster than the traditional parameterization. Compared with `dmvcp`, `dmvcpc` must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the precision matrix to positive-definiteness, which overall is slower. Compared with `rmvcp`, `rmvcpc` is faster because the Cholesky decomposition has already been performed.

Value

`dmvcpc` gives the density and `rmvcpc` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`chol`, `dcauchy`, `dmvcc`, `dmvtc`, `dmvtpc`, and `dwishartc`.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Omega <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
U <- chol(Omega)
f <- dmvcpc(cbind(x,y,z), mu, U)

X <- rmvcpc(1000, rep(0,2), diag(2))
X <- X[rowSums((X >= quantile(X, probs=0.025)) &
     (X <= quantile(X, probs=0.975)))==2,]
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Laplace

Multivariate Laplace Distribution

Description

These functions provide the density and random number generation for the multivariate Laplace distribution.

Usage

```
dmvl(x, mu, Sigma, log=FALSE)
rmvl(n, mu, Sigma)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
Sigma	This is the $k \times k$ covariance matrix Σ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{2}{(2\pi)^{k/2}|\Sigma|^{1/2}} \frac{(\pi/(2\sqrt{2(\theta-\mu)^T\Sigma^{-1}(\theta-\mu)}))^{1/2} \exp(-\sqrt{2(\theta-\mu)^T\Sigma^{-1}(\theta-\mu)})}{\sqrt{((\theta-\mu)^T\Sigma^{-1}(\theta-\mu)/2)}^{k/2-1}}$$

- Inventor: Fang et al. (1990)
- Notation 1: $\theta \sim \mathcal{MVL}(\mu, \Sigma)$
- Notation 2: $\theta \sim \mathcal{L}_k(\mu, \Sigma)$
- Notation 3: $p(\theta) = \mathcal{MVL}(\theta|\mu, \Sigma)$
- Notation 4: $p(\theta) = \mathcal{L}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ covariance matrix Σ
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \Sigma$
- Mode: $mode(\theta) = \mu$

The multivariate Laplace distribution is a multidimensional extension of the one-dimensional or univariate symmetric Laplace distribution. There are multiple forms of the multivariate Laplace distribution.

The bivariate case was introduced by Ulrich and Chen (1987), and the first form in larger dimensions may have been Fang et al. (1990), which requires a Bessel function. Alternatively, multivariate Laplace was soon introduced as a special case of a multivariate Linnik distribution (Anderson, 1992), and later as a special case of the multivariate power exponential distribution (Fernandez et al., 1995; Ernst, 1998). Bayesian considerations appear in Haro-Lopez and Smith (1999). Wainwright and Simoncelli (2000) presented multivariate Laplace as a Gaussian scale mixture. Kotz et al. (2001) present the distribution formally. Here, the density is calculated with the asymptotic formula for the Bessel function as presented in Wang et al. (2008).

The multivariate Laplace distribution is an attractive alternative to the multivariate normal distribution due to its wider tails, and remains a two-parameter distribution (though alternative three-parameter forms have been introduced as well), unlike the three-parameter multivariate t distribution, which is often used as a robust alternative to the multivariate normal distribution.

Value

`dmvl` gives the density, and `rmvl` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Anderson, D.N. (1992). "A Multivariate Linnik Distribution". *Statistical Probability Letters*, 14, p. 333–336.

Eltoft, T., Kim, T., and Lee, T. (2006). "On the Multivariate Laplace Distribution". *IEEE Signal Processing Letters*, 13(5), p. 300–303.

Ernst, M. D. (1998). "A Multivariate Generalized Laplace Distribution". *Computational Statistics*, 13, p. 227–232.

Fang, K.T., Kotz, S., and Ng, K.W. (1990). "Symmetric Multivariate and Related Distributions". Monographs on Statistics and Probability, 36, Chapman-Hall, London.

Fernandez, C., Osiewalski, J. and Steel, M.F.J. (1995). "Modeling and Inference with v-spherical Distributions". *Journal of the American Statistical Association*, 90, p. 1331–1340.

Gomez, E., Gomez-Villegas, M.A., and Marin, J.M. (1998). "A Multivariate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics-Theory and Methods*, 27(3), p. 589–600.

Haro-Lopez, R.A. and Smith, A.F.M. (1999). "On Robust Bayesian Analysis for Location and Scale Parameters". *Journal of Multivariate Analysis*, 70, p. 30–56.

Kotz., S., Kozubowski, T.J., and Podgorski, K. (2001). "The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance". Birkhauser: Boston, MA.

Ulrich, G. and Chen, C.C. (1987). "A Bivariate Double Exponential Distribution and its Generalization". *ASA Proceedings on Statistical Computing*, p. 127–129.

Wang, D., Zhang, C., and Zhao, X. (2008). "Multivariate Laplace Filter: A Heavy-Tailed Model for Target Tracking". *Proceedings of the 19th International Conference on Pattern Recognition*: FL.

Wainwright, M.J. and Simoncelli, E.P. (2000). "Scale Mixtures of Gaussians and the Statistics of Natural Images". *Advances in Neural Information Processing Systems*, 12, p. 855–861.

See Also

daml, dlaplace, dmvn, dmvnp, dmvpe, dmvt, dnorm, dnormp, and dnormv.

Examples

```
library(LaplacesDemon)
x <- dmvl(c(1,2,3), c(0,1,2), diag(3))
X <- rmvl(1000, c(0,1,2), diag(3))
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Laplace.Cholesky

Multivariate Laplace Distribution: Cholesky Parameterization

Description

These functions provide the density and random number generation for the multivariate Laplace distribution, given the Cholesky parameterization.

Usage

```
dmvlc(x, mu, U, log=FALSE)
rmvlc(n, mu, U)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor **U** of covariance matrix Σ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{2}{(2\pi)^{k/2}|\Sigma|^{1/2}} \frac{(\pi/(2\sqrt{2(\theta-\mu)^T\Sigma^{-1}(\theta-\mu)}))^{1/2} \exp(-\sqrt{2(\theta-\mu)^T\Sigma^{-1}(\theta-\mu)})}{\sqrt{((\theta-\mu)^T\Sigma^{-1}(\theta-\mu)/2)}^{k/2-1}}$$

- Inventor: Fang et al. (1990)
- Notation 1: $\theta \sim \mathcal{MVL}(\mu, \Sigma)$
- Notation 2: $\theta \sim \mathcal{L}_k(\mu, \Sigma)$
- Notation 3: $p(\theta) = \mathcal{MVL}(\theta|\mu, \Sigma)$
- Notation 4: $p(\theta) = \mathcal{L}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ covariance matrix Σ
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \Sigma$
- Mode: $mode(\theta) = \mu$

The multivariate Laplace distribution is a multidimensional extension of the one-dimensional or univariate symmetric Laplace distribution. There are multiple forms of the multivariate Laplace distribution.

The bivariate case was introduced by Ulrich and Chen (1987), and the first form in larger dimensions may have been Fang et al. (1990), which requires a Bessel function. Alternatively, multivariate Laplace was soon introduced as a special case of a multivariate Linnik distribution (Anderson, 1992), and later as a special case of the multivariate power exponential distribution (Fernandez et al., 1995; Ernst, 1998). Bayesian considerations appear in Haro-Lopez and Smith (1999). Wainwright and Simoncelli (2000) presented multivariate Laplace as a Gaussian scale mixture. Kotz et al. (2001) present the distribution formally. Here, the density is calculated with the asymptotic formula for the Bessel function as presented in Wang et al. (2008).

The multivariate Laplace distribution is an attractive alternative to the multivariate normal distribution due to its wider tails, and remains a two-parameter distribution (though alternative three-parameter forms have been introduced as well), unlike the three-parameter multivariate t distribution, which is often used as a robust alternative to the multivariate normal distribution.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with `dmvl`, `dmvlc` must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the covariance matrix to positive-definiteness, which overall is slower. Compared with `rmvl`, `rmvlc` is faster because the Cholesky decomposition has already been performed.

Value

`dmvlc` gives the density, and `rmvlc` generates random deviates.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Anderson, D.N. (1992). "A Multivariate Linnik Distribution". *Statistical Probability Letters*, 14, p. 333–336.

Eltoft, T., Kim, T., and Lee, T. (2006). "On the Multivariate Laplace Distribution". *IEEE Signal Processing Letters*, 13(5), p. 300–303.

Ernst, M. D. (1998). "A Multivariate Generalized Laplace Distribution". *Computational Statistics*, 13, p. 227–232.

Fang, K.T., Kotz, S., and Ng, K.W. (1990). "Symmetric Multivariate and Related Distributions". Monographs on Statistics and Probability, 36, Chapman-Hall, London.

Fernandez, C., Osiewalski, J. and Steel, M.F.J. (1995). "Modeling and Inference with v-spherical Distributions". *Journal of the American Statistical Association*, 90, p. 1331–1340.

Gomez, E., Gomez-Villegas, M.A., and Marin, J.M. (1998). "A Multivariate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics-Theory and Methods*, 27(3), p. 589–600.

Haro-Lopez, R.A. and Smith, A.F.M. (1999). "On Robust Bayesian Analysis for Location and Scale Parameters". *Journal of Multivariate Analysis*, 70, p. 30–56.

Kotz., S., Kozubowski, T.J., and Podgorski, K. (2001). "The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance". Birkhauser: Boston, MA.

Ulrich, G. and Chen, C.C. (1987). "A Bivariate Double Exponential Distribution and its Generalization". *ASA Proceedings on Statistical Computing*, p. 127–129.

Wang, D., Zhang, C., and Zhao, X. (2008). "Multivariate Laplace Filter: A Heavy-Tailed Model for Target Tracking". *Proceedings of the 19th International Conference on Pattern Recognition*: FL.

Wainwright, M.J. and Simoncelli, E.P. (2000). "Scale Mixtures of Gaussians and the Statistics of Natural Images". *Advances in Neural Information Processing Systems*, 12, p. 855–861.

See Also

`chol`, `daml`, `dlaplace`, `dmvnc`, `dmvnpc`, `dmvpec`, `dmvtc`, `dnorm`, `dnormp`, and `dnormv`.

Examples

```
library(LaplacesDemon)
Sigma <- diag(3)
U <- chol(Sigma)
x <- dmvlc(c(1,2,3), c(0,1,2), U)
X <- rmvlc(1000, c(0,1,2), U)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Normal

Multivariate Normal Distribution

Description

These functions provide the density and random number generation for the multivariate normal distribution.

Usage

```
dmvn(x, mu, Sigma, log=FALSE)
rmvn(n=1, mu, Sigma)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
Sigma	This is the $k \times k$ covariance matrix Σ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\theta-\mu)'\Sigma^{-1}(\theta-\mu))$
- Inventors: Robert Adrain (1808), Pierre-Simon Laplace (1812), and Francis Galton (1885)
- Notation 1: $\theta \sim \mathcal{MVN}(\mu, \Sigma)$
- Notation 2: $\theta \sim \mathcal{N}_k(\mu, \Sigma)$
- Notation 3: $p(\theta) = \mathcal{MVN}(\theta|\mu, \Sigma)$
- Notation 4: $p(\theta) = \mathcal{N}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ covariance matrix Σ
- Mean: $E(\theta) = \mu$

- Variance: $var(\theta) = \Sigma$
- Mode: $mode(\theta) = \mu$

The multivariate normal distribution, or multivariate Gaussian distribution, is a multidimensional extension of the one-dimensional or univariate normal (or Gaussian) distribution. A random vector is considered to be multivariate normally distributed if every linear combination of its components has a univariate normal distribution. This distribution has a mean parameter vector μ of length k and a $k \times k$ covariance matrix Σ, which must be positive-definite.

The conjugate prior of the mean vector is another multivariate normal distribution. The conjugate prior of the covariance matrix is the inverse Wishart distribution (see dinvwishart).

When applicable, the alternative Cholesky parameterization should be preferred. For more information, see dmvnc.

For models where the dependent variable, Y, is specified to be distributed multivariate normal given the model, the Mardia test (see plot.demonoid.ppc, plot.laplace.ppc, or plot.pmc.ppc) may be used to test the residuals.

Value

dmvn gives the density and rmvn generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dinvwishart, dmatrixnorm, dmvnc, dmvnp, dnorm, dnormp, dnormv, plot.demonoid.ppc, plot.laplace.ppc, and plot.pmc.ppc.

Examples

```
library(LaplacesDemon)
x <- dmvn(c(1,2,3), c(0,1,2), diag(3))
X <- rmvn(1000, c(0,1,2), diag(3))
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Normal.Cholesky

Multivariate Normal Distribution: Cholesky Parameterization

Description

These functions provide the density and random number generation for the multivariate normal distribution, given the Cholesky parameterization.

Usage

```
dmvnc(x, mu, U, log=FALSE)
rmvnc(n=1, mu, U)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor **U** of covariance matrix Σ.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \frac{1}{(2\pi)^{k/2}|\Sigma|^{1/2}} \exp(-\frac{1}{2}(\theta-\mu)'\Sigma^{-1}(\theta-\mu))$
- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MVN}(\mu, \Sigma)$
- Notation 2: $\theta \sim \mathcal{N}_k(\mu, \Sigma)$
- Notation 3: $p(\theta) = \mathcal{MVN}(\theta|\mu, \Sigma)$
- Notation 4: $p(\theta) = \mathcal{N}_k(\theta|\mu, \Sigma)$
- Parameter 1: location vector μ
- Parameter 2: $k \times k$ positive-definite matrix Σ
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \Sigma$
- Mode: $mode(\theta) = \mu$

The multivariate normal distribution, or multivariate Gaussian distribution, is a multidimensional extension of the one-dimensional or univariate normal (or Gaussian) distribution. A random vector is considered to be multivariate normally distributed if every linear combination of its components has a univariate normal distribution. This distribution has a mean parameter vector μ of length k and an upper-triangular $k \times k$ matrix that is Cholesky factor **U**, as per the chol function for Cholesky decomposition.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with dmvn, dmvnc must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the covariance matrix to positive-definiteness, which overall is slower. Compared with rmvn, rmvnc is faster because the Cholesky decomposition has already been performed.

For models where the dependent variable, Y, is specified to be distributed multivariate normal given the model, the Mardia test (see plot.demonoid.ppc, plot.laplace.ppc, or plot.pmc.ppc) may be used to test the residuals.

Value

dmvnc gives the density and rmvnc generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

chol, dinvwishartc, dmvn, dmvnp, dmvnpc, dnorm, dnormp, dnormv, plot.demonoid.ppc, plot.laplace.ppc, and plot.pmc.ppc.

Examples

```
library(LaplacesDemon)
Sigma <- diag(3)
U <- chol(Sigma)
x <- dmvnc(c(1,2,3), c(0,1,2), U)
X <- rmvnc(1000, c(0,1,2), U)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Normal.Precision

Multivariate Normal Distribution: Precision Parameterization

Description

These functions provide the density and random number generation for the multivariate normal distribution, given the precision parameterization.

Usage

```
dmvnp(x, mu, Omega, log=FALSE)
rmvnp(n=1, mu, Omega)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
Omega	This is the $k \times k$ precision matrix Ω.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2\pi)^{-p/2}|\Omega|^{1/2}\exp(-\frac{1}{2}(\theta-\mu)^T\Omega(\theta-\mu))$
- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MVN}(\mu, \Omega^{-1})$
- Notation 2: $\theta \sim \mathcal{N}_k(\mu, \Omega^{-1})$
- Notation 3: $p(\theta) = \mathcal{MVN}(\theta|\mu, \Omega^{-1})$
- Notation 4: $p(\theta) = \mathcal{N}_k(\theta|\mu, \Omega^{-1})$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \Omega^{-1}$
- Mode: $mode(\theta) = \mu$

The multivariate normal distribution, or multivariate Gaussian distribution, is a multidimensional extension of the one-dimensional or univariate normal (or Gaussian) distribution. It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision parameterization for convenience and familiarity. It is easier to calculate a multivariate normal density with the precision parameterization, because a matrix inversion can be avoided.

A random vector is considered to be multivariate normally distributed if every linear combination of its components has a univariate normal distribution. This distribution has a mean parameter vector μ of length k and a $k \times k$ precision matrix Ω, which must be positive-definite.

The conjugate prior of the mean vector is another multivariate normal distribution. The conjugate prior of the precision matrix is the Wishart distribution (see `dwishart`).

When applicable, the alternative Cholesky parameterization should be preferred. For more information, see `dmvnpc`.

For models where the dependent variable, Y, is specified to be distributed multivariate normal given the model, the Mardia test (see `plot.demonoid.ppc`, `plot.laplace.ppc`, or `plot.pmc.ppc`) may be used to test the residuals.

Value

`dmvnp` gives the density and `rmvnp` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dmvn`, `dmvnc`, `dmvnpc`, `dnorm`, `dnormp`, `dnormv`, `dwishart`, `plot.demonoid.ppc`, `plot.laplace.ppc`, and `plot.pmc.ppc`.

Examples

```
library(LaplacesDemon)
x <- dmvnp(c(1,2,3), c(0,1,2), diag(3))
X <- rmvnp(1000, c(0,1,2), diag(3))
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Normal.Precision.Cholesky

Multivariate Normal Distribution: Precision-Cholesky Parameterization

---

Description

These functions provide the density and random number generation for the multivariate normal distribution, given the precision-Cholesky parameterization.

Usage

```
dmvnpc(x, mu, U, log=FALSE)
rmvnpc(n=1, mu, U)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
U	This is the $k \times k$ upper-triangular of the precision matrix that is Cholesky factor $\mathbf{U}$ of precision matrix Ω.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2\pi)^{-p/2}|\Omega|^{1/2}\exp(-\frac{1}{2}(\theta-\mu)^T\Omega(\theta-\mu))$
- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathcal{MVN}(\mu, \Omega^{-1})$
- Notation 2: $\theta \sim \mathcal{N}_k(\mu, \Omega^{-1})$
- Notation 3: $p(\theta) = \mathcal{MVN}(\theta|\mu, \Omega^{-1})$
- Notation 4: $p(\theta) = \mathcal{N}_k(\theta|\mu, \Omega^{-1})$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Mean: $E(\theta) = \mu$

- Variance: $var(\theta) = \Omega^{-1}$
- Mode: $mode(\theta) = \mu$

The multivariate normal distribution, or multivariate Gaussian distribution, is a multidimensional extension of the one-dimensional or univariate normal (or Gaussian) distribution. It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision-Cholesky parameterization for convenience and familiarity. It is easier to calculate a multivariate normal density with the precision parameterization, because a matrix inversion can be avoided. The precision matrix is replaced with an upper-triangular $k \times k$ matrix that is Cholesky factor **U**, as per the chol function for Cholesky decomposition.

A random vector is considered to be multivariate normally distributed if every linear combination of its components has a univariate normal distribution. This distribution has a mean parameter vector μ of length k and a $k \times k$ precision matrix Ω, which must be positive-definite.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, Cholesky parameterization is faster than the traditional parameterization. Compared with dmvnp, dmvnpc must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the precision matrix to positive-definiteness, which overall is slower. Compared with rmvnp, rmvnpc is faster because the Cholesky decomposition has already been performed.

For models where the dependent variable, Y, is specified to be distributed multivariate normal given the model, the Mardia test (see plot.demonoid.ppc, plot.laplace.ppc, or plot.pmc.ppc) may be used to test the residuals.

Value

dmvnpc gives the density and rmvnpc generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

chol, dmvn, dmvnc, dmvnp, dnorm, dnormp, dnormv, dwishartc, plot.demonoid.ppc, plot.laplace.ppc, and plot.pmc.ppc.

Examples

```
library(LaplacesDemon)
Omega <- diag(3)
U <- chol(Omega)
x <- dmvnpc(c(1,2,3), c(0,1,2), U)
X <- rmvnpc(1000, c(0,1,2), U)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

`dist.Multivariate.Polya`

Multivariate Polya Distribution

Description

These functions provide the density and random number generation for the multivariate Polya distribution.

Usage

```
dmvpolya(x, alpha, log=FALSE)
rmvpolya(n, alpha)
```

Arguments

`x`	This is data or parameters in the form of a vector of length k.
`n`	This is the number of random draws to take from the distribution.
`alpha`	This is shape vector α with length k.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Discrete Multivariate
- Density:
$$p(\theta) = \frac{N!}{\prod_k N_k!} \frac{(\sum_k \alpha_k - 1)!}{(\sum_k \theta_k + \sum_k \alpha_k - 1)!} \frac{\prod(\theta + \alpha - 1)!}{(\alpha - 1)!}$$
- Inventor: George Polya (1887-1985)
- Notation 1: $\theta \sim \mathcal{MPO}(\alpha)$
- Notation 3: $p(\theta) = \mathcal{MPO}(\theta|\alpha)$
- Parameter 1: shape parameter vector α
- Mean: $E(\theta) =$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) =$

The multivariate Polya distribution is named after George Polya (1887-1985). It is also called the Dirichlet compound multinomial distribution or the Dirichlet-multinomial distribution. The multivariate Polya distribution is a compound probability distribution, where a probability vector p is drawn from a Dirichlet distribution with parameter vector α, and a set of N discrete samples is drawn from the categorical distribution with probability vector p and having K discrete categories. The compounding corresponds to a Polya urn scheme. In document classification, for example, the distribution is used to represent probabilities over word counts for different document types. The multivariate Polya distribution is a multivariate extension of the univariate Beta-binomial distribution.

Value

dmvpolya gives the density and rmvpolya generates random deviates.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

See Also

dcat, ddirichlet, and dmultinom.

Examples

```
library(LaplacesDemon)
dmvpolya(x=1:3, alpha=1:3, log=TRUE)
x <- rmvpolya(1000, c(0.1,0.3,0.6))
```

---

dist.Multivariate.Power.Exponential
Multivariate Power Exponential Distribution

Description

These functions provide the density and random number generation for the multivariate power exponential distribution.

Usage

```
dmvpe(x=c(0,0), mu=c(0,0), Sigma=diag(2), kappa=1, log=FALSE)
rmvpe(n, mu=c(0,0), Sigma=diag(2), kappa=1)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
Sigma	This is the $k \times k$ covariance matrix Σ.
kappa	This is the kurtosis parameter, κ, and must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{k\Gamma(k/2)}{\pi^{k/2}\sqrt{|\Sigma|}\Gamma(1+k/(2\kappa))2^{1+k/(2\kappa)}} \exp(-\frac{1}{2}(\theta-\mu)^T\Sigma(\theta-\mu))^\kappa$$

- Inventor: Gomez, Gomez-Villegas, and Marin (1998)
- Notation 1: $\theta \sim \mathcal{MPE}(\mu, \Sigma, \kappa)$
- Notation 2: $\theta \sim \mathcal{PE}_k(\mu, \Sigma, \kappa)$
- Notation 3: $p(\theta) = \mathcal{MPE}(\theta|\mu, \Sigma, \kappa)$
- Notation 4: $p(\theta) = \mathcal{PE}_k(\theta|\mu, \Sigma, \kappa)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ covariance matrix Σ
- Parameter 3: kurtosis parameter κ
- Mean: $E(\theta) =$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) =$

The multivariate power exponential distribution, or multivariate exponential power distribution, is a multidimensional extension of the one-dimensional or univariate power exponential distribution. Gomez-Villegas (1998) and Sanchez-Manzano et al. (2002) proposed multivariate and matrix generalizations of the PE family of distributions and studied their properties in relation to multivariate Elliptically Contoured (EC) distributions.

The multivariate power exponential distribution includes the multivariate normal distribution ($\kappa = 1$) and multivariate Laplace distribution ($\kappa = 0.5$) as special cases, depending on the kurtosis or κ parameter. A multivariate uniform occurs as $\kappa \to \infty$.

If the goal is to use a multivariate Laplace distribution, the `dmvl` function will perform faster and more accurately.

The `rmvpe` function is a modified form of the rmvpowerexp function in the MNM package.

Value

`dmvpe` gives the density and `rmvpe` generates random deviates.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Gomez, E., Gomez-Villegas, M.A., and Marin, J.M. (1998). "A Multivariate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics-Theory and Methods*, 27(3), p. 589–600.

Sanchez-Manzano, E.G., Gomez-Villegas, M.A., and Marn-Diazaraque, J.M. (2002). "A Matrix Variate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics, Part A - Theory and Methods* [Split from: J(CommStat)], 31(12), p. 2167–2182.

See Also

dlaplace, dmvl, dmvn, dmvnp, dnorm, dnormp, dnormv, and dpe.

Examples

```
library(LaplacesDemon)
n <- 100
k <- 3
x <- matrix(runif(n*k),n,k)
mu <- matrix(runif(n*k),n,k)
Sigma <- diag(k)
dmvpe(x, mu, Sigma, kappa=1)
X <- rmvpe(n, mu, Sigma, kappa=1)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.Power.Exponential.Cholesky

Multivariate Power Exponential Distribution: Cholesky Parameterization

---

Description

These functions provide the density and random number generation for the multivariate power exponential distribution, given the Cholesky parameterization.

Usage

```
dmvpec(x=c(0,0), mu=c(0,0), U, kappa=1, log=FALSE)
rmvpec(n, mu=c(0,0), U, kappa=1)
```

Arguments

x	This is data or parameters in the form of a vector of length k or a matrix with k columns.
n	This is the number of random draws.
mu	This is mean vector μ with length k or matrix with k columns.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor **U** of covariance matrix Σ.
kappa	This is the kurtosis parameter, κ, and must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{k\Gamma(k/2)}{\pi^{k/2}\sqrt{|\Sigma|}\Gamma(1+k/(2\kappa))2^{1+k/(2\kappa)}} \exp(-\frac{1}{2}(\theta-\mu)^T\Sigma(\theta-\mu))^\kappa$$

- Inventor: Gomez, Gomez-Villegas, and Marin (1998)
- Notation 1: $\theta \sim \mathcal{MPE}(\mu, \Sigma, \kappa)$
- Notation 2: $\theta \sim \mathcal{PE}_k(\mu, \Sigma, \kappa)$
- Notation 3: $p(\theta) = \mathcal{MPE}(\theta|\mu, \Sigma, \kappa)$
- Notation 4: $p(\theta) = \mathcal{PE}_k(\theta|\mu, \Sigma, \kappa)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ covariance matrix Σ
- Parameter 3: kurtosis parameter κ
- Mean: $E(\theta) =$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) =$

The multivariate power exponential distribution, or multivariate exponential power distribution, is a multidimensional extension of the one-dimensional or univariate power exponential distribution. Gomez-Villegas (1998) and Sanchez-Manzano et al. (2002) proposed multivariate and matrix generalizations of the PE family of distributions and studied their properties in relation to multivariate Elliptically Contoured (EC) distributions.

The multivariate power exponential distribution includes the multivariate normal distribution ($\kappa = 1$) and multivariate Laplace distribution ($\kappa = 0.5$) as special cases, depending on the kurtosis or κ parameter. A multivariate uniform occurs as $\kappa \to \infty$.

If the goal is to use a multivariate Laplace distribution, the `dmvlc` function will perform faster and more accurately.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with `dmvpe`, `dmvpec` must additionally matrix-multiply the Cholesky back to the covariance matrix, but it does not have to check for or correct the covariance matrix to positive-definiteness, which overall is slower. Compared with `rmvpe`, `rmvpec` is faster because the Cholesky decomposition has already been performed.

The `rmvpec` function is a modified form of the rmvpowerexp function in the MNM package.

Value

`dmvpec` gives the density and `rmvpec` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Gomez, E., Gomez-Villegas, M.A., and Marin, J.M. (1998). "A Multivariate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics-Theory and Methods*, 27(3), p. 589–600.

Sanchez-Manzano, E.G., Gomez-Villegas, M.A., and Marn-Diazaraque, J.M. (2002). "A Matrix Variate Generalization of the Power Exponential Family of Distributions". *Communications in Statistics, Part A - Theory and Methods* [Split from: J(CommStat)], 31(12), p. 2167–2182.

See Also

chol, dlaplace, dmvlc, dmvnc, dmvnpc, dnorm, dnormp, dnormv, and dpe.

Examples

```
library(LaplacesDemon)
n <- 100
k <- 3
x <- matrix(runif(n*k),n,k)
mu <- matrix(runif(n*k),n,k)
Sigma <- diag(k)
U <- chol(Sigma)
dmvpec(x, mu, U, kappa=1)
X <- rmvpec(n, mu, U, kappa=1)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.t *Multivariate t Distribution*

Description

These functions provide the density and random number generation for the multivariate t distribution, otherwise called the multivariate Student distribution.

Usage

```
dmvt(x, mu, S, df=Inf, log=FALSE)
rmvt(n=1, mu, S, df=Inf)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in scale matrix $\mathbf{S}$.
n	This is the number of random draws.
mu	This is a numeric vector or matrix representing the location parameter,μ (the mean vector), of the multivariate distribution (equal to the expected value when df > 1, otherwise represented as $\nu > 1$). When a vector, it must be of length k, or must have k columns as a matrix, as defined above.

S	This is a $k \times k$ positive-definite scale matrix **S**, such that S*df/(df-2) is the variance-covariance matrix when df > 2. A vector of length 1 is also allowed (in this case, $k = 1$ is set).
df	This is the degrees of freedom, and is often represented with ν.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma[(\nu + k)/2]}{\Gamma(\nu/2)\nu^{k/2}\pi^{k/2}|\Sigma|^{1/2}[1 + (1/\nu)(\theta - \mu)^{\mathrm{T}}\Sigma^{-1}(\theta - \mu)]^{(\nu+k)/2}}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathrm{t}_k(\mu, \Sigma, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}_k(\theta|\mu, \Sigma, \nu)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ scale matrix Σ
- Parameter 3: degrees of freedom $\nu > 0$ (df in the functions)
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{\nu}{\nu-2}\Sigma$, for $\nu > 2$
- Mode: $mode(\theta) = \mu$

The multivariate t distribution, also called the multivariate Student or multivariate Student t distribution, is a multidimensional extension of the one-dimensional or univariate Student t distribution. A random vector is considered to be multivariate t-distributed if every linear combination of its components has a univariate Student t-distribution. This distribution has a mean parameter vector μ of length k, and a $k \times k$ scale matrix **S**, which must be positive-definite. When degrees of freedom $\nu = 1$, this is the multivariate Cauchy distribution.

Value

dmvt gives the density and rmvt generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dinvwishart, dmvc, dmvcp, dmvtp, dst, dstp, and dt.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
S <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
df <- 4
f <- dmvt(cbind(x,y,z), mu, S, df)
X <- rmvt(1000, c(0,1,2), S, 5)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Multivariate.t.Cholesky

Multivariate t Distribution: Cholesky Parameterization

Description

These functions provide the density and random number generation for the multivariate t distribution, otherwise called the multivariate Student distribution, given the Cholesky parameterization.

Usage

```
dmvtc(x, mu, U, df=Inf, log=FALSE)
rmvtc(n=1, mu, U, df=Inf)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in scale matrix $\mathbf{S}$.
n	This is the number of random draws.
mu	This is a numeric vector or matrix representing the location parameter,μ (the mean vector), of the multivariate distribution (equal to the expected value when df > 1, otherwise represented as $\nu > 1$). When a vector, it must be of length k, or must have k columns as a matrix, as defined above.
U	This is the $k \times k$ upper-triangular matrix that is Cholesky factor $\mathbf{U}$ of scale matrix $\mathbf{S}$, such that S*df/(df-2) is the variance-covariance matrix when df > 2.
df	This is the degrees of freedom, and is often represented with ν.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma[(\nu+k)/2]}{\Gamma(\nu/2)\nu^{k/2}\pi^{k/2}|\Sigma|^{1/2}[1+(1/\nu)(\theta-\mu)^{\mathrm{T}}\Sigma^{-1}(\theta-\mu)]^{(\nu+k)/2}}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathrm{t}_k(\mu, \Sigma, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}_k(\theta|\mu, \Sigma, \nu)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ scale matrix Σ
- Parameter 3: degrees of freedom $\nu > 0$ (df in the functions)
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{\nu}{\nu-2}\Sigma$, for $\nu > 2$
- Mode: $mode(\theta) = \mu$

The multivariate t distribution, also called the multivariate Student or multivariate Student t distribution, is a multidimensional extension of the one-dimensional or univariate Student t distribution. A random vector is considered to be multivariate t-distributed if every linear combination of its components has a univariate Student t-distribution. This distribution has a mean parameter vector μ of length k, and an upper-triangular $k \times k$ matrix that is Cholesky factor **U**, as per the chol function for Cholesky decomposition. When degrees of freedom $\nu = 1$, this is the multivariate Cauchy distribution.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with dmvt, dmvtc must additionally matrix-multiply the Cholesky back to the scale matrix, but it does not have to check for or correct the scale matrix to positive-definiteness, which overall is slower. The same is true when comparing rmvt and rmvtc.

Value

dmvtc gives the density and rmvtc generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

chol, dinvwishartc, dmvc, dmvcp, dmvtp, dst, dstp, and dt.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
S <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
U <- chol(S)
df <- 4
f <- dmvtc(cbind(x,y,z), mu, U, df)
X <- rmvtc(1000, c(0,1,2), U, 5)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

`dist.Multivariate.t.Precision`
Multivariate t Distribution: Precision Parameterization

Description

These functions provide the density and random number generation for the multivariate t distribution, otherwise called the multivariate Student distribution. These functions use the precision parameterization.

Usage

```
dmvtp(x, mu, Omega, nu=Inf, log=FALSE)
rmvtp(n=1, mu, Omega, nu=Inf)
```

Arguments

`x`	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in precision matrix Ω.
`n`	This is the number of random draws.
`mu`	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution (equal to the expected value when `df > 1`, otherwise represented as $\nu > 1$). It must be of length k, as defined above.
`Omega`	This is a $k \times k$ positive-definite precision matrix Ω.
`nu`	This is the degrees of freedom ν, which must be positive.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma((\nu+k)/2)}{\Gamma(\nu/2)\nu^{k/2}\pi^{k/2}}|\Omega|^{1/2}(1+\frac{1}{\nu}(\theta-\mu)^T\Omega(\theta-\mu))^{-(\nu+k)/2}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathrm{t}_k(\mu, \Omega^{-1}, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}_k(\theta|\mu, \Omega^{-1}, \nu)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Parameter 3: degrees of freedom $\nu > 0$
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{\nu}{\nu-2}\Omega^{-1}$, for $\nu > 2$

- Mode: $mode(\theta) = \mu$

The multivariate t distribution, also called the multivariate Student or multivariate Student t distribution, is a multidimensional extension of the one-dimensional or univariate Student t distribution. A random vector is considered to be multivariate t-distributed if every linear combination of its components has a univariate Student t-distribution.

It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision parameterization for convenience and familiarity. It is easier to calculate a multivariate t density with the precision parameterization, because a matrix inversion can be avoided.

This distribution has a mean parameter vector μ of length k, and a $k \times k$ precision matrix Ω, which must be positive-definite. When degrees of freedom $\nu = 1$, this is the multivariate Cauchy distribution.

Value

`dmvtp` gives the density and `rmvtp` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dwishart`, `dmvc`, `dmvcp`, `dmvt`, `dst`, `dstp`, and `dt`.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Omega <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
nu <- 4
f <- dmvtp(cbind(x,y,z), mu, Omega, nu)
X <- rmvtp(1000, c(0,1,2), diag(3), 5)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

`dist.Multivariate.t.Precision.Cholesky`
Multivariate t Distribution: Precision-Cholesky Parameterization

---

Description

These functions provide the density and random number generation for the multivariate t distribution, otherwise called the multivariate Student distribution. These functions use the precision and Cholesky parameterization.

Usage

```
dmvtpc(x, mu, U, nu=Inf, log=FALSE)
rmvtpc(n=1, mu, U, nu=Inf)
```

Arguments

x	This is either a vector of length k or a matrix with a number of columns, k, equal to the number of columns in precision matrix Ω.
n	This is the number of random draws.
mu	This is a numeric vector representing the location parameter, μ (the mean vector), of the multivariate distribution (equal to the expected value when df > 1, otherwise represented as $\nu > 1$). It must be of length k, as defined above.
U	This is a $k \times k$ upper-triangular of the precision matrix that is Cholesky fator **U** of precision matrix Ω.
nu	This is the degrees of freedom ν, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density:

$$p(\theta) = \frac{\Gamma((\nu+k)/2)}{\Gamma(\nu/2)\nu^{k/2}\pi^{k/2}}|\Omega|^{1/2}(1+\frac{1}{\nu}(\theta-\mu)^T\Omega(\theta-\mu))^{-(\nu+k)/2}$$

- Inventor: Unknown (to me, anyway)
- Notation 1: $\theta \sim \mathrm{t}_k(\mu, \Omega^{-1}, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}_k(\theta|\mu, \Omega^{-1}, \nu)$
- Parameter 1: location vector μ
- Parameter 2: positive-definite $k \times k$ precision matrix Ω
- Parameter 3: degrees of freedom $\nu > 0$
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{\nu}{\nu-2}\Omega^{-1}$, for $\nu > 2$
- Mode: $mode(\theta) = \mu$

The multivariate t distribution, also called the multivariate Student or multivariate Student t distribution, is a multidimensional extension of the one-dimensional or univariate Student t distribution. A random vector is considered to be multivariate t-distributed if every linear combination of its components has a univariate Student t-distribution.

It is usually parameterized with mean and a covariance matrix, or in Bayesian inference, with mean and a precision matrix, where the precision matrix is the matrix inverse of the covariance matrix. These functions provide the precision parameterization for convenience and familiarity. It is easier to calculate a multivariate t density with the precision parameterization, because a matrix inversion can be avoided. The precision matrix is replaced with an upper-triangular $k \times k$ matrix that is Cholesky factor **U**, as per the chol function for Cholesky decomposition.

This distribution has a mean parameter vector μ of length k, and a $k \times k$ precision matrix Ω, which must be positive-definite. When degrees of freedom $\nu = 1$, this is the multivariate Cauchy distribution.

In practice, **U** is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with `dmvtp`, `dmvtpc` must additionally matrix-multiply the Cholesky back to the precision matrix, but it does not have to check for or correct the precision matrix to positive-definiteness, which overall is slower. Compared with `rmvtp`, `rmvtpc` is faster because the Cholesky decomposition has already been performed.

Value

`dmvtpc` gives the density and `rmvtpc` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`chol`, `dwishartc`, `dmvc`, `dmvcp`, `dmvtc`, `dst`, `dstp`, and `dt`.

Examples

```
library(LaplacesDemon)
x <- seq(-2,4,length=21)
y <- 2*x+10
z <- x+cos(y)
mu <- c(1,12,2)
Omega <- matrix(c(1,2,0,2,5,0.5,0,0.5,3), 3, 3)
U <- chol(Omega)
nu <- 4
f <- dmvtpc(cbind(x,y,z), mu, U, nu)
X <- rmvtpc(1000, c(0,1,2), U, 5)
joint.density.plot(X[,1], X[,2], color=TRUE)
```

---

dist.Normal.Inverse.Wishart

Normal-Inverse-Wishart Distribution

Description

These functions provide the density and random number generation for the normal-inverse-Wishart distribution.

Usage

```
dnorminvwishart(mu, mu0, lambda, Sigma, S, nu, log=FALSE)
rnorminvwishart(n=1, mu0, lambda, S, nu)
```

Arguments

`mu`	This is data or parameters in the form of a vector of length k or a matrix with k columns.
`mu0`	This is mean vector μ_0 with length k or matrix with k columns.
`lambda`	This is a positive-only scalar.
`n`	This is the number of random draws.
`nu`	This is the scalar degrees of freedom ν.
`Sigma`	This is a $k \times k$ covariance matrix Σ.
`S`	This is the symmetric, positive-semidefinite, $k \times k$ scale matrix $\mathbf{S}$.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\mu, \Sigma) = \mathcal{N}(\mu|\mu_0, \frac{1}{\lambda}\Sigma)\mathcal{W}^{-1}(\Sigma|\nu, \mathbf{S})$
- Inventors: Unknown
- Notation 1: $(\mu, \Sigma) \sim \mathcal{NIW}(\mu_0, \lambda, \mathbf{S}, \nu)$
- Notation 2: $p(\mu, \Sigma) = \mathcal{NIW}(\mu, \Sigma|\mu_0, \lambda, \mathbf{S}, \nu)$
- Parameter 1: location vector μ_0
- Parameter 2: $\lambda > 0$
- Parameter 3: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Parameter 4: degrees of freedom $\nu \geq k$
- Mean: Unknown
- Variance: Unknown
- Mode: Unknown

The normal-inverse-Wishart distribution, or Gaussian-inverse-Wishart distribution, is a multivariate four-parameter continuous probability distribution. It is the conjugate prior of a multivariate normal distribution with unknown mean and covariance matrix.

Value

`dnorminvwishart` gives the density and `rnorminvwishart` generates random deviates and returns a list with two components.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dmvn` and `dinvwishart`.

Examples

```
library(LaplacesDemon)
K <- 3
mu <- rnorm(K)
mu0 <- rnorm(K)
nu <- K + 1
S <- diag(K)
lambda <- runif(1) #Real scalar
Sigma <- as.positive.definite(matrix(rnorm(K^2),K,K))
x <- dnorminvwishart(mu, mu0, lambda, Sigma, S, nu, log=TRUE)
out <- rnorminvwishart(n=10, mu0, lambda, S, nu)
joint.density.plot(out$mu[,1], out$mu[,2], color=TRUE)
```

---

dist.Normal.Laplace *Normal-Laplace Distribution: Univariate Asymmetric*

Description

These functions provide the density and random generation for the univariate, asymmetric, normal-Laplace distribution with location parameter μ, scale parameter σ, and tail-behavior parameters α and β.

Usage

```
dnormlaplace(x, mu=0, sigma=1, alpha=1, beta=1, log=FALSE)
rnormlaplace(n, mu=0, sigma=1, alpha=1, beta=1)
```

Arguments

x	This is a vector of data.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the location parameter μ.
sigma	This is the scale parameter σ, which must be positive.
alpha	This is shape parameter α for left-tail behavior.
beta	This is shape parameter β for right-tail behavior.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\alpha\beta}{\alpha+\beta}\phi\frac{\theta-\mu}{\sigma}[R(\alpha\sigma - \frac{\theta-\mu}{\sigma}) + R(\beta\sigma + \frac{\theta-\mu}{\sigma})]$
- Inventor: Reed (2006)
- Notation 1: $\theta \sim \mathrm{NL}(\mu, \sigma, \alpha, \beta)$
- Notation 2: $p(\theta) = \mathrm{NL}(\theta|\mu, \sigma, \alpha, \beta)$

- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\sigma > 0$
- Parameter 3: shape parameter $\alpha > 0$
- Parameter 4: shape parameter $\beta > 0$
- Mean:
- Variance:
- Mode:

The normal-Laplace (NL) distribution is the convolution of a normal distribution and a skew-Laplace distribution. When the NL distribution is symmetric (when $\alpha = \beta$), it behaves somewhat like the normal distribution in the middle of its range, somewhat like the Laplace distribution in its tails, and functions generally between the normal and Laplace distributions. Skewness is parameterized by including a skew-Laplace component. It may be applied, for example, to the logarithmic price of a financial instrument.

Parameters α and β determine the behavior in the left and right tails, respectively. A small value corresponds to heaviness in the corresponding tail. As σ approaches zero, the NL distribution approaches a skew-Laplace distribution. As β approaches infinity, the NL distribution approaches a normal distribution, though it never quite reaches it.

Value

`dnormlaplace` gives the density, and `rnormlaplace` generates random deviates.

References

Reed, W.J. (2006). "The Normal-Laplace Distribution and Its Relatives". In *Advances in Distribution Theory, Order Statistics and Inference*, p. 61–74, Birkhauser, Boston.

See Also

`dalaplace`, `dallaplace`, `daml`, `dlaplace`, and `dnorm`

Examples

```
library(LaplacesDemon)
x <- dnormlaplace(1,0,1,0.5,2)
x <- rnormlaplace(100,0,1,0.5,2)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dlaplace(x,0,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dlaplace(x,0,1), type="l", col="green")
lines(x, dlaplace(x,0,2), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", lambda==0.5),
     paste(mu==0, ", ", lambda==1), paste(mu==0, ", ", lambda==2)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Normal.Mixture` *Mixture of Normal Distributions*

Description

These functions provide the density, cumulative, and random generation for the mixture of univariate normal distributions with probability p, mean μ and standard deviation σ.

Usage

```
dnormm(x, p, mu, sigma, log=FALSE)
pnormm(q, p, mu, sigma, lower.tail=TRUE, log.p=FALSE)
rnormm(n, p, mu, sigma)
```

Arguments

`x,q`	This is vector of values at which the density will be evaluated.
`p`	This is a vector of length M of probabilities for M components. The sum of the vector must be one.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`mu`	This is a vector of length M that is the mean parameter μ.
`sigma`	This is a vector of length M that is the standard deviation parameter σ, which must be positive.
`lower.tail`	Logical. This defaults to `TRUE`.
`log,log.p`	Logical. If `TRUE`, then probabilities p are given as $\log(p)$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \sum p_i \mathcal{N}(\mu_i, \sigma_i^2)$
- Inventor: Unknown
- Notation 1: $\theta \sim \mathcal{N}(\mu, \sigma^2)$
- Notation 2: $p(\theta) = \mathcal{N}(\theta|\mu, \sigma^2)$
- Parameter 1: mean parameters μ
- Parameter 2: standard deviation parameters $\sigma > 0$
- Mean: $E(\theta) = \sum p_i \mu_i$
- Variance: $var(\theta) = \sum p_i \sigma_i^{0.5}$
- Mode:

A mixture distribution is a probability distribution that is a combination of other probability distributions, and each distribution is called a mixture component, or component. A probability (or weight) exists for each component, and these probabilities sum to one. A mixture distribution (though not

these functions here in particular) may contain mixture components in which each component is a different probability distribution. Mixture distributions are very flexible, and are often used to represent a complex distribution with an unknown form. When the number of mixture components is unknown, Bayesian inference is the only sensible approach to estimation.

A normal mixture, or Gaussian mixture, distribution is a combination of normal probability distributions.

Value

`dnormm` gives the density, `pnormm` returns the CDF, and `rnormm` generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`ddirichlet` and `dnorm`.

Examples

```
library(LaplacesDemon)
p <- c(0.3,0.3,0.4)
mu <- c(-5, 1, 5)
sigma <- c(1,2,1)
x <- seq(from=-10, to=10, by=0.1)
plot(x, dnormm(x, p, mu, sigma, log=FALSE), type="l") #Density
plot(x, pnormm(x, p, mu, sigma), type="l") #CDF
plot(density(rnormm(10000, p, mu, sigma))) #Random Deviates
```

---

dist.Normal.Precision *Normal Distribution: Precision Parameterization*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate normal distribution with mean μ and precision τ.

Usage

```
dnormp(x, mean=0, prec=1, log=FALSE)
pnormp(q, mean=0, prec=1, lower.tail=TRUE, log.p=FALSE)
qnormp(p, mean=0, prec=1, lower.tail=TRUE, log.p=FALSE)
rnormp(n, mean=0, prec=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mean	This is the mean parameter μ.
prec	This is the precision parameter τ, which must be positive.
log, log.p	Logical. If TRUE, then probabilities p are given as $\log(p)$.
lower.tail	Logical. If TRUE (default), then probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \sqrt{\frac{\tau}{2\pi}} \exp(-\frac{\tau}{2}(\theta - \mu)^2)$
- Inventor: Carl Friedrich Gauss or Abraham De Moivre
- Notation 1: $\theta \sim \mathcal{N}(\mu, \tau^{-1})$
- Notation 2: $p(\theta) = \mathcal{N}(\theta|\mu, \tau^{-1})$
- Parameter 1: mean parameter μ
- Parameter 2: precision parameter $\tau > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \tau^{-1}$
- Mode: $mode(\theta) = \mu$

The normal distribution, also called the Gaussian distribution and the Second Law of Laplace, is usually parameterized with mean and variance, or in Bayesian inference, with mean and precision, where precision is the inverse of the variance. In contrast, Base R parameterizes the normal distribution with the mean and standard deviation. These functions provide the precision parameterization for convenience and familiarity.

Some authors attribute credit for the normal distribution to Abraham de Moivre in 1738. In 1809, Carl Friedrich Gauss published his monograph "Theoria motus corporum coelestium in sectionibus conicis solem ambientium", in which he introduced the method of least squares, method of maximum likelihood, and normal distribution, among many other innovations.

Gauss, himself, characterized this distribution according to mean and precision, though his definition of precision differed from the modern one. The modern Bayesian use of precision τ developed because it was more straightforward to estimate τ with a gamma distribution as a conjugate prior, than to estimate σ^2 with an inverse-gamma distribution as a conjugate prior.

Although the normal distribution is very common, it often does not fit data as well as more robust alternatives with fatter tails, such as the Laplace or Student t distribution.

A flat distribution is obtained in the limit as $\tau \to 0$.

For models where the dependent variable, y, is specified to be normally distributed given the model, the Jarque-Bera test (see plot.demonoid.ppc or plot.laplace.ppc) may be used to test the residuals.

These functions are similar to those in base R.

Value

dnormp gives the density, pnormp gives the distribution function, qnormp gives the quantile function, and rnormp generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dlaplace, dnorm, dnormv, prec2var, dst, dt, plot.demonoid.ppc, and plot.laplace.ppc.

Examples

```
library(LaplacesDemon)
x <- dnormp(1,0,1)
x <- pnormp(1,0,1)
x <- qnormp(0.5,0,1)
x <- rnormp(100,0,1)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dnormp(x,0,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dnormp(x,0,1), type="l", col="green")
lines(x, dnormp(x,0,5), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", tau==0.5),
     paste(mu==0, ", ", tau==1), paste(mu==0, ", ", tau==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Normal.Variance *Normal Distribution: Variance Parameterization*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate normal distribution with mean μ and variance σ^2.

Usage

```
dnormv(x, mean=0, var=1, log=FALSE)
pnormv(q, mean=0, var=1, lower.tail=TRUE, log.p=FALSE)
qnormv(p, mean=0, var=1, lower.tail=TRUE, log.p=FALSE)
rnormv(n, mean=0, var=1)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mean	This is the mean parameter μ.
var	This is the variance parameter σ^2, which must be positive.
log, log.p	Logical. If TRUE, then probabilities p are given as $\log(p)$.
lower.tail	Logical. If TRUE (default), then probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(\theta-\mu)^2}{2\sigma^2})$
- Inventor: Carl Friedrich Gauss or Abraham De Moivre
- Notation 1: $\theta \sim \mathcal{N}(\mu, \sigma^2)$
- Notation 2: $p(\theta) = \mathcal{N}(\theta|\mu, \sigma^2)$
- Parameter 1: mean parameter μ
- Parameter 2: variance parameter $\sigma^2 > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) = \sigma^2$
- Mode: $mode(\theta) = \mu$

The normal distribution, also called the Gaussian distribution and the Second Law of Laplace, is usually parameterized with mean and variance. Base R uses the mean and standard deviation. These functions provide the variance parameterization for convenience and familiarity. For example, it is easier to code dnormv(1,1,1000) than dnorm(1,1,sqrt(1000)).

Some authors attribute credit for the normal distribution to Abraham de Moivre in 1738. In 1809, Carl Friedrich Gauss published his monograph "Theoria motus corporum coelestium in sectionibus conicis solem ambientium", in which he introduced the method of least squares, method of maximum likelihood, and normal distribution, among many other innovations.

Gauss, himself, characterized this distribution according to mean and precision, though his definition of precision differed from the modern one.

Although the normal distribution is very common, it often does not fit data as well as more robust alternatives with fatter tails, such as the Laplace or Student t distribution.

A flat distribution is obtained in the limit as $\sigma^2 \to \infty$.

For models where the dependent variable, y, is specified to be normally distributed given the model, the Jarque-Bera test (see plot.demonoid.ppc or plot.laplace.ppc) may be used to test the residuals.

These functions are similar to those in base R.

Value

dnormv gives the density, pnormv gives the distribution function, qnormv gives the quantile function, and rnormv generates random deviates.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dlaplace, dnorm, dnormp, dst, dt, plot.demonoid.ppc, and plot.laplace.ppc.

Examples

```
library(LaplacesDemon)
x <- dnormv(1,0,1)
x <- pnormv(1,0,1)
x <- qnormv(0.5,0,1)
x <- rnormv(100,0,1)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dnormv(x,0,0.5), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dnormv(x,0,1), type="l", col="green")
lines(x, dnormv(x,0,5), type="l", col="blue")
legend(2, 0.9, expression(paste(mu==0, ", ", sigma^2==0.5),
     paste(mu==0, ", ", sigma^2==1), paste(mu==0, ", ", sigma^2==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Normal.Wishart *Normal-Wishart Distribution*

Description

These functions provide the density and random number generation for the normal-Wishart distribution.

Usage

```
dnormwishart(mu, mu0, lambda, Omega, S, nu, log=FALSE)
rnormwishart(n=1, mu0, lambda, S, nu)
```

Arguments

`mu`	This is data or parameters in the form of a vector of length k or a matrix with k columns.
`mu0`	This is mean vector μ_0 with length k or matrix with k columns.
`lambda`	This is a positive-only scalar.
`n`	This is the number of random draws.
`nu`	This is the scalar degrees of freedom ν.
`Omega`	This is a $k \times k$ precision matrix Ω.
`S`	This is the symmetric, positive-semidefinite, $k \times k$ scale matrix $\mathbf{S}$.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\mu, \Omega) = \mathcal{N}(\mu|\mu_0, (\lambda\Omega)^{-1})\mathcal{W}(\Omega|\nu, \mathbf{S})$
- Inventors: Unknown
- Notation 1: $(\mu, \Omega) \sim \mathcal{NW}(\mu_0, \lambda, \mathbf{S}, \nu)$
- Notation 2: $p(\mu, \Omega) = \mathcal{NW}(\mu, \Omega|\mu_0, \lambda, \mathbf{S}, \nu)$
- Parameter 1: location vector μ_0
- Parameter 2: $\lambda > 0$
- Parameter 3: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Parameter 4: degrees of freedom $\nu \geq k$
- Mean: Unknown
- Variance: Unknown
- Mode: Unknown

The normal-Wishart distribution, or Gaussian-Wishart distribution, is a multivariate four-parameter continuous probability distribution. It is the conjugate prior of a multivariate normal distribution with unknown mean and precision matrix.

Value

`dnormwishart` gives the density and `rnormwishart` generates random deviates and returns a list with two components.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dmvnp` and `dwishart`.

Examples

```
library(LaplacesDemon)
K <- 3
mu <- rnorm(K)
mu0 <- rnorm(K)
nu <- K + 1
S <- diag(K)
lambda <- runif(1) #Real scalar
Omega <- as.positive.definite(matrix(rnorm(K^2),K,K))
x <- dnormwishart(mu, mu0, lambda, Omega, S, nu, log=TRUE)
out <- rnormwishart(n=10, mu0, lambda, S, nu)
joint.density.plot(out$mu[,1], out$mu[,2], color=TRUE)
```

---

dist.Pareto *Pareto Distribution*

Description

These functions provide the density, distribution function, quantile function, and random generation for the pareto distribution.

Usage

```
dpareto(x, alpha, log=FALSE)
ppareto(q, alpha)
qpareto(p, alpha)
rpareto(n, alpha)
```

Arguments

x,q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
alpha	This is the shape parameter α, which must be positive.
log	Logical. If log=TRUE, then the logarithm of the density or result is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\alpha}{\theta^{\alpha+1}}, \theta \geq 1$
- Inventor: Vilfredo Pareto (1848-1923)
- Notation 1: $\theta \sim \mathcal{PA}(\alpha)$
- Notation 2: $p(\theta) = \mathcal{PA}(\theta|\alpha)$
- Parameter 1: shape parameter $\alpha > 0$

- Mean: $E(\theta) = \frac{\alpha}{\alpha-1}$
- Variance: $var(\theta) = \frac{\alpha}{(\alpha-1)^2(\alpha-2)}, \alpha > 2$
- Mode: $mode(\theta) = 1$

The Pareto distribution, sometimes called the Bradford distribution, is related to the exponential distribution. The gamma distribution is the conjugate prior distribution for the shape parameter α in the Pareto distribution. The Pareto distribution is the conjugate prior distribution for the range parameters of a uniform distribution. An extension, elsewhere, is the symmetric Pareto distribution.

Value

`dpareto` gives the density, `ppareto` gives the distribution function, `qpareto` gives the quantile function, and `rpareto` generates random deviates.

See Also

`dexp`, `dlnorm`, `dlnormp`, `dnorm`, `dnormp`, `dnormv`.

Examples

```
library(LaplacesDemon)
x <- dpareto(1,1)
x <- ppareto(0.5,1)
x <- qpareto(0.5,1)
x <- rpareto(10,1)

#Plot Probability Functions
x <- seq(from=1, to=5, by=0.01)
plot(x, dpareto(x,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dpareto(x,0.5), type="l", col="green")
lines(x, dpareto(x,1), type="l", col="blue")
legend(2, 0.9, expression(alpha==0.1, alpha==0.5, alpha==1),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Power.Exponential

Power Exponential Distribution: Univariate Symmetric

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, symmetric, power exponential distribution with location parameter μ, scale parameter σ, and kurtosis parameter κ.

Usage

```
dpe(x, mu=0, sigma=1, kappa=2, log=FALSE)
ppe(q, mu=0, sigma=1, kappa=2, lower.tail=TRUE, log.p=FALSE)
qpe(p, mu=0, sigma=1, kappa=2, lower.tail=TRUE, log.p=FALSE)
rpe(n, mu=0, sigma=1, kappa=2)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the location parameter μ.
sigma	This is the scale parameter σ, which must be positive.
kappa	This is the kurtosis parameter κ, which must be positive.
log,log.p	Logical. If log=TRUE, then the logarithm of the density or result is returned.
lower.tail	Logical. If lower.tail=TRUE (default), probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{1}{2\kappa^{1/\kappa}\Gamma(1+\frac{1}{\kappa})\sigma} \exp(-\frac{|\theta-\mu|^\kappa}{\kappa\sigma^\kappa})$
- Inventor: Subbotin, M.T. (1923)
- Notation 1: $\theta \sim \mathcal{PE}(\mu, \sigma, \kappa)$
- Notation 2: $p(\theta) = \mathcal{PE}(\theta|\mu, \sigma, \kappa)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\sigma > 0$
- Parameter 3: kurtosis parameter $\kappa > 0$
- Mean: $E(\theta) = \mu$
- Variance: $var(\theta) =$
- Mode: $mode(\theta) = \mu$

The power exponential distribution is also called the exponential power distribution, generalized error distribution, generalized Gaussian distribution, and generalized normal distribution. The original form was introduced by Subbotin (1923) and re-parameterized by Lunetta (1963). These functions use the more recent parameterization by Lunetta (1963). A shape parameter, $\kappa > 0$, is added to the normal distribution. When $\kappa = 1$, the power exponential distribution is the same as the Laplace distribution. When $\kappa = 2$, the power exponential distribution is the same as the normal distribution. As $\kappa \to \infty$, this becomes a uniform distribution $\in (\mu-\sigma, \mu+\sigma)$. Tails that are heavier than normal occur when $\kappa < 2$, or lighter than normal when $\kappa > 2$. This distribution is univariate and symmetric, and there exist multivariate and asymmetric versions.

These functions are similar to those in the normalp package.

Value

dpe gives the density, ppe gives the distribution function, qpe gives the quantile function, and rpe generates random deviates.

References

Lunetta, G. (1963). "Di una Generalizzazione dello Schema della Curva Normale". *Annali della Facolt'a di Economia e Commercio di Palermo*, 17, p. 237–244.

Subbotin, M.T. (1923). "On the Law of Frequency of Errors". *Matematicheskii Sbornik*, 31, p. 296–301.

See Also

dlaplace, dlaplacep, dmvpe, dnorm, dnormp, dnormv, and dunif.

Examples

```
library(LaplacesDemon)
x <- dpe(1,0,1,2)
x <- ppe(1,0,1,2)
x <- qpe(0.5,0,1,2)
x <- rpe(100,0,1,2)

#Plot Probability Functions
x <- seq(from=0.1, to=3, by=0.01)
plot(x, dpe(x,0,1,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dpe(x,0,1,2), type="l", col="green")
lines(x, dpe(x,0,1,5), type="l", col="blue")
legend(1.5, 0.9, expression(paste(mu==0, ", ", sigma==1, ", ", kappa==0.1),
     paste(mu==0, ", ", sigma==1, ", ", kappa==2),
     paste(mu==0, ", ", sigma==1, ", ", kappa==5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Scaled.Inverse.Wishart

Scaled Inverse Wishart Distribution

Description

These functions provide the density and random number generation for the scaled inverse Wishart distribution.

Usage

```
dsiw(Q, nu, S, zeta, mu, delta, log=FALSE)
rsiw(nu, S, mu, delta)
```

Arguments

`Q`	This is the symmetric, positive-definite $k \times k$ matrix $\mathbf{Q}$.
`nu`	This is the scalar degrees of freedom, ν regarding $\mathbf{Q}$. The default recommendation is `nu=k+1`.
`S`	This is the symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$ regarding $\mathbf{Q}$. The default recommendation is `S=diag(k)`.
`zeta`	This is a positive-only vector of length k of auxiliary scale parameters ζ.
`mu`	This is a vector of length k of location hyperparameters μ regarding ζ.
`delta`	This is a positive-only vector of length k of scale hyperparameters δ regarding ζ.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: (see below)
- Inventor: O'Malley and Zaslavsky (2005)
- Notation 1: $p(\Sigma) \sim \mathcal{SIW}(\mathbf{Q}, \nu, \mathbf{S}, \zeta, \mu, \delta)$
- Notation 2: $p(\Sigma) = \mathcal{SIW}(\Sigma|\mathbf{Q}, \nu, \mathbf{S}, \zeta, \mu, \delta$
- Parameter 1: symmetric, positive-definite $k \times k$ matrix $\mathbf{Q}$
- Parameter 2: degrees of freedom ν
- Parameter 3: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Parameter 4: Auxiliary scale parameter vector ζ
- Parameter 5: Hyperparameter location vector μ
- Parameter 6: Hyperparameter scale vector δ
- Mean:
- Variance:
- Mode:

The scaled inverse Wishart (SIW) distribution is a prior probability distribution for a covariance matrix, and is an alternative to the inverse Wishart distribution.

While the inverse Wishart distribution is applied directly to covariance matrix Σ, the SIW distribution is applied to a decomposed matrix $\mathbf{Q}$ and diagonal scale matrix ζ. For information on how to apply it to $\mathbf{Q}$, see the example below.

SIW is more flexible than the inverse Wishart distribution because it has additional, and some say somewhat redundant, scale parameters. This makes up for one limitation of the inverse Wishart, namely that all uncertainty about posterior variances is represented in one parameter. The SIW prior may somewhat alleviate the dependency in the inverse Wishart between variances and correlations, though the SIW prior still retains some of this relationship.

The Huang-Wand (dhuangwand) prior is a hierarchical alternative.

Value

`dsiw` gives the density and `rsiw` generates random deviates.

References

O'Malley, A.J. and Zaslavsky, A.M. (2005), "Domain-Level Covariance Analysis for Survey Data with Structured Nonresponse".

See Also

`dhuangwand`, `dinvwishartc`, `dmvn`, and `dwishart`.

Examples

```
library(LaplacesDemon)
### In the model specification function, input U and zeta, then:
# Q <- t(U) %*% U
# Zeta <- diag(zeta)
# Sigma <- Zeta %*% Q %*% Zeta
# Sigma.prior <- dsiw(Q, nu=Data$K+1, S=diag(Data$K), zeta, mu=0, delta=1)
### Examples
x <- dsiw(diag(3), 4, diag(3), runif(3), rep(0,3), rep(1,3), log=TRUE)
x <- rsiw(4, diag(3), rep(0,3), rep(1,3))
```

---

dist.Skew.Discrete.Laplace

Skew Discrete Laplace Distribution: Univariate

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, skew discrete Laplace distribution with parameters p and q.

Usage

```
dsdlaplace(x, p, q, log=FALSE)
psdlaplace(x, p, q)
qsdlaplace(prob, p, q)
rsdlaplace(n, p, q)
```

Arguments

`x`	This is a vector of data.
`p`	This is a scalar or vector of parameter $p \in [0, 1]$.
`q`	This is a scalar or vector of parameter $q \in [0, 1]$.
`prob`	This is a probability scalar or vector.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Discrete Univariate
- Density 1: $p(\theta) = \frac{(1-p)(1-q)}{1-pq} p^{\theta}; \theta = 0, 1, 2, 3, \ldots$
- Density 2: $p(\theta) = \frac{(1-p)(1-q)}{1-pq} q^{|\theta|}; x = 0, -1, -2, -3, \ldots$
- Inventor: Kozubowski, T.J. and Inusah, S. (2006)
- Notation 1: $\theta \sim \mathcal{DL}(p, q)$
- Notation 2: $p(\theta) = \mathcal{DL}(\theta|p, q)$
- Parameter 1: $p \in [0, 1]$
- Parameter 2: $q \in [0, 1]$
- Mean 1: $E(\theta) = \frac{1}{1-p} - \frac{1}{1-q} = \frac{p}{1-p} - \frac{q}{1-q}$
- Mean 2: $E(|\theta|) = \frac{q(1-p)^2+p(1-q)^2}{(1-qp)(1-q)(1-p)}$
- Variance: $var(\theta) = \frac{1}{(1-p)^2(1-q)^2}[\frac{q(1-p)^3(1+q)+p(1-q)^3(1+p)}{1-pq} - (p-q)^2]$
- Mode:

This is a discrete form of the skew-Laplace distribution. The symmetric discrete Laplace distribution occurs when $p = q$. DL(p,0) is a geometric distribution, and DL(0,q) is a geometric distribution of non-positive integers. The distribution is degenerate when DL(0,0). Since the geometric distribution is a discrete analog of the exponential distribution, the distribution of the difference of two geometric variables is a discrete Laplace distribution.

These functions are similar to those in the `DiscreteLaplace` package.

Value

`dslaplace` gives the density, `pslaplace` gives the distribution function, `qslaplace` gives the quantile function, and `rslaplace` generates random deviates.

References

Kozubowski, T.J. and Inusah, S. (2006). "A Skew Laplace Distribution on Integers". *AISM*, 58, p. 555–571.

See Also

`dalaplace`, `dexp`, `dlaplace`, `dlaplacep`, and `dslaplace`.

Examples

```
library(LaplacesDemon)
x <- dsdlaplace(1,0.5,0.5)
x <- psdlaplace(1,0.5,0.5)
x <- qsdlaplace(0.5,0.5,0.5)
x <- rsdlaplace(5,0.5,0.5)

#Plot Probability Functions
     x <- c(-3:3)
```

```
plot(x, dsdlaplace(x,0.5,0.5), ylim=c(0,0.6), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dsdlaplace(x,0.3,0.6), type="l", col="green")
lines(x, dsdlaplace(x,0.9,0.1), type="l", col="blue")
legend(-2.5, 0.5, expression(paste(p==0.5, ", ", q==0.5),
     paste(p==0.3, ", ", q==0.6),
     paste(p==0.9, ", ", q==0.1)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Skew.Laplace` *Skew-Laplace Distribution: Univariate*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate, skew-Laplace distribution with location parameter μ, and two mixture parameters: α and β.

Usage

```
dslaplace(x, mu, alpha, beta, log=FALSE)
pslaplace(q, mu, alpha, beta)
qslaplace(p, mu, alpha, beta)
rslaplace(n, mu, alpha, beta)
```

Arguments

`x, q`	These are each a vector of quantiles.
`p`	This is a vector of probabilities.
`n`	This is the number of observations, which must be a positive integer that has length 1.
`mu`	This is the location parameter μ.
`alpha`	This is a mixture parameter α, which must be positive.
`beta`	This is a mixture parameter β, which must be positive.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Continuous Univariate
- Density 1: $p(\theta) = \frac{1}{\alpha+\beta} \exp(\frac{\theta-\mu}{\alpha}), \theta \leq \mu$
- Density 2: $p(\theta) = \frac{1}{\alpha+\beta} \exp(\frac{\mu-\theta}{\beta}), \theta > \mu$
- Inventor: Fieller, et al. (1992)
- Notation 1: $\theta \sim \mathcal{SL}(\mu, \alpha, \beta)$
- Notation 2: $p(\theta) = \mathcal{SL}(\theta|\mu, \alpha, \beta)$

- Parameter 1: location parameter μ
- Parameter 2: mixture parameter $\alpha > 0$
- Parameter 3: mixture parameter $\beta > 0$
- Mean: $E(\theta) = \mu + \beta - \alpha$
- Variance: $var(\theta) = \alpha^2 + \beta^2$
- Mode: $mode(\theta) = \mu$

This is the three-parameter general skew-Laplace distribution, which is an extension of the two-parameter central skew-Laplace distribution. The general form allows the mode to be shifted along the real line with parameter μ. In contrast, the central skew-Laplace has mode zero, and may be reproduced here by setting $\mu = 0$.

The general skew-Laplace distribution is a mixture of a negative exponential distribution with mean β, and the negative of an exponential distribution with mean α. The weights of the positive and negative components are proportional to their means. The distribution is symmetric when $\alpha = \beta$, in which case the mean is μ.

These functions are similar to those in the `HyperbolicDist` package.

Value

`dslaplace` gives the density, `pslaplace` gives the distribution function, `qslaplace` gives the quantile function, and `rslaplace` generates random deviates.

References

Fieller, N.J., Flenley, E.C., and Olbricht, W. (1992). "Statistics of Particle Size Data". *Applied Statistics*, 41, p. 127–146.

See Also

`dalaplace`, `dexp`, `dlaplace`, `dlaplacep`, and `dsdlaplace`.

Examples

```
library(LaplacesDemon)
x <- dslaplace(1,0,1,1)
x <- pslaplace(1,0,1,1)
x <- qslaplace(0.5,0,1,1)
x <- rslaplace(100,0,1,1)

#Plot Probability Functions
x <- seq(from=0.1, to=3, by=0.01)
plot(x, dslaplace(x,0,1,1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dslaplace(x,0,0.5,2), type="l", col="green")
lines(x, dslaplace(x,0,2,0.5), type="l", col="blue")
legend(1.5, 0.9, expression(paste(mu==0, ", ", alpha==1, ", ", beta==1),
     paste(mu==0, ", ", alpha==0.5, ", ", beta==2),
     paste(mu==0, ", ", alpha==2, ", ", beta==0.5)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Stick` *Truncated Stick-Breaking Prior Distribution*

Description

These functions provide the density and random number generation of the original, truncated stick-breaking (TSB) prior distribution given θ and γ, as per Ishwaran and James (2001).

Usage

```
dStick(theta, gamma, log=FALSE)
rStick(M, gamma)
```

Arguments

`M`	This accepts an integer that is equal to one less than the number of truncated number of possible mixture components ($M = 1$). Unlike most random deviate functions, this is not the number of random deviates to return.
`theta`	This is θ, a vector of length $M-1$, where M is the truncated number of possible mixture components.
`gamma`	This is γ, a scalar, and is usually gamma-distributed.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.

Details

- Application: Discrete Multivariate
- Density: $p(\pi) = \frac{(1-\theta)^{\beta-1}}{\mathrm{B}(1,\beta)}$
- Inventor: Sethuraman, J. (1994)
- Notation 1: $\pi \sim \mathrm{Stick}(\theta, \gamma)$
- Notation 2: $\pi \sim \mathrm{GEM}(\theta, \gamma)$
- Notation 3: $p(\pi) = \mathrm{Stick}(\pi|\theta, \gamma)$
- Notation 4: $p(\pi) = \mathrm{GEM}(\pi|\theta, \gamma)$
- Parameter 1: shape parameter $\theta \in (0, 1)$
- Parameter 2: shape parameter $\gamma > 0$
- Mean: $E(\pi) = \frac{1}{1+\gamma}$
- Variance: $var(\pi) = \frac{\gamma}{(1+\gamma)^2(\gamma+2)}$
- Mode: $mode(\pi) = 0$

The original truncated stick-breaking (TSB) prior distribution assigns each θ to be beta-distributed with parameters $\alpha = 1$ and $\beta = \gamma$ (Ishwaran and James, 2001). This distribution is commonly used in truncated Dirichlet processes (TDPs).

Value

dStick gives the density and rStick generates a random deviate vector of length M.

References

Ishwaran, H. and James, L. (2001). "Gibbs Sampling Methods for Stick Breaking Priors". *Journal of the American Statistical Association*, 96(453), p. 161–173.

Sethuraman, J. (1994). "A Constructive Definition of Dirichlet Priors". *Statistica Sinica*, 4, p. 639–650.

See Also

ddirichlet, dmvpolya, and Stick.

Examples

```
library(LaplacesDemon)
dStick(runif(4), 0.1)
rStick(4, 0.1)
```

---

dist.Student.t *Student t Distribution: Univariate*

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate Student t distribution with location parameter μ, scale parameter σ, and degrees of freedom parameter ν.

Usage

```
dst(x, mu=0, sigma=1, nu=10, log=FALSE)
pst(q, mu=0, sigma=1, nu=10, lower.tail=TRUE, log.p=FALSE)
qst(p, mu=0, sigma=1, nu=10, lower.tail=TRUE, log.p=FALSE)
rst(n, mu=0, sigma=1, nu=10)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the location parameter μ.
sigma	This is the scale parameter σ, which must be positive.
nu	This is the degrees of freedom parameter ν, which must be positive.

`lower.tail` Logical. If `lower.tail=TRUE`, then probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.

`log, log.p` Logical. If `log=TRUE`, then the logarithm of the density or probability is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\Gamma[(\nu+1)/2]}{\Gamma(\nu/2)}\sqrt{\nu\pi}\sigma[1 + \frac{1}{\nu}[\frac{\theta-\mu}{\sigma}]^2]^{(-\nu+1)/2}$
- Inventor: William Sealy Gosset (1908)
- Notation 1: $\theta \sim \mathrm{t}(\mu, \sigma, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}(\theta|\mu, \sigma, \nu)$
- Parameter 1: location parameter μ
- Parameter 2: scale parameter $\sigma > 0$
- Parameter 3: degrees of freedom $\nu > 0$
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{\nu}{\nu-2}\sigma^2$, for $\nu > 2$
- Mode: $mode(\theta) = \mu$

The Student t-distribution is often used as an alternative to the normal distribution as a model for data. It is frequently the case that real data have heavier tails than the normal distribution allows for. The classical approach was to identify outliers and exclude or downweight them in some way. However, it is not always easy to identify outliers (especially in high dimensions), and the Student t-distribution is a natural choice of model-form for such data. It provides a parametric approach to robust statistics.

The degrees of freedom parameter, ν, controls the kurtosis of the distribution, and is correlated with the scale parameter σ. The likelihood can have multiple local maxima and, as such, it is often necessary to fix ν at a fairly low value and estimate the other parameters taking this as given. Some authors report that values between 3 and 9 are often good choices, and some authors suggest 5 is often a good choice.

In the limit $\nu \to \infty$, the Student t-distribution approaches $\mathcal{N}(\mu, \sigma^2)$. The case of $\nu = 1$ is the Cauchy distribution.

The `pst` and `qst` functions are similar to those in the `gamlss.dist` package.

Value

`dst` gives the density, `pst` gives the distribution function, `qst` gives the quantile function, and `rst` generates random deviates.

See Also

`dcauchy`, `dmvt`, `dmvtp`, `dnorm`, `dnormp`, `dnormv`, `dstp`, and `dt`.

Examples

```
library(LaplacesDemon)
x <- dst(1,0,1,10)
x <- pst(1,0,1,10)
x <- qst(0.5,0,1,10)
x <- rst(100,0,1,10)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
plot(x, dst(x,0,1,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dst(x,0,1,1), type="l", col="green")
lines(x, dst(x,0,1,10), type="l", col="blue")
legend(1, 0.9, expression(paste(mu==0, ", ", sigma==1, ", ", nu==0.5),
     paste(mu==0, ", ", sigma==1, ", ", nu==1),
     paste(mu==0, ", ", sigma==1, ", ", nu==10)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

---

dist.Student.t.Precision

Student t Distribution: Precision Parameterization

Description

These functions provide the density, distribution function, quantile function, and random generation for the univariate Student t distribution with location parameter μ, precision parameter τ, and degrees of freedom parameter ν.

Usage

```
dstp(x, mu=0, tau=1, nu=10, log=FALSE)
pstp(q, mu=0, tau=1, nu=10, lower.tail=TRUE, log.p=FALSE)
qstp(p, mu=0, tau=1, nu=10, lower.tail=TRUE, log.p=FALSE)
rstp(n, mu=0, tau=1, nu=10)
```

Arguments

x, q	These are each a vector of quantiles.
p	This is a vector of probabilities.
n	This is the number of observations, which must be a positive integer that has length 1.
mu	This is the location parameter μ.
tau	This is the precision parameter τ, which must be positive.
nu	This is the degrees of freedom parameter ν, which must be positive.
lower.tail	Logical. If lower.tail=TRUE, then probabilities are $Pr[X \leq x]$, otherwise, $Pr[X > x]$.
log, log.p	Logical. If log=TRUE, then the logarithm of the density or probability is returned.

Details

- Application: Continuous Univariate
- Density: $p(\theta) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)}\sqrt{\frac{\tau}{\nu\pi}}(1+\frac{\tau}{\nu}(\theta-\mu)^2)^{-(\nu+1)/2}$
- Inventor: William Sealy Gosset (1908)
- Notation 1: $\theta \sim \mathrm{t}(\mu, \sqrt{\tau^{-1}}, \nu)$
- Notation 2: $p(\theta) = \mathrm{t}(\theta|\mu, \sqrt{\tau^{-1}}, \nu)$
- Parameter 1: location parameter μ
- Parameter 2: precision parameter $\tau > 0$
- Parameter 3: degrees of freedom $\nu > 0$
- Mean: $E(\theta) = \mu$, for $\nu > 1$, otherwise undefined
- Variance: $var(\theta) = \frac{1}{\tau}\frac{\nu}{\nu-2}$, for $\nu > 2$
- Mode: $mode(\theta) = \mu$

The Student t-distribution is often used as an alternative to the normal distribution as a model for data. It is frequently the case that real data have heavier tails than the normal distribution allows for. The classical approach was to identify outliers and exclude or downweight them in some way. However, it is not always easy to identify outliers (especially in high dimensions), and the Student t-distribution is a natural choice of model-form for such data. It provides a parametric approach to robust statistics.

The degrees of freedom parameter, ν, controls the kurtosis of the distribution, and is correlated with the precision parameter τ. The likelihood can have multiple local maxima and, as such, it is often necessary to fix ν at a fairly low value and estimate the other parameters taking this as given. Some authors report that values between 3 and 9 are often good choices, and some authors suggest 5 is often a good choice.

In the limit $\nu \to \infty$, the Student t-distribution approaches $\mathcal{N}(\mu, \sigma^2)$. The case of $\nu = 1$ is the Cauchy distribution.

Value

`dstp` gives the density, `pstp` gives the distribution function, `qstp` gives the quantile function, and `rstp` generates random deviates.

See Also

`dcauchy`, `dmvt`, `dmvtp`, `dnorm`, `dnormp`, `dnormv`, `dst`, `dt`.

Examples

```
library(LaplacesDemon)
x <- dstp(1,0,1,10)
x <- pstp(1,0,1,10)
x <- qstp(0.5,0,1,10)
x <- rstp(100,0,1,10)

#Plot Probability Functions
x <- seq(from=-5, to=5, by=0.1)
```

```
plot(x, dstp(x,0,1,0.1), ylim=c(0,1), type="l", main="Probability Function",
     ylab="density", col="red")
lines(x, dstp(x,0,1,1), type="l", col="green")
lines(x, dstp(x,0,1,10), type="l", col="blue")
legend(1, 0.9, expression(paste(mu==0, ", ", tau==1, ", ", nu==0.5),
     paste(mu==0, ", ", tau==1, ", ", nu==1),
     paste(mu==0, ", ", tau==1, ", ", nu==10)),
     lty=c(1,1,1), col=c("red","green","blue"))
```

`dist.Truncated` *Truncated Distributions*

Description

Density, distribution function, quantile function and random generation for truncated distributions.

Usage

```
dtrunc(x, spec, a=-Inf, b=Inf, log=FALSE, ...)
extrunc(spec, a=-Inf, b=Inf, ...)
ptrunc(x, spec, a=-Inf, b=Inf, ...)
qtrunc(p, spec, a=-Inf, b=Inf, ...)
rtrunc(n, spec, a=-Inf, b=Inf, ...)
vartrunc(spec, a=-Inf, b=Inf, ...)
```

Arguments

`n`	This is a the number of random draws for `rtrunc`.
`p`	This is a vector of probabilities.
`x`	This is a vector to be evaluated.
`spec`	The base name of a probability distribution is specified here. For example, to estimate the density of a truncated normal distribution, enter `norm`.
`a`	This is the lower bound of truncation, which defaults to negative infinity.
`b`	This is the upper bound of truncation, which defaults to infinity.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.
`...`	Additional arguments pertain to the probability distribution specified in the `spec` argument.

Details

A truncated distribution is a conditional distribution that results from a priori restricting the domain of some other probability distribution. More than merely preventing values outside of truncated bounds, a proper truncated distribution integrates to one within the truncated bounds. For more information on propriety, see `is.proper`. In contrast to a truncated distribution, a censored distribution occurs when the probability distribution is still allowed outside of a pre-specified range. Here, distributions are truncated to the interval $[a, b]$, such as $p(\theta) \in [a, b]$.

The dtrunc function is often used in conjunction with the interval function to truncate prior probability distributions in the model specification function for use with these numerical approximation functions: LaplaceApproximation, LaplacesDemon, and PMC.

The R code of Nadarajah and Kotz (2006) has been modified to work with log-densities.

Value

dtrunc gives the density, extrunc gives the expectation, ptrunc gives the distribution function, qtrunc gives the quantile function, rtrunc generates random deviates, and vartrunc gives the variance of the truncated distribution.

References

Nadarajah, S. and Kotz, S. (2006). "R Programs for Computing Truncated Distributions". *Journal of Statistical Software*, 16, Code Snippet 2, p. 1–8.

See Also

interval, is.proper, LaplaceApproximation, LaplacesDemon, and PMC.

Examples

```
library(LaplacesDemon)
x <- seq(-0.5, 0.5, by = 0.1)
y <- dtrunc(x, "norm", a=-0.5, b=0.5, mean=0, sd=2)
```

---

dist.Wishart *Wishart Distribution*

Description

These functions provide the density and random number generation for the Wishart distribution.

Usage

```
dwishart(Omega, nu, S, log=FALSE)
rwishart(nu, S)
```

Arguments

Omega	This is the symmetric, positive-definite $k \times k$ matrix Ω.
nu	This is the scalar degrees of freedom ν.
S	This is the symmetric, positive-semidefinite, $k \times k$ scale matrix $\mathbf{S}$.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2^{\nu k/2}\pi^{k(k-1)/4}\prod_{i=1}^{k}\Gamma(\frac{\nu+1-i}{2}))^{-1}|\mathbf{S}|^{-nu/2}|\Omega|^{(nu-k-1)/2}\exp(-\frac{1}{2}tr(\mathbf{S}^{-1}\Omega))$
- Inventor: John Wishart (1928)
- Notation 1: $\Omega \sim \mathcal{W}_\nu(\mathbf{S})$
- Notation 2: $p(\Omega) = \mathcal{W}_\nu(\Omega|\mathbf{S})$
- Parameter 1: degrees of freedom $\nu \geq k$
- Parameter 2: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Mean: $E(\Omega) = \nu\mathbf{S}$
- Variance: $var(\Omega) = \nu(\mathbf{S}_{i,j}^2 + \mathbf{S}_{i,i}\mathbf{S}_{j,j})$
- Mode: $mode(\Omega) = (\nu - k - 1)\mathbf{S}$, for $\nu \geq k + 1$

The Wishart distribution is a generalization to multiple dimensions of the chi-square distribution, or, in the case of non-integer degrees of freedom, of the gamma distribution. However, the Wishart distribution is not called the multivariate chi-squared distribution because the marginal distribution of the off-diagonal elements is not chi-squared.

The Wishart is the conjugate prior distribution for the precision matrix Ω, the inverse of which (covariance matrix Σ) is used in a multivariate normal distribution.

The integral is finite when $\nu \geq k$, where ν is the scalar degrees of freedom parameter, and k is the dimension of scale matrix $\mathbf{S}$. The density is finite when $\nu gek + 1$, which is recommended.

The degrees of freedom, ν, is equivalent to specifying a prior sample size, indicating the confidence in $\mathbf{S}$, where $\mathbf{S}$ is a prior guess at the order of covariance matrix Σ. A flat prior distribution is obtained as $\nu \to 0$.

When applicable, the alternative Cholesky parameterization should be preferred. For more information, see dwishartc.

The Wishart prior lacks flexibility, having only one parameter, ν, to control the variability for all $k(k+1)/2$ elements. Popular choices for the scale matrix $\mathbf{S}$ include an identity matrix or sample covariance matrix. When the model sample size is small, the specification of the scale matrix can be influential.

Although the related inverse Wishart distribution has a dependency between variance and correlation, the Wishart distribution does not have this dependency.

The matrix gamma (dmatrixgamma) distribution is a more general version of the Wishart distribution, and the Yang-Berger (dyangberger) distribution is an alterative that is a least informative prior (LIP).

Value

`dwishart` gives the density and `rwishart` generates random deviates.

References

Wishart, J. (1928). "The Generalised Product Moment Distribution in Samples from a Normal Multivariate Population". *Biometrika*, 20A(1-2), p. 32–52.

See Also

dchisq, dgamma, dinvwishart, dmatrixgamma, dmvnp, dwishartc, Prec2Cov, and dyangberger.

Examples

```
library(LaplacesDemon)
x <- dwishart(matrix(c(2,-.3,-.3,4),2,2), 3, matrix(c(1,.1,.1,1),2,2))
x <- rwishart(3, matrix(c(1,.1,.1,1),2,2))
```

---

dist.Wishart.Cholesky *Wishart Distribution: Cholesky Parameterization*

Description

These functions provide the density and random number generation for the Wishart distribution with the Cholesky parameterization.

Usage

```
dwishartc(U, nu, S, log=FALSE)
rwishartc(nu, S)
```

Arguments

U	This is the upper-triangular $k \times k$ matrix for the Cholesky factor $\mathbf{U}$ of precision matrix Ω.
nu	This is the scalar degrees of freedom ν.
S	This is the symmetric, positive-semidefinite, $k \times k$ scale matrix $\mathbf{S}$.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = (2^{\nu k/2}\pi^{k(k-1)/4}\prod_{i=1}^{k}\Gamma(\frac{\nu+1-i}{2}))^{-1}|\mathbf{S}|^{-nu/2}|\Omega|^{(nu-k-1)/2}\exp(-\frac{1}{2}tr(\mathbf{S}^{-1}\Omega))$
- Inventor: John Wishart (1928)
- Notation 1: $\Omega \sim \mathcal{W}_\nu(\mathbf{S})$
- Notation 2: $p(\Omega) = \mathcal{W}_\nu(\Omega|\mathbf{S})$
- Parameter 1: degrees of freedom $\nu \geq k$
- Parameter 2: symmetric, positive-semidefinite $k \times k$ scale matrix $\mathbf{S}$
- Mean: $E(\Omega) = \nu\mathbf{S}$
- Variance: $var(\Omega) = \nu(\mathbf{S}_{i,j}^2 + \mathbf{S}_{i,i}\mathbf{S}_{j,j})$

- Mode: $mode(\Omega) = (\nu - k - 1)\mathbf{S}$, for $\nu \geq k + 1$

The Wishart distribution is a generalization to multiple dimensions of the chi-square distribution, or, in the case of non-integer degrees of freedom, of the gamma distribution. However, the Wishart distribution is not called the multivariate chi-squared distribution because the marginal distribution of the off-diagonal elements is not chi-squared.

The Wishart is the conjugate prior distribution for the precision matrix Ω, the inverse of which (covariance matrix Σ) is used in a multivariate normal distribution. In this parameterization, Ω has been decomposed to the upper-triangular Cholesky factor $\mathbf{U}$, as per `chol`.

The integral is finite when $\nu \geq k$, where ν is the scalar degrees of freedom parameter, and k is the dimension of scale matrix $\mathbf{S}$. The density is finite when $\nu gek + 1$, which is recommended.

The degrees of freedom, ν, is equivalent to specifying a prior sample size, indicating the confidence in $\mathbf{S}$, where $\mathbf{S}$ is a prior guess at the order of covariance matrix Σ. A flat prior distribution is obtained as $\nu \to 0$.

In practice, $\mathbf{U}$ is fully unconstrained for proposals when its diagonal is log-transformed. The diagonal is exponentiated after a proposal and before other calculations. Overall, the Cholesky parameterization is faster than the traditional parameterization. Compared with `dwishart`, `dwishartc` must additionally matrix-multiply the Cholesky back to the precision matrix, but it does not have to check for or correct the precision matrix to positive-semidefiniteness, which overall is slower. Compared with `rwishart`, `rwishartc` must additionally calculate a Cholesky decomposition, and is therefore slower.

The Wishart prior lacks flexibility, having only one parameter, ν, to control the variability for all $k(k + 1)/2$ elements. Popular choices for the scale matrix $\mathbf{S}$ include an identity matrix or sample covariance matrix. When the model sample size is small, the specification of the scale matrix can be influential.

Although the related inverse Wishart distribution has a dependency between variance and correlation, the Wishart distribution does not have this dependency.

The matrix gamma (`dmatrixgamma`) distribution is a more general version of the Wishart distribution, and the Yang-Berger (`dyangberger`) distribution is an alterative that is a least informative prior (LIP).

Value

`dwishartc` gives the density and `rwishartc` generates random deviates.

References

Wishart, J. (1928). "The Generalised Product Moment Distribution in Samples from a Normal Multivariate Population". *Biometrika*, 20A(1-2), p. 32–52.

See Also

`chol`, `dchisq`, `dgamma`, `dinvwishart`, `dinvwishartc`, `dmatrixgamma`, `dmvnp`, `dmvnpc`, `Prec2Cov`, and `dyangbergerc`.

Examples

```
library(LaplacesDemon)
Omega <- matrix(c(2,-.3,-.3,4),2,2)
U <- chol(Omega)
x <- dwishartc(U, 3, matrix(c(1,.1,.1,1),2,2))
x <- rwishartc(3, matrix(c(1,.1,.1,1),2,2))
```

---

dist.YangBerger *Yang-Berger Distribution*

Description

This is the density function for the Yang-Berger prior distribution for a covariance matrix or precision matrix.

Usage

```
dyangberger(x, log=FALSE)
dyangbergerc(x, log=FALSE)
```

Arguments

x	This is the $k \times k$ positive-definite covariance matrix or precision matrix for dyangberger or the Cholesky factor **U** of the covariance matrix or precision matrix for dyangbergerc.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \frac{1}{|\theta| \prod^{(d_j - d_{j-1})}}$, where d are increasing eigenvalues. See equation 13 in Yang and Berger (1994).
- Inventor: Yang and Berger (1994)
- Notation 1: $\theta \sim \mathcal{YB}$
- Mean:
- Variance:
- Mode:

Yang and Berger (1994) derived a least informative prior (LIP) for a covariance matrix or precision matrix. The Yang-Berger (YB) distribution does not have any parameters. It is a reference prior for objective Bayesian inference. The Cholesky parameterization is also provided here.

The YB prior distribution results in a proper posterior. It involves an eigendecomposition of the covariance matrix or precision matrix. It is difficult to interpret a model that uses the YB prior, due to a lack of intuition regarding the relationship between eigenvalues and correlations.

Compared to Jeffreys prior for a covariance matrix, this reference prior encourages equal eigenvalues, and therefore results in a covariance matrix or precision matrix with a better shrinkage of its eigenstructure.

Value

dyangberger and dyangbergerc give the density.

References

Yang, R. and Berger, J.O. (1994). "Estimation of a Covariance Matrix using the Reference Prior". *Annals of Statistics*, 2, p. 1195-1211.

See Also

dinvwishart and dwishart

Examples

```
library(LaplacesDemon)
X <- matrix(c(1,0.8,0.8,1), 2, 2)
dyangberger(X, log=TRUE)
```

---

dist.Zellner *Hyperprior-g Prior and Zellner's g-Prior*

Description

These functions provide the density of the hyper-g prior (Liang et al., 2008), and both the density and random generation of Zellner's g-prior (Zellner, 1986).

Usage

```
dhyperg(g, alpha=3, log=FALSE)
dzellner(beta, g, sigma, X, log=FALSE)
rzellner(n, g, sigma, X)
```

Arguments

alpha	This is a positive scale hyperhyperparameter that is proper when $\alpha > 2$. The default is alpha=3.
beta	This is regression effects β, a vector of length J.
g	This is hyperparameter g, a positive scalar.
n	This is the number of random deviates to generate.
sigma	This is the residual standard deviation σ, a positive scalar.
X	This is a full-rank $N \times J$ design matrix $\mathbf{X}$ for N records and J predictors, where $J + 1 < N$. Zellner's g-prior has been extended (elsewhere) via singular value decomposition (SVD) to the case where $J > N$.
log	Logical. If log=TRUE, then the logarithm of the density is returned.

Details

- Application: Continuous Multivariate
- Density: $p(\theta) = \frac{1}{(2\pi)^{J/2}|(g\sigma^2(\mathbf{X}^T\mathbf{X})^{-1})^{-1}|^{1/2}} \exp(-\frac{1}{2}(\theta-\mu)'(g\sigma^2(\mathbf{X}^T\mathbf{X})^{-1})^{-1}(\theta-\mu))$
- Inventor: Zellner, A. (1986)
- Notation 1: $\theta \sim \mathrm{N}_J(0, g\sigma^2(\mathbf{X}^T\mathbf{X})^{-1})$
- Notation 2: $p(\theta) = \mathrm{N}_J(\theta|g, \sigma^2, \mathbf{X})$
- Parameter 1: location parameter β
- Parameter 2: scale parameter $g > 0$
- Parameter 3: scale parameter $\sigma^2 > 0$
- Mean:
- Variance:
- Mode:

Zellner's g-prior is a popular, data-dependent, elliptical, improper, least-informative prior distribution on regression effects β in a Gaussian regression model. It is a particular form in the conjugate Normal-Gamma family. Zellner's g-prior is also used for estimating Bayes factors (for hypothesis testing) with a simpler form, as well as in model selection and variable selection. The marginal posterior distribution of regression effects β is multivariate t.

One of many nice properties of Zellner's g-prior is that it adapts automatically to near-collinearity between different predictors. Zellner's g-prior puts most of its prior mass in the direction that causes the regression coefficients of correlated predictors to be smoothed away from each other. When coupled with model selection, Zellner's g-prior discourages highly collinear predictors from entering the models simultaneously by inducing a negative correlation between the coefficients. However, when it is desirable for collinear predictors to enter simultaneously, a modification has been proposed (though not included here) in which $(\mathbf{X}^T\mathbf{X})^{-1}$ is replaced with $(\mathbf{X}^T\mathbf{X})^{\lambda}$. For more information, see Krishna et al. (2009).

For variable selection, large values of g, with a prior mean of zero for β, encourage models with few, large coefficients. Conversely, small values of g encourage saturated models with many, small coefficients.

The design matrix $\mathbf{X}$ is converted to Fisher's information matrix, which is used as a covariance matrix for β. This is computationally efficient, because each element of the covariance matrix does not need to be estimated as a parameter. When $\mathbf{X}$ is nearly singular, regression effects β may be poorly estimated.

Hyperparameter g acts as an inverse relative prior sample size, or as a dimensionality penalty. Zellner (1986) recommended that a hyperprior distribution is assigned to g so that it is estimated from the data, although in practice g has often been fixed, usually to N when no information is available, since it has the interpretation of adding prior information equivalent to one observation. A variety of hyperpriors have been suggested for g, such as in Bove and Held (2011), Liang et al. (2008), and Maruyama and George (2011). g becomes diffuse as it approaches infinity, and the Bayes factor approaches zero. The hyper-g prior of Liang et al. (2008) is proper when $\alpha > 2$, and any value in the interval $(2, 4]$ may be reasonable.

Value

`dhyperg` gives the density of the hyper-g prior of Liang et al. (2008), `dzellner` gives the density of Zellner's g-prior, and `rzellner` generates random deviates.

References

Bove, D.S. and Held, L. (2011). "Hyper-g Priors for Generalized Linear Models". *Bayesian Analysis*, 6(3), p. 387–410.

Krishna, A., Bondell, H.D., and Ghosh, S.K. (2009). "Bayesian Variable Selection Using an Adaptive Powered Correlation Prior". *Journal of Statistical Planning Inference*, 139(8), p. 2665-2674..

Liang, F., Paulo, R., Molina, G., Clyde, M.A., and Berger, J.O. (2008). "Mixtures of g Priors for Bayesian Variable Selection". *Journal of the American Statistical Association*, 103, p. 410–423.

Maruyama, Y. and George, E.I. (2011). "Fully Bayes Factors with a Generalised g-Prior". *Annals of Statistics*, 39, p. 2740–2765.

Zellner, A. (1986). "On Assessing Prior Distributions and Bayesian Regression Analysis with g-Prior Distributions". In *Bayesian Inference and Decision Techniques: Essays in Honor of Bruno de Finetti*, p. 233–243. Elsevier: Amsterdam, North Holland.

See Also

BayesFactor and dmvt

Examples

```
library(LaplacesDemon)
set.seed(667)
beta <- rnorm(10)
g <- 100
sigma <- 2
X <- cbind(1,matrix(rnorm(100*9),100,9))
dhyperg(g, alpha=3)
dzellner(beta, g, sigma, X)
rzellner(1, g, sigma, X)
```

Elicitation *Prior Elicitation*

Description

Prior elicitation is the act of inducing personal opinion to be expressed by the probabilities the person associates with an event (Savage, 1971). The `elicit` function elicits personal opinion and the `delicit` function estimates probability density to be used with model specification in the IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, or VariationalBayes functions.

Usage

```
delicit(theta, x, a=-Inf, b=Inf, log=FALSE)
elicit(n, cats, cat.names, show.plot=FALSE)
```

Arguments

`theta`	This is a scalar or vector of parameters for which the density is estimated with respect to the kernel density estimate of `x`.
`x`	This is the elicited vector.
`a`	This is an optional lower bound for support.
`b`	This is an optional upper bound for support.
`log`	Logical. If `log=TRUE`, then the logarithm of the density is returned.
`n`	This is the number of chips.
`cats`	This is a vector of k categories, bins, or intervals. When the variable is continuous, the mid-point of each category is used. For example, if the continuous interval [0,1] has 5 equal-sized categories, then `cats=c(0.1,0.3,0.5,0.7,0.9)`.
`cat.names`	This is a vector of category names. For example, if the continuous interval [0,1] has 5 equal-sized categories, then one way or naming the categories may be `cat.names=c("0:<.2", ".2:<.4", ".4:<.6", ".6:<.8", ".8:1")`.
`show.plot`	Logical. If `show.plot=TRUE`, then a barplot is shown after each allocation of chips.

Details

The `elicit` function elicits a univariate, discrete, non-conjugate, informative, prior probability distribution by offering a number of chips (specified as `n` by the statistician) for the user to allocate into categories specified by the statistician. The results of multiple elicitations (meaning, with multiple people), each the output of `elicit`, may be combined with the `c` function in base R.

This discrete distribution is included with the data for a model and supplied to a model specification function, where in turn it is supplied to the `delicit` function, which estimates the density at the current value of the prior distribution, $p(\theta)$. The prior distribution may be either continuous or discrete, will be proper, and may have bounded support (constrained to an interval).

For a minimal example, a statistician elicits the prior probability distribution for a regression effect, β. Non-statisticians would not be asked about expected parameters, but could be asked about how much **y** would be expected to change given a one-unit change in **x**. After consulting with others who have prior knowledge, the support does not need to be bounded, and their guesses at the range result in the statistician creating 5 catgories from the interval [-1,4], where each interval has a width of one. The statistician schedules time with 3 people, and each person participates when the statistician runs the following R code:

```
x <- elicit(n=10, cats=c(-0.5, 0.5, 1.5, 2.5, 3.5), cat.names=c("-1:<0", "0:<1", "1:<2",
"2:<3", "3:4"), show.plot=TRUE)
```

Each of the 3 participants receives 10 chips to allocate among the 5 categories according to personal beliefs in the probability of the regression effect. When the statistician and each participant accept their elicited distribution, all 3 vectors are combined into one vector. In the model form, the prior is expressed as

$$p(\beta) \sim \mathcal{EL}$$

and the code for the model specification is

```
elicit.prior <- delicit(beta, x, log=TRUE)
```

This method is easily extended to priors that are multivariate, correlated, or conditional.

As an alternative, Hahn (2006) also used a categorical approach, eliciting judgements about the relative likelihood of each category, and then minimizes the KLD (for more information on KLD, see the KLD function).

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Hahn, E.D. (2006). "Re-examining Informative Prior Elicitation Through the Lens of Markov chain Monte Carlo Methods". *Journal of the Royal Statistical Society*, A 169 (1), p. 37–48.

Savage, L.J. (1971). "Elicitation of Personal Probabilities and Expectations". *Journal of the American Statistical Association*, 66(336), p. 783–801.

See Also

de.Finetti.Game, KLD, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, and VariationalBayes.

Examples

```
library(LaplacesDemon)
x <- c(1,2,2,3,3,3,4,7,8,8,9,10) #Elicited with elicit function
theta <- seq(from=-5,to=15,by=.1)
plot(theta, delicit(theta,x), type="l", xlab=expression(theta),
     ylab=expression("p(" * theta * ")"))
```

ESS *Effective Sample Size due to Autocorrelation*

Description

This function may be used to estimate the effective sample size (ESS) (not to be confused with Elliptical Slice Sampling) of a continuous target distribution, where the sample size is reduced by autocorrelation. ESS is a measure of how well each continuous chain is mixing.

ESS is a univariate function that is often applied to each continuous, marginal posterior distribution. A multivariate form is not included. By chance alone due to multiple independent tests, 5% of the continuous parameters may indicate that ESS is below a user threshold of acceptability, such as 100, even when above the threshold. Assessing convergence is difficult.

Usage

```
ESS(x)
```

Arguments

x	This required argument is a vector or matrix of posterior samples.

Details

Effective Sample Size (ESS) was recommended by Radford Neal in the panel discussion of Kass et al. (1998). When a continuous, marginal posterior distribution is sampled with a Markov chain Monte Carlo (MCMC) algorithm, there is usually autocorrelation present in the samples. More autocorrelation is associated with less posterior sampled information, because the information in the samples is autocorrelated, or put another way, successive samples are not independent from earlier samples. This reduces the effective sample size of, and precision in representing, the continuous, marginal posterior distribution. `ESS` is one of the criteria in the `Consort` function, where stopping the MCMC updates is not recommended until $\texttt{ESS} \geq 100$. Although the need for precision of each modeler differs with each model, it is often a good goal to obtain $\texttt{ESS} = 1000$.

`ESS` is related to the integrated autocorrelation time (see `IAT` for more information).

ESS is usually defined as

$$\text{ESS}(\theta) = \frac{S}{1 + 2\sum_{k=1}^{\infty} \rho_k(\theta)},$$

where S is the number of posterior samples, ρ_k is the autocorrelation at lag k, and θ is the vector of marginal posterior samples. The infinite sum is often truncated at lag k when $\rho_k(\theta) < 0.05$. Just as with the `effectiveSize` function in the `coda` package, the `AIC` argument in the `ar` function is used to estimate the order.

ESS is a measure of how well each continuous chain is mixing, and a continuous chain mixes better when in the target distribution. This does not imply that a poorly mixing chain still searching for its target distribution will suddenly mix well after finding it, though mixing should improve. A poorly mixing continuous chain does not necessarily indicate problems. A smaller ESS is often due to correlated parameters, and is commonly found with scale parameters. Posterior correlation may be obtained from the `PosteriorChecks` function, and plotted with the `plotMatrix` function. Common remedies for poor mixing include re-parameterizing the model or trying a different MCMC algorithm that better handles correlated parameters. Slow mixing is indicative of an inefficiency in which a continuous chain takes longer to find its target distribution, and once found, takes longer to explore it. Therefore, slow mixing results in a longer required run-time to find and adequately represent the continuous target distribution, and increases the chance that the user may make inferences from a less than adequate representation of the continuous target distribution.

There are many methods of re-parameterization to improve mixing. It is helpful when predictors are centered and scaled, such as with the `CenterScale` function. Parameters for predictors are often assigned prior distributions that are independent per parameter, in which case an exchangeable prior distribution or a multivariate prior distribution may help. If a parameter with poor mixing is bounded with the `interval` function, then transforming it to the real line (such as with a log transformation for a scale parameter) is often helpful, since constraining a parameter to an interval often reduces ESS. Another method is to re-parameterize so that one or more latent variables represent the process that results in slow mixing. Such re-parameterization uses data augmentation.

This is numerically the same as the `effectiveSize` function in the `coda` package, but programmed to accept a simple vector or matrix so it does not require an `mcmc` or `mcmc.list` object, and the result is bound to be less than or equal to the original number of samples.

Value

A vector is returned, and each element is the effective sample size (ESS) for a corresponding column of x, after autocorrelation has been taken into account.

References

Kass, R.E., Carlin, B.P., Gelman, A., and Neal, R. (1998). "Markov Chain Monte Carlo in Practice: A Roundtable Discussion". *The American Statistician*, 52, p. 93–100.

See Also

CenterScale, Consort, IAT, interval, LaplacesDemon, plotMatrix, and PosteriorChecks.

Gelfand.Diagnostic *Gelfand's Convergence Diagnostic*

Description

Gelfand et al. (1990) proposed a convergence diagnostic for Markov chains. The Gelfand.Diagnostic function is an interpretation of Gelfand's "thick felt-tip pen" MCMC convergence diagnostic. This diagnostic plots a series of kernel density plots at k intervals of cumulative samples. Given a vector of S samples from a marginal posterior distribution, θ, multiple kernel density lines are plotted together, where each includes samples from a different interval. It is assumed that burnin iterations have been discarded.

Gelfand et al. (1990) assert that convergence is violated when the plotted lines are farther apart than the width of a thick, felt-tip pen. This depends on the size of the plot, and, of course, the pen. The estimated width of a "thick felt-tip pen" is included as a black, vertical line. The pen in Gelfand.Diagnostic is included for historical reasons. This diagnostic requires numerous samples.

Usage

```
Gelfand.Diagnostic(x, k=3, pen=FALSE)
```

Arguments

x	This required argument is a vector of marginal posterior samples, such as selected from the output of LaplacesDemon.
k	This argument specifies the number k of kernel density plots given cumulative intervals of samples. This argument defaults to $k = 3$.
pen	Logical. This argument defaults to pen=FALSE. When pen=TRUE, the thick felt-tip pen is included as a black, vertical line.

Value

The Gelfand.Diagnostic returns a plot.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Gelfand, A.E., Hills, S., Racine-Poon, A., and Smith, A.F.M. (1990). "Illustration of Bayesian Inference in Normal Data Models Using Gibbs Sampling". *Journal of the American Statistical Association*, 85, p. 972–985.

See Also

burnin and LaplacesDemon.

Examples

```
library(LaplacesDemon)
x <- rnorm(1000)
Gelfand.Diagnostic(x)
```

Gelman.Diagnostic *Gelman and Rubin's MCMC Convergence Diagnostic*

Description

Gelman and Rubin (1992) proposed a general approach to monitoring convergence of MCMC output in which $m > 1$ parallel chains are updated with initial values that are overdispersed relative to each target distribution, which must be normally distributed. Convergence is diagnosed when the chains have 'forgotten' their initial values, and the output from all chains is indistinguishable. The Gelman.Diagnostic function makes a comparison of within-chain and between-chain variances, and is similar to a classical analysis of variance. A large deviation between these two variances indicates non-convergence.

This diagnostic is popular as a stopping rule, though it requires parallel chains. The LaplacesDemon.hpc function is an extension of LaplacesDemon to enable parallel chains. As an alternative, the popular single-chain stopping rule is based on MCSE.

Usage

```
Gelman.Diagnostic(x, confidence=0.95, transform=FALSE)
```

Arguments

x	This required argument accepts an object of class demonoid.hpc, or a list of multiple objects of class demonoid, where the number of components in the list is the number of chains.
confidence	This is the coverage probability of the confidence interval for the potential scale reduction factor (PSRF).

`transform` Logical. If `TRUE`, then marginal posterior distributions in `x` may be transformed to improve the normality of the distribution, which is assumed. A log-transform is applied to marginal posterior distributions in the interval $(0, \infty]$, or a logit-transform is applied to marginal posterior distributions in the interval $(0, 1)$.

Details

To use the `Gelman.Diagnostic` function, the user must first have multiple MCMC chains for the same model, and three chains is usually sufficient. The easiest way to obtain multiple chains is with the `LaplacesDemon.hpc` function.

Although the `LaplacesDemon` function does not simultaneously update multiple MCMC chains, it is easy enough to obtain multiple chains, and if the computer has multiple processors (which is common), then multiple chains may be obtained simultaneously as follows. The model file may be opened in separate, concurrent R sessions, and it is recommended that a maximum number of sessions is equal to the number of processors, minus one. Each session constitutes its own chain, and the code is identical, except the initial values should be randomized with the `GIV` function so the chains begin in different places. The resulting object of class `demonoid` for each chain is saved, all objects are read into one session, put into a list, and passed to the `Gelman.Diagnostic` function.

Initial values must be overdispersed with respect to each target distribution, though these distributions are unknown in the beginning. Since the `Gelman.Diagnostic` function relies heavily on overdispersion with respect to the target distribution, the user should consider using MCMC twice, first to estimate the target distributions, and secondly to overdisperse initial values with respect to them. This may help identify multimodal target distributions. If multiple modes are found, it remain possible that more modes exist. When multiple modes are found, and if chains are combined with the `Combine` function, each mode is probably not represented in a proportion correct to the distribution.

The 'potential scale reduction factor' (PSRF) is an estimated factor by which the scale of the current distribution for the target distribution might be reduced if the simulations were continued for an infinite number of iterations. Each PSRF declines to 1 as the number of iterations approaches infinity. PSRF is also often represented as R-hat. PSRF is calculated for each marginal posterior distribution in `x`, together with upper and lower confidence limits. Approximate convergence is diagnosed when the upper limit is close to 1. The recommended proximity of each PSRF to 1 varies with each problem, but a general goal is to achieve PSRF < 1.1. PSRF is an estimate of how much narrower the posterior might become with an infinite number of iterations. When PSRF = 1.1, for example, it may be interpreted as a potential reduction of 10% in posterior interval width, given infinite iterations. The multivariate form bounds above the potential scale reduction factor for any linear combination of the (possibly transformed) variables.

The confidence limits are based on the assumption that the target distribution is stationary and normally distributed. The `transform` argument may be used to improve the normal approximation.

A large PSRF indicates that the between-chain variance is substantially greater than the within-chain variance, so that longer simulation is needed. If a PSRF is close to 1, then the associated chains are likely to have converged to one target distribution. A large PSRF (perhaps generally when a PSRF > 1.2) indicates convergence failure, and can indicate the presence of a multimodal marginal posterior distribution in which different chains may have converged to different local modes (see `is.multimodal`), or the need to update the associated chains longer, because burn-in (see `burnin`) has yet to be completed.

The `Gelman.Diagnostic` is essentially the same as the `gelman.diag` function in the `coda` package, but here it is programmed to work with objects of class `demonoid`.

There are two ways to estimate the variance of the stationary distribution: the mean of the empirical variance within each chain, W, and the empirical variance from all chains combined, which can be expressed as

$$\widehat{\sigma}^2 = \frac{(n-1)W}{n} + \frac{B}{n}$$

where n is the number of iterations and B/n is the empirical between-chain variance.

If the chains have converged, then both estimates are unbiased. Otherwise the first method will *underestimate* the variance, since the individual chains have not had time to range all over the stationary distribution, and the second method will *overestimate* the variance, since the initial values were chosen to be overdispersed (and this assumes the target distribution is known, see above).

This convergence diagnostic is based on the assumption that each target distribution is normal. A Bayesian probability interval (see `p.interval`) can be constructed using a t-distribution with mean

$$\widehat{\mu} = \text{Sample mean of all chains combined,}$$

variance

$$\widehat{V} = \widehat{\sigma}^2 + \frac{B}{mn},$$

and degrees of freedom estimated by the method of moments

$$d = \frac{2\widehat{V}^2}{\text{Var}(\widehat{V})}$$

Use of the t-distribution accounts for the fact that the mean and variance of the posterior distribution are estimated. The convergence diagnostic itself is

$$R = \sqrt{\frac{(d+3)\widehat{V}}{(d+1)W}}$$

Values substantially above 1 indicate lack of convergence. If the chains have not converged, then Bayesian probability intervals based on the t-distribution are too wide, and have the potential to shrink by this factor if the MCMC run is continued.

The multivariate version of Gelman and Rubin's diagnostic was proposed by Brooks and Gelman (1998). Unlike the univariate proportional scale reduction factor, the multivariate version does not include an adjustment for the estimated number of degrees of freedom.

Value

A list is returned with the following components:

`PSRF`	This is a list containing the point-estimates of the potential scale reduction factor (labelled `Point Est.`) and the associated upper confidence limits (labelled `Upper C.I.`).
`MPSRF`	This is the point-estimate of the multivariate potential scale reduction factor.

References

Brooks, S.P. and Gelman, A. (1998). "General Methods for Monitoring Convergence of Iterative Simulations". *Journal of Computational and Graphical Statistics*, 7, p. 434–455.

Gelman, A. and Rubin, D.B. (1992). "Inference from Iterative Simulation using Multiple Sequences". *Statistical Science*, 7, p. 457–511.

See Also

Combine, GIV, is.multimodal, LaplacesDemon, LaplacesDemon.hpc, MCSE, and p.interval.

Examples

```
#library(LaplacesDemon)
###After updating multiple chains with LaplacesDemon.hpc, do:
#Gelman.Diagnostic(Fit)
```

Geweke.Diagnostic *Geweke's Convergence Diagnostic*

Description

Geweke (1992) proposed a convergence diagnostic for Markov chains. This diagnostic is based on a test for equality of the means of the first and last part of a Markov chain (by default the first 10% and the last 50%). If the samples are drawn from a stationary distribution of the chain, then the two means are equal and Geweke's statistic has an asymptotically standard normal distribution.

The test statistic is a standard Z-score: the difference between the two sample means divided by its estimated standard error. The standard error is estimated from the spectral density at zero, and so takes into account any autocorrelation.

The Z-score is calculated under the assumption that the two parts of the chain are asymptotically independent.

The Geweke.Diagnostic is a univariate diagnostic that is usually applied to each marginal posterior distribution. A multivariate form is not included. By chance alone due to multiple independent tests, 5% of the marginal posterior distributions should appear non-stationary when stationarity exists. Assessing multivariate convergence is difficult.

Usage

```
Geweke.Diagnostic(x)
```

Arguments

x	This required argument is a vector or matrix of posterior samples, such as from the output of the LaplacesDemon function. Each column vector in a matrix is a chain to be assessed. A minimum of 100 samples are required.

Details

The Geweke.Diagnostic is essentially the same as the geweke.diag function in the coda package, but programmed to accept a simple vector or matrix, so it does not require an mcmc object.

Value

A vector is returned, in which each element is a Z-score for a test of equality that compares the means of the first and last parts of each chain supplied as x to Geweke.Diagnostic.

References

Geweke, J. (1992). "Evaluating the Accuracy of Sampling-Based Approaches to Calculating Posterior Moments". In *Bayesian Statistics 4* (ed JM Bernardo, JO Berger, AP Dawid, and AFM Smith). Clarendon Press, Oxford, UK.

See Also

burnin, is.stationary, and LaplacesDemon

Examples

```
library(LaplacesDemon)
Geweke.Diagnostic(rnorm(100))
Geweke.Diagnostic(matrix(rnorm(100),10,10))
```

GIV *Generate Initial Values*

Description

The GIV function generates initial values for use with the IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes functions.

Usage

```
GIV(Model, Data, n=1000, PGF=FALSE)
```

Arguments

Model	This required argument is a model specification function. For more information, see LaplacesDemon.
Data	This required argument is a list of data. For more information, see LaplacesDemon.
n	This is the number of attempts to generate acceptable initial values.
PGF	Logical. When TRUE, a Parameter-Generating Function (PGF) is required to be in Data, and GIV will generate initial values according to the user-specified PGF. This argument defaults to FALSE, in which case initial values are generated randomly without respect to a user-specified function.

Details

Initial values are required for optimization or sampling algorithms. A user may specify initial values, or use the GIV function for random generation. Initial values determined by the user may fail to produce a finite posterior in complicated models, and the GIV function is here to help.

GIV has several uses. First, the IterativeQuadrature, LaplaceApproximation, LaplacesDemon, and VariationalBayes functions use GIV internally if unacceptable initial values are discovered. Second, the user may use GIV when developing their model specification function, Model, to check for potential problems. Third, the user may prefer to randomly generate acceptable initial values. Lastly, GIV is recommended when running multiple or parallel chains with the LaplacesDemon.hpc function (such as for later use with the Gelman.Diagnostic) for dispersed starting locations. For dispersed starting locations, GIV should be run once for each parallel chain, and the results should be stored per row in a matrix of initial values. For more information, see the LaplacesDemon.hpc documentation for initial values.

It is strongly recommended that the user specifies a Parameter-Generating Function (PGF), and includes this function in the list of data. Although the PGF may be specified according to the prior distributions (possibly considered as a Prior-Generating Function), it is often specified with a more restricted range. For example, if a user has a model with the following prior distributions

$$\beta_j \sim \mathcal{N}(0, 1000), j = 1, \ldots, 5$$

$$\sigma \sim \mathcal{HC}(25)$$

then the PGF, given the prior distributions, is

```
PGF <- function(Data) return(c(rnormv(5,0,1000),rhalfcauchy(1,25)))
```

However, the user may not want to begin with initial values that could be so far from zero (as determined by the variance of 1000), and may instead prefer

```
PGF <- function(Data) return(c(rnormv(5,0,10),rhalfcauchy(1,5)))
```

When PGF=FALSE, initial values are attempted to be constrained to the interval $[-100, 100]$. This is done to prevent numeric overflows with parameters that are exponentiated within the model specification function. First, GIV passes the upper and lower bounds of this interval to the model, and any changed parameters are noted.

At this point, it is hoped that a non-finite posterior is not found. If found, then the remainder of the process is random and without the previous bounds. This can be particularly problematic in the case of, say, initial values that are the elements of a matrix that must be positive-definite, especially with large matrices. If a random solution is not found, then GIV will fail.

If the posterior is finite and PGF=FALSE, then initial values are randomly generated with a normal proposal and a small variance at the center of the returned range of each parameter. As GIV fails to find acceptable initial values, the algorithm iterates toward its maximum number of iterations, n. In each iteration, the variance increases for the proposal.

Initial values are considered acceptable only when the first two returned components of Model (which are LP and Dev) are finite, and when initial values do not change through constraints, as returned in the fifth component of the list: parm.

If GIV fails to return acceptable initial values, then it is best to study the model specification function. When the model is complicated, here is a suggestion. Remove the log-likelihood, LL, from the equation that calculates the logarithm of the unnormalized joint posterior density, LP. For example,

convert LP <- LL + beta.prior to LP <- beta.prior. Now, maximize LP, which is merely the set of prior densities, with any optimization algorithm. Replace LL, and run the model with initial values that are in regions of high prior density (preferably with PGF=TRUE. If this fails, then the model specification should be studied closely, because a non-finite posterior should (especially) never be associated with regions of high prior density.

Value

The GIV function returns a vector equal in length to the number of parameters, and each element is an initial value for the associated parameter in Data$parm.names. When GIV fails to find acceptable initial values, each returned element is NA.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

as.initial.values, Gelman.Diagnostic, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, and VariationalBayes.

Examples

```
library(LaplacesDemon)

#############################  Demon Data  ##############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

########################  Data List Preparation  ########################
mon.names <- c("LP","sigma")
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

########################  Model Specification  ########################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
```

```
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

########################  Generate Initial Values  ########################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)
```

`Hangartner.Diagnostic` *Hangartner's Convergence Diagnostic*

Description

Hangartner et al. (2011) proposed a convergence diagnostic for discrete Markov chains. A simple Pearson's Chi-squared test for two or more non-overlapping periods of a discrete Markov chain is a reliable diagnostic of convergence. It does not rely upon the estimation of spectral density, on suspect normality assumptions, or determining overdispersion within a small number of outcomes, all of which can be problematic with discrete measures. A discrete Markov chain is split into two or more non-overlapping windows. Two windows are recommended, and results may be sensitive to the number of selected windows, as well as sample size. As such, a user may try several window configurations before concluding there is no evidence of non-convergence.

As the number of discrete events in the sample space increases, this diagnostic becomes less appropriate and standard diagnostics become more appropriate.

Usage

```
Hangartner.Diagnostic(x, J=2)
```

Arguments

`x`	This required argument is a vector of marginal posterior samples of a discrete Markov chain, such as selected from the output of `LaplacesDemon`.
`J`	This argument specifies the number J of windows to be used, and defaults to $J = 2$.

Value

The `Hangartner.Diagnostic` returns an object of class `hangartner`, including the output from a Pearson's Chi-squared test. A frequentist p-value less than or equal to 0.05 is usually considered to be indicative of non-convergence.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Hangartner, D., Gill, J., and Cranmer, S., (2011). "An MCMC Diagnostic for Purely Discrete Parameters". Paper presented at the annual meeting of the Southern Political Science Association, Hotel InterContinental, New Orleans, Louisiana Online.

See Also

`LaplacesDemon` and `TransitionMatrix`.

Examples

```
library(LaplacesDemon)
N <- 1000
K <- 3
x <- rcat(N, rep(1/K,K))
hd <- Hangartner.Diagnostic(x, J=2)
hd
```

`Heidelberger.Diagnostic`

Heidelberger and Welch's MCMC Convergence Diagnostic

Description

Heidelberger and Welch (1981; 1983) proposed a two-part MCMC convergence diagnostic that calculates a test statistic (based on the Cramer-von Mises test statistic) to accept or reject the null hypothesis that the Markov chain is from a stationary distribution.

Usage

```
Heidelberger.Diagnostic(x, eps=0.1, pvalue=0.05)
```

Arguments

`x`	This required argument accepts an object of class `demonoid`. It attempts to use `Posterior2`, but when this is missing it uses `Posterior1`.
`eps`	This argument specifies the target value for the ratio of halfwidth to sample mean.
`pvalue`	This argument specifies the level of statistical significance.

Details

The Heidelberg and Welch MCMC convergence diagnostic consists of two parts:

First Part 1. Generate a chain of N iterations and define an alpha level. 2. Calculate the test statistic on the whole chain. Accept or reject the null hypothesis that the chain is from a stationary distribution. 3. If the null hypothesis is rejected, then discard the first 10% of the chain. Calculate the test statistic and accept or reject the null hypothesis. 4. If the null hypothesis is rejected, then discard the next 10% and calculate the test statistic. 5. Repeat until the null hypothesis is accepted or 50% of the chain is discarded. If the test still rejects the null hypothesis, then the chain fails the test and needs to be run longer.

Second Part If the chain passes the first part of the diagnostic, then the part of the chain that was not discarded from the first part is used to test the second part.

The halfwidth test calculates half the width of the (1 - alpha)% probability interval (credible interval) around the mean.

If the ratio of the halfwidth and the mean is lower than `eps`, then the chain passes the halfwidth test. Otherwise, the chain fails the halfwidth test and must be updated for more iterations until sufficient accuracy is obtained. In order to avoid problems caused by sequential testing, the test should not be repeated too frequently. Heidelberger and Welch (1981) suggest increasing the run length by a factor I > 1.5, each time, so that estimate has the same, reasonably large, proportion of new data.

The Heidelberger and Welch MCMC convergence diagnostic conducts multiple hypothesis tests. The number of potentially wrong results increases with the number of non-independent hypothesis tests conducted.

The `Heidelberger.Diagnostic` is a univariate diagnostic that is usually applied to each marginal posterior distribution. A multivariate form is not included. By chance alone due to multiple independent tests, 5% of the marginal posterior distributions should appear non-stationary when stationarity exists. Assessing multivariate convergence is difficult.

Value

The `Heidelberger.Diagnostic` function returns an object of class `heidelberger`. This object is a $J \times 6$ matrix, and it is intended to be summarized with the `print.heidelberger` function. Nonetheless, this object of class `heidelberger` has J rows, each of which corresponds to a Markov chain. The column names are `stest`, `start`, `pvalue`, `htest`, `mean`, and `halfwidth`. The `stest` column indicates convergence with a one, and non-convergence with a zero, regarding the stationarity test. When non-convergence is indicated, the remaining columns have missing values. The `start` column indicates the starting iteration, and the `pvalue` column shows the p-value associated with the first test. The `htest` column indicates convergence for the halfwidth test. The `mean` and `halfwidth` columns report the mean and halfwidth.

Note

The `Heidelberger.Diagnostic` function was adapted from the `heidel.diag` function in the coda package.

References

Heidelberger, P. and Welch, P.D. (1981). "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations". *Comm. ACM.*, 24, p. 233–245.

Heidelberger, P. and Welch, P.D. (1983). "Simulation Run Length Control in the Presence of an Initial Transient". *Opns Res.*, 31, p. 1109–1144.

Schruben, L.W. (1982). "Detecting Initialization Bias in Simulation Experiments". *Opns. Res.*, 30, p. 569–590.

See Also

burnin, is.stationary, LaplacesDemon, and print.heidelberger.

Examples

```
#library(LaplacesDemon)
###After updating with LaplacesDemon, do:
#hd <- Heidelberger.Diagnostic(Fit)
#print(hd)
```

---

hpc_server *Server Listening*

Description

This function is not intended to be called directly by the user. It is an internal-only function to prevent cluster problems while using the INCA algorithm in the LaplacesDemon.hpc function.

Usage

```
server_Listening(n=2, port=19009)
```

Arguments

n	This is the number of CPUs. For more information, see LaplacesDemon.hpc.
port	This is a port for server listening, and defaults to port 19009.

Details

For the INCA algorithm, a server has been built into the LaplacesDemon.hpc function. The server exchanges information between processes, and has been designed to be portable. The server_Listening function is run as a separate process via the system function, when INCA is selected in LaplacesDemon.hpc.

Socket connections and the serialize function are used as per the **Snow** package to update a single proposal covariance matrix given all parallel chains. The sockets are opened/closed in each process with a small random sleep time to avoid collisions during connections to the internal server of LaplacesDemon.hpc. Blocking sockets are used to synchronize processes.

Author(s)

Silvere Vialet-Chabrand <silvere@vialet-chabrand.com>

See Also

LaplacesDemon and LaplacesDemon.hpc.

IAT *Integrated Autocorrelation Time*

Description

The IAT function estimates integrated autocorrelation time, which is the computational inefficiency of a continuous chain or MCMC sampler. IAT is also called the IACT, ACT, autocorrelation time, autocovariance time, correlation time, or inefficiency factor. A lower value of IAT is better. IAT is a MCMC diagnostic that is an estimate of the number of iterations, on average, for an independent sample to be drawn, given a continuous chain or Markov chain. Put another way, IAT is the number of correlated samples with the same variance-reducing power as one independent sample.

IAT is a univariate function. A multivariate form is not included.

Usage

```
IAT(x)
```

Arguments

x	This requried argument is a vector of samples from a chain.

Details

IAT is a MCMC diagnostic that is often used to compare continuous chains of MCMC samplers for computational inefficiency, where the sampler with the lowest IATs is the most efficient sampler. Otherwise, chains may be compared within a model, such as with the output of LaplacesDemon to learn about the inefficiency of the continuous chain. For more information on comparing MCMC algorithmic inefficiency, see the Juxtapose function.

IAT is also estimated in the PosteriorChecks function. IAT is usually applied to a stationary, continuous chain after discarding burn-in iterations (see burnin for more information). The IAT of a continuous chain correlates with the variability of the mean of the chain, and relates to Effective Sample Size (ESS) and Monte Carlo Standard Error (MCSE).

IAT and ESS are inversely related, though not perfectly, because each is estimated a little differently. Given N samples and taking autocorrelation into account, ESS estimates a reduced number of M samples. Conversely, IAT estimates the number of autocorrelated samples, on average, required to produce one independently drawn sample.

The IAT function is similar to the IAT function in the Rtwalk package of Christen and Fox (2010), which is currently unavailabe on CRAN.

Value

The IAT function returns the integrated autocorrelation time of a chain.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Christen, J.A. and Fox, C. (2010). "A General Purpose Sampling Algorithm for Continuous Distributions (the t-walk)". *Bayesian Analysis*, 5(2), p. 263–282.

See Also

burnin, Compare, ESS, LaplacesDemon, MCSE, and PosteriorChecks.

Examples

```
library(LaplacesDemon)
theta <- rnorm(100)
IAT(theta)
```

Importance *Variable Importance*

Description

The Importance function considers variable importance (or predictor importance) to be the effect that the variable has on replicates $\mathbf{y}^{rep}$ (or $\mathbf{Y}^{rep}$) when the variable is removed from the model by setting it equal to zero. Here, variable importance is considered in terms of the comparison of posterior predictive checks. This may be considered to be a form of sensitivity analysis, and can be useful for model revision, variable selection, and model interpretation.

Currently, this function only tests the variable importance of design matrix $\mathbf{X}$.

Usage

```
Importance(object, Model, Data, Categorical=FALSE, Discrep, d=0, CPUs=1,
  Type="PSOCK")
```

Arguments

object	An object of class demonoid, iterquad, laplace, pmc, or vb is required.
Model	The model specification function is required.
Data	A data set in a list is required. The dependent variable is required to be named either y or Y. The Importance function will sequentially remove each column vector in X, so X is required to be in data set Data.
Categorical	Logical. If TRUE, then y and yhat are considered to be categorical (such as y=0 or y=1), rather than continuous. This defaults to FALSE.
Discrep	This optional argument allows a discrepancy statistic to be included. For more information on discrepancy statistics, see summary.demonoid.ppc.

d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0. For more information on discrepancy, see summary.demonoid.ppc.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".

Details

Variable importance is defined here as the impact of each variable (predictor, or column vector) in design matrix $\mathbf{X}$ on $\mathbf{y}^{rep}$ (or $\mathbf{Y}^{rep}$), when the variable is removed.

First, the full model is predicted with the predict.demonoid, predict.iterquad, predict.laplace, predict.pmc, or predict.vb function, and summarized with the summary.demonoid.ppc, summary.iterquad.ppc, summary.laplace.ppc, summary.pmc.ppc, or summary.vb.ppc function, respectively. The results are stored in the first row of the output. Each successive row in the output corresponds to the application of predict and summary functions, but with each variable in design matrix $\mathbf{X}$ being set to zero and effectively removed. The results show the impact of sequentially removing each predictor.

The criterion for variable importance may differ from model to model. As a default, BPIC is recommended. The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$.

With BPIC, variable importance has a positive relationship, such that larger values indicate a more important variable, because removing that variable resulted in a worse fit to the data. The best model has the lowest BPIC.

In a model in which the dependent variable is not categorical, it is also recommended to consider the L-criterion (Laud and Ibrahim, 1995), provided that sample size is small enough that it does not result in Inf. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".

With the L-criterion, variable importance has a positive relationship, such that larger values indicate a more important variable, because removing that variable resulted in a worse fit to the data. Ibrahim (1995) recommended considering the model with the lowest L-criterion, say as L_1, and the model with the closest L-criterion, say as L_2, and creating a comparison score as $\phi = (L_2 - L_1)/S_L$, where S.L is from the L_1 model. If the comparison score, ϕ is less than 2, then L_2 is within 2 standard deviations of L_1, and is the recommended cut-off for model choice.

The Importance function may suggest that a model fits the data better with a variable removed. In which case, the user may choose to leave the variable in the model (perhaps the model is misspecified without the variable), investigate and possibly re-specify the relationship between the independent and dependent variable(s), or remove the variable and update the model again.

In contrast to variable importance, the PosteriorChecks function calculates parameter importance, which is the probability that each parameter's marginal posterior distribution is greater than zero, where an important parameter does not include zero in its probability interval (see p.interval). Parameter importance and variable importance may disagree, and both should be studied.

The Importance function tends to indicate that a model fits the data better when variables are removed that have parameters with marginal posterior distributions that include 0 in the 95% probability interval (variables associated with lower parameter importance).

Often, in complicated models, it is difficult to assess variable importance by examining the marginal posterior distribution of the associated parameter(s). Consider polynomial regression, in which each variable may have multiple parameters.

The information provided by the Importance function may be used for model revision, or reporting the relative importance of variables.

The plot.importance function is available to plot the output of the Importance function according to BPIC, predictive concordance (Gelfand, 1996), the selected discrepancy statistic (Gelman et al., 1996), or the L-criterion.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

Value

Importance returns an object of class importance, which is a matrix with a number of rows equal to the number of columns in design matrix $\mathbf{X} + 1$ (including the full model), and 4 columns, which are BPIC, Concordance (or Mean.Lift if categorical), Discrep, and L-criterion. Each row represents a model with a predictor in $\mathbf{X}$ removed (except for the first row, which is the full model), and the resulting posterior predictive checks. For non-categorical dependent variables, an attribute is returned with the object, and the attribute is a vector of S.L, the calibration number of the L-criterion.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

is.importance, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, plot.importance, PosteriorChecks, p.interval, predict.demonoid, predict.iterquad, predict.laplace, predict.pmc, predict.vb, summary.demonoid.ppc, summary.iterquad.ppc, summary.laplace.ppc, summary.pmc.ppc, summary.vb.ppc, and VariationalBayes.

Examples

```
#First, update the model with the LaplacesDemon function, such as
#the example with linear regression, creating an object called Fit.
#Then
#Importance(Fit, Model, MyData, Discrep="Chi-Square", CPUs=1)
```

---

interval *Constrain to Interval*

Description

This function constrains the value(s) of a scalar, vector, matrix, or array to a specified interval, $[a, b]$. In Bayesian inference, it is often used both to truncate a parameter to an interval, such as $p(\theta) \in [a, b]$. The interval function is often used in conjunction with the dtrunc function to truncate the prior probability distribution associated with the constrained parameter. While dtrunc prevents assigning density outside of its interval and re-estimates density within the interval, the interval function is used to prevent the parameter from moving outside of the interval in the first place.

After the parameter is constrained to an interval in IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, or VariationalBayes, the constrained parameter should be updated back into the parm vector, so the algorithm knows it has been constrained.

This is unrelated to the probability interval (see p.interval and LPL.interval).

Usage

```
interval(x, a=-Inf, b=Inf, reflect=TRUE)
```

Arguments

x	This required argument is a scalar, vector, matrix or array, and its elements will be constrained to the interval [a,b].
a	This optional argument allows the specification of the lower bound of the interval, and defaults to -Inf.
b	This optional argument allows the specification of the upper bound of the interval, and defaults to Inf.
reflect	Logical. When TRUE, a value outside of the constrained interval is reflected or bounced back into the interval. When FALSE, a value outside of the interval is assigned the nearest boundary of the interval. This argument defaults to TRUE.

Details

It is common for a parameter to be constrained to an interval. The `interval` function provides two methods of constraining proposals. The default is to reflect an out-of-bounds proposal off of the boundaries until the proposal is within the specified interval. This is rare in the literature but works very well in practice. The other method does not reflect off of boundaries, but sets the value equal to the violated boundary. This is also rare in the literature and is not generally recommended.

If the `interval` function is unacceptable, then there are several alternatives.

It is common to re-parameterize by transforming the constrained parameter to the real line. For example, a positive-only scale parameter may be log-transformed. A parameter that is re-parameterized to the real line often mixes better in MCMC, exhibiting a higher effective sample size (ESS), and each evaluation of the model specification function is faster as well. However, without a hard constraint, it remains possible for the transformed parameter still become problematic, such as a log-transformed scale parameter that reaches negative infinity. This is much more common in the literature.

Another method is to allow the parameters to move outside of the desired, constrained interval in MCMC during the model update, and when the model update is finished, to discard any samples outside of the constraint boundaries. This is a method of rejecting unacceptable proposals in regions of zero probability. However, it is possible for parameters to remain outside of acceptable bounds long enough to be problematic.

In LaplacesDemon, the Gibbs sampler allows more control in the FC function, where a user can customize how constraints are handled.

Value

The `interval` function returns a scalar, vector, matrix, or array in accord with its argument, `x`. Each element is constrained to the interval [a,b].

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

dtrunc, ESS, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LPL.interval, PMC, p.interval, VariationalBayes.

Examples

```
#See the Examples vignette for numerous examples.
library(LaplacesDemon)
x <- 2
interval(x,0,1)
X <- matrix(runif(25,-2,2),5,5)
interval(X,-1,1)
```

`is.appeased` *Appeased*

Description

This function returns `TRUE` if Laplace's Demon is appeased by the object of class `demonoid`, and `FALSE` otherwise. If appeased, then the object passes several tests that indicate potential convergence of the Markov chains.

Usage

```
is.appeased(x)
```

Arguments

`x`	This is an object of class `demonoid`.

Details

After updating a model with the `LaplacesDemon` function, an output object is created. The output object is of class `demonoid`. The object may be passed to the `Consort` function, which will apply several criteria regarding the potential convergence of its Markov chains. If all criteria are met, then Laplace's Demon is appeased. Otherwise, Laplace's Demon suggests R code to be copy/pasted and executed. The `Consort` function prints a large amount of information to the screen. The `is.appeased` function may be applied as an alternative, though it only informs the user as to whether or not Laplace's Demon was appeased, as `TRUE` or `FALSE`.

Value

The `is.appeased` function returns a logical value indicating whether or not the supplied object passes several potential Markov chain convergence criteria. If the object passes all criteria, then Laplace's Demon is appeased, and the logical value returned is `TRUE`.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`Consort` and `LaplacesDemon`.

`is.bayesian` *Logical Check of a Bayesian Model*

Description

This function provides a logical test of whether or not a `Model` specification function is Bayesian.

Usage

```
is.bayesian(Model, Initial.Values, Data)
```

Arguments

`Model`	This is a model specification function. For more information, see the `LaplacesDemon` function.
`Initial.Values`	This is a vector of initial values, or current parameter values. For more information, see the `LaplacesDemon` function.
`Data`	This is a list of data. For more information, see the `LaplacesDemon` function.

Details

This function tests whether or not a model is Bayesian by comparing the first two returned arguments: the logarithm of the unnormalized joint posterior density (`LP`) and deviance (`Dev`). The deviance (D) is

$$\mathrm{D} = -2\mathrm{LL}$$

,
where LL is the log-likelihood. Consequently,

$$\mathrm{LL} = \mathrm{D}/-2$$

,
and LP is the sum of LL and prior probability densities. If LP = LL, then the model is not Bayesian, because prior densities are absent.

Value

The `is.bayesian` function returns a logical value of `TRUE` when the model is Bayesian, and `FALSE` otherwise.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`LaplacesDemon`.

`is.class` *Logical Check of Classes*

Description

These functions each provide a logical test of the class of an object.

Usage

```
is.bayesfactor(x)
is.blocks(x)
is.bmk(x)
is.demonoid(x)
is.demonoid.hpc(x)
is.demonoid.ppc(x)
is.demonoid.val(x)
is.hangartner(x)
is.heidelberger(x)
is.importance(x)
is.iterquad(x)
is.iterquad.ppc(x)
is.juxtapose(x)
is.laplace(x)
is.laplace.ppc(x)
is.miss(x)
is.pmc(x)
is.pmc.ppc(x)
is.pmc.val(x)
is.posteriorchecks(x)
is.raftery(x)
is.rejection(x)
is.sensitivity(x)
is.vb(x)
is.vb.ppc(x)
```

Arguments

`x`	This is an object that will be subjected to a logical test of its class.

Details

Functions in Laplace's Demon often assigns a class to an output object. For example, after updating a model with the `LaplacesDemon` or `LaplacesDemon.hpc` function, an output object is created. The output object is of class `demonoid` or `demonoid.hpc`, respectively. Likewise, after passing a model to the `LaplaceApproximation` function, an output object is created, and it is of class `laplace`. The class of these and other objects may be logically tested.

By assigning a class to an output object, the package is able to discern which other functions are appropriate for it. For example, after updating a model with LaplacesDemon, which creates an object of class demonoid, the user may desire to plot its output. Since it is assigned a class, the user may use the generic plot function, which internally selects the plot.demonoid function, which differs from plot.laplace for objects of class laplace.

For more information on object classes, see the class function.

Value

The is.bayesfactor function returns a logical value indicating whether or not the supplied object is of class bayesfactor.

The is.blocks function returns a logical value indicating whether or not the supplied object is of class blocks.

The is.bmk function returns a logical value indicating whether or not the supplied object is of class bmk.

The is.demonoid function returns a logical value indicating whether or not the supplied object is of class demonoid.

The is.demonoid.hpc function returns a logical value indicating whether or not the supplied object is of class demonoid.hpc.

The is.demonoid.ppc function returns a logical value indicating whether or not the supplied object is of class demonoid.ppc.

The is.demonoid.val function returns a logical value indicating whether or not the supplied object is of class demonoid.val.

The is.hangartner function returns a logical value indicating whether or not the supplied object is of class hangartner.

The is.heidelberger function returns a logical value indicating whether or not the supplied object is of class heidelberger.

The is.importance function returns a logical value indicating whether or not the supplied object is of class importance.

The is.iterquad function returns a logical value indicating whether or not the supplied object is of class iterquad.

The is.iterquad.ppc function returns a logical value indicating whether or not the supplied object is of class iterquad.ppc.

The is.juxtapose function returns a logical value indicating whether or not the supplied object is of class juxtapose.

The is.laplace function returns a logical value indicating whether or not the supplied object is of class laplace.

The is.laplace.ppc function returns a logical value indicating whether or not the supplied object is of class laplace.ppc.

The is.miss function returns a logical value indicating whether or not the supplied object is of class miss.

The is.pmc function returns a logical value indicating whether or not the supplied object is of class pmc.

The `is.pmc.ppc` function returns a logical value indicating whether or not the supplied object is of class `pmc.ppc`.

The `is.pmc.val` function returns a logical value indicating whether or not the supplied object is of class `pmc.val`.

The `is.posteriorchecks` function returns a logical value indicating whether or not the supplied object is of class `posteriorchecks`.

The `is.raftery` function returns a logical value indicating whether or not the supplied object is of class `raftery`.

The `is.rejection` function returns a logical value indicating whether or not the supplied object is of class `rejection`.

The `is.sensitivity` function returns a logical value indicating whether or not the supplied object is of class `sensitivity`.

The `is.vb` function returns a logical value indicating whether or not the supplied object is of class `vb`.

The `is.vb.ppc` function returns a logical value indicating whether or not the supplied object is of class `vb.ppc`.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

BayesFactor, Blocks, BMK.Diagnostic, class, Hangartner.Diagnostic, Heidelberger.Diagnostic, Importance, IterativeQuadrature, Juxtapose, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, MISS, PMC, PosteriorChecks, predict.demonoid, predict.laplace, predict.pmc, predict.vb, Raftery.Diagnostic, RejectionSampling, SensitivityAnalysis, Validate, and VariationalBayes.

---

is.constant *Logical Check of a Constant*

Description

This function provides a logical test of whether or not a vector is a constant.

Usage

```
is.constant(x)
```

Arguments

x	This is a vector.

Details

As opposed to a variable, a constant is a vector in which the elements contain less than or equal to one unique value.

Value

The `is.constant` function returns a logical result, reporting `TRUE` when a vector is a constant, or `FALSE` otherwise.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`unique`

Examples

```
library(LaplacesDemon)
is.constant(rep(1,10)) #TRUE
is.constant(1:10) #FALSE
```

`is.constrained` *Logical Check of Constraints*

Description

This function provides a logical test of constraints for each initial value or parameter for a model specification, given data.

Usage

```
is.constrained(Model, Initial.Values, Data)
```

Arguments

`Model`	This is a model specification function. For more information, see the `LaplacesDemon` function.
`Initial.Values`	This is a vector of initial values, or current parameter values. For more information, see the `LaplacesDemon` function.
`Data`	This is a list of data. For more information, see the `LaplacesDemon` function.

Details

This function is useful for testing whether or not initial values changed due to constraints when being passed through a `Model` specification function. If any initial value changes, then the constrained values that are ouput in the fifth component of the `Model` specification are suitable as initial values, not the tested initial values.

A parameter may be constrained and this function may not discover the constraint, since the discovery depends on the initial values and whether or not they change as they are passed through the model.

Value

The is.constrained function returns a logical vector, equal in length to the number of initial values. Each element receives TRUE if the corresponding initial value changed due to a constraint, or FALSE if it did not.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

LaplacesDemon.

---

is.data *Logical Check of Data*

Description

This function provides a logical test of whether or not a given list of data meets minimum criteria to be considered data for IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, or VariationalBayes.

Usage

```
is.data(Data)
```

Arguments

Data	This is a list of data. For more information, see the LaplacesDemon function.

Details

This function is useful for testing whether or not a list of data meets minimum criteria to be considered data in this package. The minimum requirements are that Data is a list, and it contains mon.names and parm.names.

This function is not extensive. For example, it does not match the length of parm.names with the length of Initial.Values, or compare the length of mon.names to the number of monitored variables output from the Model specification function. Additional checks are conducted in IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes.

Value

The is.data function returns a logical value. It returns TRUE if Data meets minimum requirements to be considered data in this package, and FALSE otherwise.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

IterativeQuadrature LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes.

`is.model` *Logical Check of a Model*

Description

This function provides a logical test of whether or not a `Model` specification function meets mininum requirements to be considered as such.

Usage

```
is.model(Model, Initial.Values, Data)
```

Arguments

`Model`	This is a model specification function. For more information, see the `LaplacesDemon` function.
`Initial.Values`	This is a vector of initial values, or current parameter values. For more information, see the `LaplacesDemon` function.
`Data`	This is a list of data. For more information, see the `LaplacesDemon` function.

Details

This function tests for minimum criteria for `Model` to be considered a model specification function. Specifically, it tests:

- `Model` must be a function
- `Model` must execute without errors
- `Model` must return a list
- `Model` must have five components in the list
- The first component must be named LP and have length 1
- The second component must be named Dev and have length 1
- The third component must be named Monitor
- The lengths of Monitor and mon.names must be equal
- The fourth component must be named yhat
- The fifth component must be named parm
- The lengths of parm and parm.names must be equal

This function is not extensive, and checks only for these minimum criteria. Additional checks are conducted in `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, and `VariationalBayes`.

Value

The `is.model` function returns a logical value of `TRUE` when `Model` meets minimum criteria of a model specification function, and `FALSE` otherwise.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, and `VariationalBayes`.

---

is.proper *Logical Check of Propriety*

Description

This function provides a logical check of the propriety of a univariate prior probability distribution or the joint posterior distribution.

Usage

```
is.proper(f, a, b, tol=1e-5)
```

Arguments

`f`	This is either a probability density function or an object of class `demonoid`, `laplace`, `pmc`, or `vb`.
`a`	This is the lower limit of integration, and may be negative infinity.
`b`	This is the upper limit of integration, and may be positive infinity.
`tol`	This is the tolerance, and indicates the allowable difference from one.

Details

A proper probability distribution is a probability distribution that integrates to one, and an improper probability distribution does not integrate to one. If a probability distribution integrates to any positive and finite value other than one, then it is an improper distribution, but is merely unnormalized. An unnormalized distribution may be multiplied by a constant so that it integrates to one.

In Bayesian inference, the posterior probability distribution should be proper. An improper prior distribution can cause an improper posterior distribution. When the posterior distribution is improper, inferences are invalid, it is non-integrable, and Bayes factors cannot be used (though there are exceptions).

To avoid these problems, it is suggested that the prior probability distribution should be proper, though it is possible to use an improper prior distribution and have it result in a proper posterior distribution.

To check the propriety of a univariate prior probability distribution, create a function `f`. For example, to check the propriety of a vague normal distribution, such as

$$\theta \sim \mathcal{N}(0, 1000)$$

the function is `function(x){dnormv(x,0,1000)}`. Next, set the lower and upper limits of integration, `a` and `b`. Internally, this function calls `integrate` from base R, which uses adaptive quadrature. By using $f(x)$ as shorthand for the specified function, `is.proper` will check to see if the area of the following integral is one:

$$\int_a^b f(x)dx$$

Multivariate prior probability distributions currently cannot be checked for approximate propriety. This is currently unavailable in this package.

To check the propriety of the joint posterior distribution, the only argument to be supplied is an object of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb`. The `is.proper` function checks the logarithm of the marginal likelihood (see `LML`) for a finite value, and returns `TRUE` when the LML is finite. This indicates that the marginal likelihood is finite for all observed $\mathbf{y}$ in the model data set. This implies:

$$\int p(\theta|\mathbf{y})p(\theta)d\theta < \infty$$

If the object is of class `demonoid` and the algorithm was adaptive, or if the object is of class `iterquad`, `laplace`, or `vb` and the algorithm did not converge, then `is.proper` will return `FALSE` because LML was not estimated. In this case, it is possible for the joint posterior to be proper, but `is.proper` will be unable to determine propriety without the estimate of LML. If desired, the `LML` may be estimated by the user, and if it is finite, then the joint posterior distribution is proper.

Value

The `is.proper` function returns a logical value indicating whether or not the univariate prior or joint posterior probability distribution integrates to one within its specified limits. `TRUE` is returned for a proper univariate probability distribution.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`dnormv`, `integrate`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `LML`, `PMC`, and `VariationalBayes`.

Examples

```
library(LaplacesDemon)
### Prior Probability Distribution
is.proper(function(x) {dnormv(x,0,1000)}, -Inf, Inf) #x ~ N(0,1000)
is.proper(function(x) {dhalfcauchy(x,25)}, 0, Inf) #x ~ HC(25)
is.proper(function(x) {dunif(x,0,1)}, 0, 1) #x ~ U(0,1)
is.proper(function(x) {dunif(x,-Inf,Inf)}, -Inf, Inf) #x ~ U(-Inf,Inf)
### Joint Posterior Distribution
##This assumes that Fit is an object of class demonoid, iterquad,
##  laplace, or pmc
#is.proper(Fit)
```

---

is.stationary *Logical Check of Stationarity*

Description

This function returns TRUE if the object is stationary according to the Geweke.Diagnostic function, and FALSE otherwise.

Usage

```
is.stationary(x)
```

Arguments

x	This is a vector, matrix, or object of class demonoid.

Details

Stationarity, here, refers to the limiting distribution in a Markov chain. A series of samples from a Markov chain, in which each sample is the result of an iteration of a Markov chain Monte Carlo (MCMC) algorithm, is analyzed for stationarity, meaning whether or not the samples trend or its moments change across iterations. A stationary posterior distribution is an equilibrium distribution, and assessing stationarity is an important diagnostic toward inferring Markov chain convergence.

In the cases of a matrix or an object of class demonoid, all Markov chains (as column vectors) must be stationary for is.stationary to return TRUE.

Alternative ways to assess stationarity of chains are to use the BMK.Diagnostic or Heidelberger.Diagnostic functions.

Value

is.stationary returns a logical value indicating whether or not the supplied object is stationary according to the Geweke.Diagnostic function.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

BMK.Diagnostic, Geweke.Diagnostic, Heidelberger.Diagnostic, and LaplacesDemon.

Examples

```
library(LaplacesDemon)
is.stationary(rnorm(100))
is.stationary(matrix(rnorm(100),10,10))
```

IterativeQuadrature *Iterative Quadrature*

Description

The IterativeQuadrature function iteratively approximates the first two moments of marginal posterior distributions of a Bayesian model with deterministic integration.

Usage

```
IterativeQuadrature(Model, parm, Data, Covar=NULL, Iterations=100,
     Algorithm="CAGH", Specs=NULL, Samples=1000, sir=TRUE,
     Stop.Tolerance=c(1e-5,1e-15), CPUs=1, Type="PSOCK")
```

Arguments

Model	This required argument receives the model from a user-defined function. The user-defined function is where the model is specified. IterativeQuadrature passes two arguments to the model function, parms and Data. For more information, see the LaplacesDemon function and "LaplacesDemon Tutorial" vignette.
parm	This argument requires a vector of initial values equal in length to the number of parameters. IterativeQuadrature will attempt to approximate these initial values for the parameters as means (or posterior modes) of normal integrals. The GIV function may be used to randomly generate initial values. Parameters must be continuous.
Data	This required argument accepts a list of data. The list of data must include mon.names which contains monitored variable names, and parm.names which contains parameter names.
Covar	This argument accepts a $J \times J$ covariance matrix for J initial values. When a covariance matrix is not supplied, a scaled identity matrix is used.
Iterations	This argument accepts an integer that determines the number of iterations that IterativeQuadrature will attempt to approximate the posterior with normal integrals. Iterations defaults to 100. IterativeQuadrature will stop before this number of iterations if the tolerance is less than or equal to the Stop.Tolerance criterion. The required amount of computer memory increases with Iterations. If computer memory is exceeded, then all will be lost.

`Algorithm` This optional argument accepts a quoted string that specifies the iterative quadrature algorithm. The default method is `Method="CAGH"`. Options include `"AGHSG"` for Adaptive Gauss-Hermite Sparse Grid, and `"CAGH"` for Componentwise Adaptive Gaussian-Hermite.

`Specs` This argument accepts a list of specifications for an algorithm.

`Samples` This argument indicates the number of posterior samples to be taken with sampling importance resampling via the `SIR` function, which occurs only when `sir=TRUE`. Note that the number of samples should increase with the number and intercorrelations of the parameters.

`sir` This logical argument indicates whether or not Sampling Importance Resampling (SIR) is conducted via the `SIR` function to draw independent posterior samples. This argument defaults to `TRUE`. Even when `TRUE`, posterior samples are drawn only when `IterativeQuadrature` has converged. Posterior samples are required for many other functions, including `plot.iterquad` and `predict.iterquad`. Less time can be spent on sampling by increasing `CPUs`, if available, which parallelizes the sampling.

`Stop.Tolerance` This argument accepts a vector of two positive numbers, and defaults to `1e-5,1e-15`. Tolerance is calculated each iteration, and the criteria varies by algorithm. The algorithm is considered to have converged to the user-specified `Stop.Tolerance` when the tolerance is less than or equal to the value of `Stop.Tolerance`, and the algorithm terminates at the end of the current iteration. Unless stated otherwise, the first element is the stop tolerance for the change in μ, the second element is the stop tolerance for the change in mean integration error, and the first tolerance must be met before the second tolerance is considered.

`CPUs` This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to `CPUs=1`, in which parallel processing does not occur. When multiple CPUs are specified, model function evaluations are parallelized across the nodes, and sampling with `SIR` is parallelized when `sir=TRUE`.

`Type` This argument specifies the type of parallel processing to perform, accepting either `Type="PSOCK"` or `Type="MPI"`.

Details

Quadrature is a historical term in mathematics that means determining area. Mathematicians of ancient Greece, according to the Pythagorean doctrine, understood determination of area of a figure as the process of geometrically constructing a square having the same area (squaring). Thus the name quadrature for this process.

In medieval Europe, quadrature meant the calculation of area by any method. With the invention of integral calculus, quadrature has been applied to the computation of a univariate definite integral. Numerical integration is a broad family of algorithms for calculating the numerical value of a definite integral. Numerical quadrature is a synonym for quadrature applied to one-dimensional integrals. Multivariate quadrature, also called cubature, is the application of quadrature to multidimensional integrals.

A quadrature rule is an approximation of the definite integral of a function, usually stated as a weighted sum of function values at specified points within the domain of integration. The specified points are referred to as abscissae, abscissas, integration points, or nodes, and have associated

weights. The calculation of the nodes and weights of the quadrature rule differs by the type of quadrature. There are numerous types of quadrature algorithms. Bayesian forms of quadrature usually use Gauss-Hermite quadrature (Naylor and Smith, 1982), and placing a Gaussian Process on the function is a common extension (O'Hagan, 1991; Rasmussen and Ghahramani, 2003) that is called 'Bayesian Quadrature'. Often, these and other forms of quadrature are also referred to as model-based integration.

Gauss-Hermite quadrature uses Hermite polynomials to calculate the rule. However, there are two versions of Hermite polynomials, which result in different kernels in different fields. In physics, the kernel is `exp(-x^2)`, while in probability the kernel is `exp(-x^2/2)`. The weights are a normal density. If the parameters of the normal distribution, μ and σ^2, are estimated from data, then it is referred to as adaptive Gauss-Hermite quadrature, and the parameters are the conditional mean and conditional variance. Outside of Gauss-Hermite quadrature, adaptive quadrature implies that a difficult range in the integrand is subdivided with more points until it is well-approximated. Gauss-Hermite quadrature performs well when the integrand is smooth, and assumes normality or multivariate normality. Adaptive Gauss-Hermite quadrature has been demonstrated to outperform Gauss-Hermite quadrature in speed and accuracy.

A goal in quadrature is to minimize integration error, which is the error between the evaluations and the weights of the rule. Therefore, a goal in Bayesian Gauss-Hermite quadrature is to minimize integration error while approximating a marginal posterior distribution that is assumed to be smooth and normally-distributed. This minimization often occurs by increasing the number of nodes until a change in mean integration error is below a tolerance, rather than minimizing integration error itself, since the target may be only approximately normally distributed, or minimizing the sum of integration error, which would change with the number of nodes.

To approximate integrals in multiple dimensions, one approach applies N nodes of a univariate quadrature rule to multiple dimensions (using the `GaussHermiteCubeRule` function for example) via the product rule, which results in many more multivariate nodes. This requires the number of function evaluations to grow exponentially as dimension increases. Multidimensional quadrature is usually limited to less than ten dimensions, both due to the number of nodes required, and because the accuracy of multidimensional quadrature algorithms decreases as the dimension increases. Three methods may overcome this curse of dimensionality in varying degrees: componentwise quadrature, sparse grids, and Monte Carlo.

Componentwise quadrature is the iterative application of univariate quadrature to each parameter. It is applicable with high-dimensional models, but sacrifices the ability to calculate the conditional covariance matrix, and calculates only the variance of each parameter.

Sparse grids were originally developed by Smolyak for multidimensional quadrature. A sparse grid is based on a one-dimensional quadrature rule. Only a subset of the nodes from the product rule is included, and the weights are appropriately rescaled. Although a sparse grid is more efficient because it reduces the number of nodes to achieve the same accuracy, the user must contend with increasing the accuracy of the grid, and it remains inapplicable to high-dimensional integrals.

Monte Carlo is a large family of sampling-based algorithms. O'Hagan (1987) asserts that Monte Carlo is frequentist, inefficient, regards irrelevant information, and disregards relevant information. Quadrature, he maintains (O'Hagan, 1992), is the most Bayesian approach, and also the most efficient. In high dimensions, he concedes, a popular subset of Monte Carlo algorithms is currently the best for cheap model function evaluations. These algorithms are called Markov chain Monte Carlo (MCMC). High-dimensional models with expensive model evaluation functions, however, are not well-suited to MCMC. A large number of MCMC algorithms is available in the `LaplacesDemon` function.

Following are some reasons to consider iterative quadrature rather than MCMC. Once an MCMC sampler finds equilibrium, it must then draw enough samples to represent all targets. Iterative quadrature does not need to continue drawing samples. Multivariate quadrature is consistently reported as more efficient than MCMC when its assumptions hold, though multivariate quadrature is limited to small dimensions. High-dimensional models therefore default to MCMC, between the two. Componentwise quadrature algorithms like CAGH, however, may also be more efficient with cloc-time than MCMC in high dimensions, especially against componentwise MCMC algorithms. Another reason to consider iterative quadrature are that assessing convergence in MCMC is a difficult topic, but not for iterative quadrature. A user of iterative quadrature does not have to contend with effective sample size and autocorrelation, assessing stationarity, acceptance rates, diminishing adaptation, etc. Stochastic sampling in MCMC is less efficient when samples occur in close proximity (such as when highly autocorrelated), whereas in quadrature the nodes are spread out by design.

In general, the conditional means and conditional variances progress smoothly to the target in multidimensional quadrature. For componentwise quadrature, movement to the target is not smooth, and often resembles a Markov chain or optimization algorithm.

Iterative quadrature is often applied after `LaplaceApproximation` to obtain a more reliable estimate of parameter variance or covariance than the negative inverse of the `Hessian` matrix of second derivatives, which is suitable only when the contours of the logarithm of the unnormalized joint posterior density are approximately ellipsoidal (Naylor and Smith, 1982, p. 224).

When `Algorithm="AGH"`, the Naylor and Smith (1982) algorithm is used. The AGH algorithm uses multivariate quadrature with the physicist's (not the probabilist's) kernel.

There are four algorithm specifications: `N` is the number of univariate nodes, `Nmax` is the maximum number of univariate nodes, `Packages` accepts any package required for the model function when parallelized, and `Dyn.libs` accepts dynamic libraries for parallelization, if required. The number of univariate nodes begins at N and increases by one each iteration. The number of multivariate nodes grows quickly with N. Naylor and Smith (1982) recommend beginning with as few nodes as $N = 3$. Any of the following events will cause N to increase by 1 when N is less than `Nmax`:

- All LP weights are zero (and non-finite weights are set to zero)
- μ does not result in an increase in LP
- All elements in Σ are not finite
- The square root of the sum of the squared changes in μ is less than or equal to the `Stop.Tolerance`

Tolerance includes two metrics: change in mean integration error and change in parameters. Including the change in parameters for tolerance was not mentioned in Naylor and Smith (1982).

Naylor and Smith (1982) consider a transformation due to correlation. This is not included here.

The AGH algorithm does not currently handle constrained parameters, such as with the `interval` function. If a parameter is constrained and changes during a model evaluation, this changes the node and the multivariate weight. This is currently not corrected.

An advantage of AGH over componentwise adaptive quadrature is that AGH estimates covariance, where a componentwise algorithm ignores it. A disadvantage of AGH over a componentwise algorithm is that the number of nodes increases so quickly with dimension, that AGH is limited to small-dimensional models.

When `Algorithm="AGHSG"`, the Naylor and Smith (1982) algorithm is applied to a sparse grid, rather than a traditional multivariate quadrature rule. This is identical to the AGH algorithm above, except that a sparse grid replaces the multivariate quadrature rule.

The sparse grid reduces the number of nodes. The cost of reducing the number of nodes is that the user must consider the accuracy, K.

There are four algorithm specifications: `K` is the accuracy (as a positive integer), `Kmax` is the maximum accuracy, `Packages` accepts any package required for the model function when parallelized, and `Dyn.libs` accepts dynamic libraries for parallelization, if required. These arguments represent accuracy rather than the number of univariate nodes, but otherwise are similar to the AGH algorithm.

When `Algorithm="CAGH"`, a componentwise version of the adaptive Gauss-Hermite quadrature of Naylor and Smith (1982) is used. Each iteration, each marginal posterior distribution is approximated sequentially, in a random order, with univariate quadrature. The conditional mean and conditional variance are also approximated each iteration, making it an adaptive algorithm.

There are four algorithm specifications: `N` is the number of nodes, `Nmax` is the maximum number of nodes, `Packages` accepts any package required for the model function when parallelized, and `Dyn.libs` accepts dynamic libraries for parallelization, if required. The number of nodes begins at N. All parameters have the same number of nodes. Any of the following events will cause N to increase by 1 when N is less than `Nmax`, and these conditions refer to all parameters (not individually):

- Any LP weights are not finite
- All LP weights are zero
- μ does not result in an increase in LP
- The square root of the sum of the squared changes in μ is less than or equal to the `Stop.Tolerance`

It is recommended to begin with `N=3` and set `Nmax` between 10 and 100. As long as CAGH does not experience problematic weights, and as long as CAGH is improving LP with μ, the number of nodes does not increase. When CAGH becomes either universally problematic or universally stable, then N slowly increases until the sum of both the mean integration error and the sum of the squared changes in μ is less than the `Stop.Tolerance` for two consecutive iterations.

If the highest LP occurs at the lowest or highest node, then the value at that node becomes the conditional mean, rather than calculating it from all weighted samples; this facilitates movement when the current integral is poorly centered toward a well-centered integral. If all weights are zero, then a random proposal is generated with a small variance.

Tolerance includes two metrics: change in mean integration error and change in parameters, as the square root of the sum of the squared differences.

When a parameter constraint is encountered, the node and weight of the quadrature rule is recalculated.

An advantage of CAGH over multidimensional adaptive quadrature is that CAGH may be applied in large dimensions. Disadvantages of CAGH are that only variance, not covariance, is estimated, and ignoring covariance may be problematic.

Value

IterativeQuadrature returns an object of class iterquad that is a list with the following components:

Algorithm	This is the name of the iterative quadrature algorithm.
Call	This is the matched call of IterativeQuadrature.
Converged	This is a logical indicator of whether or not IterativeQuadrature converged within the specified Iterations according to the supplied Stop.Tolerance criterion. Convergence does not indicate that the global maximum has been found, but only that the tolerance was less than or equal to the Stop.Tolerance criteria.
Covar	This is the estimated covariance matrix. The Covar matrix may be scaled and input into the Covar argument of the LaplacesDemon or PMC function for further estimation. To scale this matrix for use with Laplace's Demon or PMC, multiply it by $2.38^2/d$, where d is the number of initial values.
Deviance	This is a vector of the iterative history of the deviance in the IterativeQuadrature function, as it sought convergence.
History	This is a matrix of the iterative history of the parameters in the IterativeQuadrature function, as it sought convergence.
Initial.Values	This is the vector of initial values that was originally given to IterativeQuadrature in the parm argument.
LML	This is an approximation of the logarithm of the marginal likelihood of the data (see the LML function for more information). When the model has converged and sir=TRUE, the NSIS method is used. When the model has converged and sir=FALSE, the LME method is used. This is the logarithmic form of equation 4 in Lewis and Raftery (1997). As a rough estimate of Kass and Raftery (1995), the LME-based LML is worrisome when the sample size of the data is less than five times the number of parameters, and LML should be adequate in most problems when the sample size of the data exceeds twenty times the number of parameters (p. 778). The LME is inappropriate with hierarchical models. However LML is estimated, it is useful for comparing multiple models with the BayesFactor function.
LP.Final	This reports the final scalar value for the logarithm of the unnormalized joint posterior density.
LP.Initial	This reports the initial scalar value for the logarithm of the unnormalized joint posterior density.
LPw	This is the latest matrix of the logarithm of the unnormalized joint posterior density. It is weighted and normalized so that each column sums to one.
M	This is the final $N \times J$ matrix of quadrature weights that have been corrected for non-standard normal distributions, where N is the number of nodes and J is the number of parameters.
Minutes	This is the number of minutes that IterativeQuadrature was running, and this includes the initial checks as well as drawing posterior samples and creating summaries.

`Monitor` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the monitored variables.

`N` This is the final number of nodes.

`Posterior` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the parameters.

`Summary1` This is a summary matrix that summarizes the point-estimated posterior means. Uncertainty around the posterior means is estimated from the covariance matrix. Rows are parameters. The following columns are included: Mean, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Summary2` This is a summary matrix that summarizes the posterior samples drawn with sampling importance resampling (`SIR`) when `sir=TRUE`, given the point-estimated posterior modes and the covariance matrix. Rows are parameters. The following columns are included: Mean, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Tolerance.Final`
This is the last `Tolerance` of the `LaplaceApproxiation` algorithm.

`Tolerance.Stop` This is the `Stop.Tolerance` criteria.

`Z` This is the final $N \times J$ matrix of the conditional mean, where N is the number of nodes and J is the number of parameters.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Naylor, J.C. and Smith, A.F.M. (1982). "Applications of a Method for the Efficient Computation of Posterior Distributions". *Applied Statistics*, 31(3), p. 214–225.

O'Hagan, A. (1987). "Monte Carlo is Fundamentally Unsound". *The Statistician*, 36, p. 247–249.

O'Hagan, A. (1991). "Bayes-Hermite Quadrature". *Journal of Statistical Planning and Inference*, 29, p. 245–260.

O'Hagan, A. (1992). "Some Bayesian Numerical Analysis". In Bernardo, J.M., Berger, J.O., David, A.P., and Smith, A.F.M., editors, *Bayesian Statistics*, 4, p. 356–363, Oxford University Press.

Rasmussen, C.E. and Ghahramani, Z. (2003). "Bayesian Monte Carlo". In Becker, S. and Obermayer, K., editors, *Advances in Neural Information Processing Systems*, 15, MIT Press, Cambridge, MA.

See Also

`GaussHermiteCubeRule`, `GaussHermiteQuadRule`, `GIV`, `Hermite`, `Hessian`, `LaplaceApproximation`, `LaplacesDemon`, `LML`, `PMC`, `SIR`, and `SparseGrid`.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#####################  Load the LaplacesDemon Library  #####################
library(LaplacesDemon)

#############################  Demon Data  ##############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,10]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

########################  Data List Preparation  ########################
mon.names <- "mu[1]"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

########################  Model Specification  ########################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=mu[1],
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

#########################  Initial Values  #########################
#Initial.Values <- GIV(Model, MyData, PGF=TRUE)
Initial.Values <- rep(0,J+1)

########################  Adaptive Gauss-Hermite  ########################
#Fit <- IterativeQuadrature(Model, Initial.Values, MyData, Covar=NULL,
#     Iterations=100, Algorithm="AGH",
```

```
#     Specs=list(N=5, Nmax=7, Packages=NULL, Dyn.libs=NULL), CPUs=1)

##################  Adaptive Gauss-Hermite Sparse Grid  ##################
#Fit <- IterativeQuadrature(Model, Initial.Values, MyData, Covar=NULL,
#     Iterations=100, Algorithm="AGHSG",
#     Specs=list(K=5, Kmax=7, Packages=NULL, Dyn.libs=NULL), CPUs=1)

#################  Componentwise Adaptive Gauss-Hermite  #################
#Fit <- IterativeQuadrature(Model, Initial.Values, MyData, Covar=NULL,
#     Iterations=100, Algorithm="CAGH",
#     Specs=list(N=3, Nmax=10, Packages=NULL, Dyn.libs=NULL), CPUs=1)

#Fit
#print(Fit)
#PosteriorChecks(Fit)
#caterpillar.plot(Fit, Parms="beta")
#plot(Fit, MyData, PDF=FALSE)
#Pred <- predict(Fit, Model, MyData, CPUs=1)
#summary(Pred, Discrep="Chi-Square")
#plot(Pred, Style="Covariates", Data=MyData)
#plot(Pred, Style="Density", Rows=1:9)
#plot(Pred, Style="Fitted")
#plot(Pred, Style="Jarque-Bera")
#plot(Pred, Style="Predictive Quantiles")
#plot(Pred, Style="Residual Density")
#plot(Pred, Style="Residuals")
#Levene.Test(Pred)
#Importance(Fit, Model, MyData, Discrep="Chi-Square")

#End
```

`joint.density.plot` *Joint Density Plot*

Description

This function plots the joint kernel density from samples of two marginal posterior distributions.

Usage

```
joint.density.plot(x, y, Title=NULL, contour=TRUE, color=FALSE, Trace=NULL)
```

Arguments

`x,y`	These are vectors consisting of samples from two marginal posterior distributions, such as those output by `LaplacesDemon` in components `Posterior1` (all samples) or `Posterior2` (stationary samples).
`Title`	This is the title of the joint posterior density plot.

`contour` This logical argument indicates whether or not contour lines will be added to the plot. `contour` defaults to `TRUE`.

`color` This logical argument indicates whether or not color will be added to the plot. `color` defaults to `FALSE`.

`Trace` This argument defaults to `NULL`, in which case it does not trace the exploration of the joint density. To trace the exploration of the joint density, specify `Trace` with the beginning and ending iteration or sample. For example, to view the trace of the first ten iterations or samples, specify `Trace=c(1,10)`.

Details

This function produces either a bivariate scatterplot that may have kernel density contour lines added, or a bivariate plot with kernel density-influenced colors, which may also have kernel density contour lines added. A joint density plot may be more informative than two univariate density plots.

The `Trace` argument allows the user to view the exploration of the joint density, such as from MCMC chain output. An efficient algorithm jumps to random points of the joint density, and an inefficient algorithm explores more slowly. The initial point of the trace (which is the first element passed to `Trace`) is plotted with a green dot. The user should consider plotting the joint density of the two marginal posterior distributions with the highest `IAT`, as identified with the `PosteriorChecks` function, since these are the two least efficient MCMC chains. Different sequences of iterations may be plotted. This 'joint trace plot' may show behavior of the MCMC algorithm to the user.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`IAT`, `LaplacesDemon`, and `PosteriorChecks`

Examples

```
library(LaplacesDemon)
X <- rmvn(1000, runif(2), diag(2))
joint.density.plot(X[,1], X[,2], Title="Joint Density Plot",
     contour=TRUE, color=FALSE)
joint.density.plot(X[,1], X[,2], Title="Joint Density Plot",
     contour=FALSE, color=TRUE)
joint.density.plot(X[,1], X[,2], Title="Joint Density Plot",
     contour=TRUE, color=TRUE)
joint.density.plot(X[,1], X[,2], Title="Joint Trace Plot",
     contour=FALSE, color=TRUE, Trace=c(1,10))
```

`joint.pr.plot` *Joint Probability Region Plot*

Description

Given two vectors, the `joint.pr.plot` function creates a scatterplot with ellipses of probability regions.

Usage

```
joint.pr.plot(x, y, quantiles=c(0.25,0.50,0.75,0.95))
```

Arguments

`x`	This required argument is a vector.
`y`	This required argument is a vector.
`quantiles`	These are the quantiles for which probability regions are estimated with ellipses. The center of the ellipse is plotted by default. The 0.95 quantile creates a probability region that contains approximately 95% of the data or samples of `x` and `y`. By default, four quantiles are included.

Details

A probability region is also commonly called a credible region. For more information on probability regions, see `p.interval`.

Joint probability regions are plotted only for two variables, and the regions are estimated with functions modified from the `car` package. The internal ellipse functions assume bivariate normality.

This function is often used to plot posterior distributions of samples, such as from the `LaplacesDemon` function.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`LaplacesDemon` and `p.interval`

Examples

```
library(LaplacesDemon)
x <- rnorm(100)
y <- rnorm(100)
joint.pr.plot(x, y)
```

Juxtapose *Juxtapose MCMC Algorithm Inefficiency*

Description

This function gives a side-by-side comparison of (or juxtaposes) the inefficiency of MCMC algorithms in LaplacesDemon for applied use, and is a valuable tool for selecting what is likely to be the least inefficient algorithm for the user's current model, prior to updating the final, intended model.

Usage

```
Juxtapose(x)
```

Arguments

x	This is a list of multiple components. Each component must be an object of class demonoid.

Details

Laplace's Demon recommends using the Juxtapose function on the user's model (or most likely a simplified version of it) with a smaller, simulated data set to select the least inefficient MCMC algorithm before using real data and updating the model for numerous iterations. The least inefficient MCMC algorithm differs for different models and data sets. Using Juxtapose in this way does not guarantee that the selected algorithm will remain the best choice with real data, but it should be better than otherwise selecting an algorithm.

The user must make a decision regarding their model and data. The more similar the model and data is to the final, intended model and data, the more appropriate will be the results of the Juxtapose function. However, if the full model and data are used, then the user may as well instead skip using Juxtapose and proceed directly to LaplacesDemon. Replacing the actual data set with a smaller, simulated set is fairly straightforward, but the decision-making will most likely focus on what is the best way to reduce the full model specification. A simple approach may be to merely reduce the number of predictors. However, complicated models may have several components that slow down estimation time, and extend the amount of time until global stationarity is estimated. Laplace's Demon offers no guidance here, and leaves it in the realm of user discretion.

First, the user should simulate a smaller data set, and if best, reduce the model specification. Next, the user must select candidate algorithms. Then, the user must update each algorithm with LaplacesDemon for numerous iterations, with the goal of achieving stationarity for all parameters early in the iterations. Each update should begin with the same model specification function, vector of initial values, and data. Each output object of class demonoid should be renamed. An example follows.

Suppose a user considers three candidate algorithms for their model: AMWG, NUTS, and twalk. The user updates each model, saving the model that used the AMWG algorithm as, say, Fit1, the NUTS model as Fit2, and the twalk model as Fit3.

Next, the output model objects are put in a list and passed to the Juxtapose function. See the example below.

The `Juxtapose` function uses an internal version of the `IAT`, which is a slightly modified version of that found in the `SamplerCompare` package. The `Juxtapose` function returns an object of class `juxtapose`. It is a matrix in which each row is a result and each column is an algorithm.

The rows are:

- `iter.min`: This is the iterations per minute.
- `t.iter.min`: This is the thinned iterations per minute.
- `prop.stat`: This is the proportion of iterations that were stationary.
- `IAT.025`: This is the 2.5% quantile of the integrated autocorrelation time of the worst parameter, estimated only on samples when all parameters are estimated to be globally stationary.
- `IAT.500`: This is the median integrated autocorrelation time of the worst parameter, estimated only on samples when all parameters are estimated to be globally stationary.
- `IAT.975`: This is the 97.5% quantile of the integrated autocorrelation time of the worst parameter, estimated only on samples when all parameters are estimated to be globally stationary.
- `ISM.025`: This is the 2.5% quantile of the number of independent samples per minute.
- `ISM.500`: This is the median of the number of the independent samples per minute. The least inefficient MCMC algorithm has the highest `ISM.500`.
- `ISM.975`: This is the 97.5% quantile of the number of the independent samples per minute.

As for calculating ISM, let TIM be the observed number of thinned iterations per minute, PS be the percent of iterations in which all parameters were estimated to be globally stationary, and IAT_q be a quantile from a simulated distribution of the integrated autocorrelation time among the parameters.

$$ISM = \frac{PS \times TIM}{IAT_q}$$

There are various ways to measure the inefficiency of MCMC samplers. `IAT` is used perhaps most often. As with the `SamplerCompare` package, Laplace's Demon uses the worst parameter, in terms of `IAT`. Often, the number of evaluations or number of parameters is considered. The `Juxtapose` function, instead considers the final criterion of MCMC efficiency, in an applied context, to be `ISM`, or the number of Independent (thinned) Samples per Minute. The algorithm with the highest `ISM.500` is the best, or least inefficient, algorithm with respect to its worst `IAT`, the proportion of iterations required to seem to have global stationarity, and the number of (thinned) iterations per minute.

A disadvantage of using time is that it will differ by computer, and is less likely to be reported in a journal. The advantage, though, is that it is more meaningful to a user. Increases in the number of evaluations, parameters, and time should all correlate well, but time may enlighten a user as to expected run-time given the model just studied, even though the real data set will most likely be larger than the simulated data used initially. NUTS is an example of a sampler in which the number of evaluations varies per iteration. For an alternative approach, see Thompson (2010).

The `Juxtapose` function also adjusts `ISM` by `prop.stat`, the proportion of the iterations in which all chains were estimated to be stationary. This adjustment is weighted by burn-in iterations, penalizing an algorithm that took longer to achieve global stationarity. The goal, again, is to assist the user in selecting the least inefficient MCMC algorithm in an applied setting.

The Juxtapose function has many other potential uses than those described above. One additional use of the Juxtapose function is to compare inefficiencies within a single algorithm in which algorithmic specifications varied with different model updates. Another use is to investigate parallel chains in an object of class demonoid.hpc, as returned from the LaplacesDemon.hpc function. Yet another use is to compare the effects of small changes to a model specification function, such as with priors, or due to an increase in the amount of simulated data.

An object of class juxtapose may be plotted with the plot.juxtapose function, which displays ISM by default, or optionally IAT. For more information, see the plot.juxtapose function.

Independent samples per minute, calculated as ESS divided by minutes of run-time, are also available by parameter in the PosteriorChecks function.

Value

This function returns an object of class juxtapose. It is a $9 \times J$ matrix with nine results for J MCMC algorithms.

References

Thompson, M. (2010). "Graphical Comparison of MCMC Performance". ArXiv e-prints, eprint 1011.4458.

See Also

IAT, is.juxtapose, LaplacesDemon, LaplacesDemon.hpc, plot.juxtapose, and PosteriorChecks.

Examples

```
### Update three demonoid objects, each from different MCMC algorithms.
### Suppose Fit1 was updated with AFSS, Fit2 with AMWG, and
### Fit3 with NUTS. Then, compare the inefficiencies:
#Juxt <- Juxtapose(list(Fit1=Fit1, Fit2=Fit2, Fit3=Fit3)); Juxt
#plot(Juxt, Style="ISM")
```

KLD *Kullback-Leibler Divergence (KLD)*

Description

This function calculates the Kullback-Leibler divergence (KLD) between two probability distributions, and has many uses, such as in lowest posterior loss probability intervals, posterior predictive checks, prior elicitation, reference priors, and Variational Bayes.

Usage

```
KLD(px, py, base)
```

Arguments

px	This is a required vector of probability densities, considered as $p(\mathbf{x})$. Log-densities are also accepted, in which case both px and py must be log-densities.
py	This is a required vector of probability densities, considered as $p(\mathbf{y})$. Log-densities are also accepted, in which case both px and py must be log-densities.
base	This optional argument specifies the logarithmic base, which defaults to base=exp(1) (or e) and represents information in natural units (nats), where base=2 represents information in binary units (bits).

Details

The Kullback-Leibler divergence (KLD) is known by many names, some of which are Kullback-Leibler distance, K-L, and logarithmic divergence. KLD is an asymmetric measure of the difference, distance, or direct divergence between two probability distributions $p(\mathbf{y})$ and $p(\mathbf{x})$ (Kullback and Leibler, 1951). Mathematically, however, KLD is not a distance, because of its asymmetry.

Here, $p(\mathbf{y})$ represents the "true" distribution of data, observations, or theoretical distribution, and $p(\mathbf{x})$ represents a theory, model, or approximation of $p(\mathbf{y})$.

For probability distributions $p(\mathbf{y})$ and $p(\mathbf{x})$ that are discrete (whether the underlying distribution is continuous or discrete, the observations themselves are always discrete, such as from $i = 1, \ldots, N$),

$$\mathrm{KLD}[p(\mathbf{y})||p(\mathbf{x})] = \sum_{i}^{N} p(\mathbf{y}_i) \log \frac{p(\mathbf{y}_i)}{p(\mathbf{x}_i)}$$

In Bayesian inference, KLD can be used as a measure of the information gain in moving from a prior distribution, $p(\theta)$, to a posterior distribution, $p(\theta|\mathbf{y})$. As such, KLD is the basis of reference priors and lowest posterior loss intervals (LPL.interval), such as in Berger, Bernardo, and Sun (2009) and Bernardo (2005). The intrinsic discrepancy was introduced by Bernardo and Rueda (2002). For more information on the intrinsic discrepancy, see LPL.interval.

Value

KLD returns a list with the following components:

KLD.px.py	This is $\mathrm{KLD}_i[p(\mathbf{x}_i)||p(\mathbf{y}_i)]$.
KLD.py.px	This is $\mathrm{KLD}_i[p(\mathbf{y}_i)||p(\mathbf{x}_i)]$.
mean.KLD	This is the mean of the two components above. This is the expected posterior loss in LPL.interval.
sum.KLD.px.py	This is $\mathrm{KLD}[p(\mathbf{x})||p(\mathbf{y})]$. This is a directed divergence.
sum.KLD.py.px	This is $\mathrm{KLD}[p(\mathbf{y})||p(\mathbf{x})]$. This is a directed divergence.
mean.sum.KLD	This is the mean of the two components above.
intrinsic.discrepancy	This is minimum of the two directed divergences.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Berger, J.O., Bernardo, J.M., and Sun, D. (2009). "The Formal Definition of Reference Priors". *The Annals of Statistics*, 37(2), p. 905–938.

Bernardo, J.M. and Rueda, R. (2002). "Bayesian Hypothesis Testing: A Reference Approach". *International Statistical Review*, 70, p. 351–372.

Bernardo, J.M. (2005). "Intrinsic Credible Regions: An Objective Bayesian Approach to Interval Estimation". *Sociedad de Estadistica e Investigacion Operativa*, 14(2), p. 317–384.

Kullback, S. and Leibler, R.A. (1951). "On Information and Sufficiency". *The Annals of Mathematical Statistics*, 22(1), p. 79–86.

See Also

`LPL.interval` and `VariationalBayes`.

Examples

```
library(LaplacesDemon)
px <- dnorm(runif(100),0,1)
py <- dnorm(runif(100),0.1,0.9)
KLD(px,py)
```

KS.Diagnostic *Kolmogorov-Smirnov Convergence Diagnostic*

Description

The Kolmogorov-Smirnov test is a nonparametric test of stationarity that has been applied as an MCMC diagnostic (Brooks et al, 2003), such as to the posterior samples from the `LaplacesDemon` function. The first and last halves of the chain are compared. This test assumes IID, which is violated in the presence of autocorrelation.

The `KS.Diagnostic` is a univariate diagnostic that is usually applied to each marginal posterior distribution. A multivariate form is not included. By chance alone due to multiple independent tests, 5% of the marginal posterior distributions should appear non-stationary when stationarity exists. Assessing multivariate convergence is difficult.

Usage

```
KS.Diagnostic(x)
```

Arguments

`x`	This is a vector of posterior samples for which a Kolmogorov-Smirnov test will be applied that compares the first and last halves for stationarity.

Details

There are two main approaches to using the Kolmogorov-Smirnov test as an MCMC diagnostic. There is a version of the test that has been adapted to account for autocorrelation (and is not included here). Otherwise, the chain is thinned enough that autocorrelation is not present or is minimized, in which case the two-sample Kolmogorov-Smirnov test is applied. The CDFs of both samples are compared. The `ks.test` function in base R is used.

The advantage of the Kolmogorov-Smirnov test is that it is easier and faster to calculate. The disadvantages are that autocorrelation biases results, and the test is generally biased on the conservative side (indicating stationarity when it should not).

Value

The `KS.Diagnostic` function returns a frequentist p-value, and stationarity is indicated when $p > 0.05$.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Brooks, S.P., Giudici, P., and Philippe, A. (2003). "Nonparametric Convergence Assessment for MCMC Model Selection". *Journal of Computational and Graphical Statistics*. 12(1), p. 1–22.

See Also

`is.stationary`, `ks.test`, and `LaplacesDemon`.

Examples

```
library(LaplacesDemon)
x <- rnorm(1000)
KS.Diagnostic(x)
```

`LaplaceApproximation` *Laplace Approximation*

Description

The `LaplaceApproximation` function deterministically maximizes the logarithm of the unnormalized joint posterior density with one of several optimization algorithms. The goal of Laplace Approximation is to estimate the posterior mode and variance of each parameter. This function is useful for optimizing initial values and estimating a covariance matrix to be input into the `IterativeQuadrature`, `LaplacesDemon`, `PMC`, or `VariationalBayes` function, or sometimes for model estimation in its own right.

Usage

```
LaplaceApproximation(Model, parm, Data, Interval=1.0E-6,
     Iterations=100, Method="SPG", Samples=1000, CovEst="Hessian",
     sir=TRUE, Stop.Tolerance=1.0E-5, CPUs=1, Type="PSOCK")
```

Arguments

`Model` This required argument receives the model from a user-defined function. The user-defined function is where the model is specified. `LaplaceApproximation` passes two arguments to the model function, `parms` and `Data`. For more information, see the `LaplacesDemon` function and "LaplacesDemon Tutorial" vignette.

`parm` This argument requires a vector of initial values equal in length to the number of parameters. `LaplaceApproximation` will attempt to optimize these initial values for the parameters, where the optimized values are the posterior modes, for later use with the `IterativeQuadrature`, `LaplacesDemon`, `PMC`, or the `VariationalBayes` function. The `GIV` function may be used to randomly generate initial values. Parameters must be continuous.

`Data` This required argument accepts a list of data. The list of data must include `mon.names` which contains monitored variable names, and `parm.names` which contains parameter names. `LaplaceApproximation` must be able to determine the sample size of the data, and will look for a scalar sample size variable `n` or `N`. If not found, it will look for variable `y` or `Y`, and attempt to take its number of rows as sample size. `LaplaceApproximation` needs to determine sample size due to the asymptotic nature of this method. Sample size should be at least $\sqrt{J}$ with J exchangeable parameters.

`Interval` This argument receives an interval for estimating approximate gradients. The logarithm of the unnormalized joint posterior density of the Bayesian model is evaluated at the current parameter value, and again at the current parameter value plus this interval.

`Iterations` This argument accepts an integer that determines the number of iterations that `LaplaceApproximation` will attempt to maximize the logarithm of the unnormalized joint posterior density. `Iterations` defaults to 100. `LaplaceApproximation` will stop before this number of iterations if the tolerance is less than or equal to the `Stop.Tolerance` criterion. The required amount of computer memory increases with `Iterations`. If computer memory is exceeded, then all will be lost.

`Method` This optional argument accepts a quoted string that specifies the method used for Laplace Approximation. The default method is `Method="SPG"`. Options include `"AGA"` for adaptive gradient ascent, `"BFGS"` for the Broyden-Fletcher-Goldfarb-Shanno algorithm, `"BHHH"` for the algorithm of Berndt et al., `"CG"` for conjugate gradient, `"DFP"` for the Davidon-Fletcher-Powell algorithm, `"HAR"` for adaptive hit-and-run, `"HJ"` for Hooke-Jeeves, `"LBFGS"` for limited-memory BFGS, `"LM"` for Levenberg-Marquardt, `"NM"` for Nelder-Mead, `"NR"` for Newton-Raphson, `"PSO"` for Particle Swarm Optimization, `"Rprop"` for resilient backpropagation, `"SGD"` for Stochastic Gradient Descent, `"SOMA"` for the Self-Organizing Migration Algorithm, `"SPG"` for Spectral Projected Gradient, `"SR1"` for Symmetric Rank-One, and `"TR"` for Trust Region.

`Samples` This argument indicates the number of posterior samples to be taken with sampling importance resampling via the `SIR` function, which occurs only when `sir=TRUE`. Note that the number of samples should increase with the number and intercorrelations of the parameters.

`CovEst` This argument accepts a quoted string that indicates how the covariance matrix is estimated after the model finishes. This covariance matrix is used to obtain the standard deviation of each parameter, and may also be used for posterior sampling via Sampling Importance Resampling (SIR) (see the `sir` argument below), if converged. By default, the covariance matrix is approximated as the negative inverse of the `"Hessian"` matrix of second derivatives, estimated with Richardson extrapolation. Alternatives include `CovEst="Identity"`, `CovEst="OPG"`, or `CovEst="Sandwich"`. When `CovEst="Identity"`, the covariance matrix is not estimated, and is merely assigned an identity matrix. When `LaplaceApproximation` is performed internally by `LaplacesDemon`, an identity matrix is returned and scaled. When `CovEst="OPG"`, the covariance matrix is approximated with the inverse of the sum of the outer products of the gradient, which requires `X`, and either `y` or `Y` in the list of data. For OPG, a partial derivative is taken for each row in `X`, and each element in `y` or row in `Y`. Therefore, this requires $N + NJ$ model evaluations for a data set with N records and J variables. The OPG method is an asymptotic approximation of the Hessian, and usually requires fewer calculations with a small data set, or more with large data sets. Both methods require a matrix inversion, which becomes costly as dimension grows. The Richardson-based Hessian method is more accurate, but requires more calculation in large dimensions. An alternative approach to obtaining covariance is to use the `BayesianBootstrap` on the data, or sample the posterior with iterative quadrature (`IterativeQuadrature`), MCMC (`LaplacesDemon`), or `VariationalBayes`.

`sir` This logical argument indicates whether or not Sampling Importance Resampling (SIR) is conducted via the `SIR` function to draw independent posterior samples. This argument defaults to `TRUE`. Even when `TRUE`, posterior samples are drawn only when `LaplaceApproximation` has converged. Posterior samples are required for many other functions, including `plot.laplace` and `predict.laplace`. The only time that it is advantageous for `sir=FALSE` is when `LaplaceApproximation` is used to help the initial values for `IterativeQuadrature`, `LaplacesDemon`, `PMC`, or `VariationalBayes`, and it is unnecessary for time to be spent on sampling. Less time can be spent on sampling by increasing `CPUs`, which parallelizes the sampling.

`Stop.Tolerance` This argument accepts any positive number and defaults to 1.0E-5. Tolerance is calculated each iteration, and the criteria varies by algorithm. The algorithm is considered to have converged to the user-specified `Stop.Tolerance` when the tolerance is less than or equal to the value of `Stop.Tolerance`, and the algorithm terminates at the end of the current iteration. Often, multiple criteria are used, in which case the maximum of all criteria becomes the tolerance. For example, when partial derivatives are taken, it is commonly required that the Euclidean norm of the partial derivatives is a criterion, and another common criterion is the Euclidean norm of the differences between the current and previous parameter values. Several algorithms have other, specific tolerances.

`CPUs` This argument accepts an integer that specifies the number of central processing

units (CPUs) of the multicore computer or computer cluster. This argument defaults to `CPUs=1`, in which parallel processing does not occur. Parallelization occurs only for sampling with `SIR` when `sir=TRUE`.

`Type` This argument specifies the type of parallel processing to perform, accepting either `Type="PSOCK"` or `Type="MPI"`.

Details

The Laplace Approximation or Laplace Method is a family of asymptotic techniques used to approximate integrals. Laplace's method accurately approximates unimodal posterior moments and marginal posterior distributions in many cases. Since it is not applicable in all cases, it is recommended here that Laplace Approximation is used cautiously in its own right, or preferably, it is used before MCMC.

After introducing the Laplace Approximation (Laplace, 1774, p. 366–367), a proof was published later (Laplace, 1814) as part of a mathematical system of inductive reasoning based on probability. Laplace used this method to approximate posterior moments.

Since its introduction, the Laplace Approximation has been applied successfully in many disciplines. In the 1980s, the Laplace Approximation experienced renewed interest, especially in statistics, and some improvements in its implementation were introduced (Tierney et al., 1986; Tierney et al., 1989). Only since the 1980s has the Laplace Approximation been seriously considered by statisticians in practical applications.

There are many variations of Laplace Approximation, with an effort toward replacing Markov chain Monte Carlo (MCMC) algorithms as the dominant form of numerical approximation in Bayesian inference. The run-time of Laplace Approximation is a little longer than Maximum Likelihood Estimation (MLE), usually shorter than variational Bayes, and much shorter than MCMC (Azevedo and Shachter, 1994).

The speed of Laplace Approximation depends on the optimization algorithm selected, and typically involves many evaluations of the objective function per iteration (where an MCMC algorithm with a multivariate proposal usually evaluates once per iteration), making many MCMC algorithms faster per iteration. The attractiveness of Laplace Approximation is that it typically improves the objective function better than iterative quadrature, MCMC, and PMC when the parameters are in low-probability regions. Laplace Approximation is also typically faster than MCMC and PMC because it is seeking point-estimates, rather than attempting to represent the target distribution with enough simulation draws. Laplace Approximation extends MLE, but shares similar limitations, such as its asymptotic nature with respect to sample size and that marginal posterior distributions are Gaussian. Bernardo and Smith (2000) note that Laplace Approximation is an attractive family of numerical approximation algorithms, and will continue to develop.

`LaplaceApproximation` seeks a global maximum of the logarithm of the unnormalized joint posterior density. The approach differs by `Method`. The `LaplacesDemon` function uses the `LaplaceApproximation` algorithm to optimize initial values and save time for the user.

Most optimization algorithms assume that the logarithm of the unnormalized joint posterior density is defined and differentiable. Some methods calculate an approximate gradient for each initial value as the difference in the logarithm of the unnormalized joint posterior density due to a slight increase in the parameter.

When `Method="AGA"`, the direction and distance for each parameter is proposed based on an approximate truncated gradient and an adaptive step size. The step size parameter, which is often

plural and called rate parameters in other literature, is adapted each iteration with the univariate version of the Robbins-Monro stochastic approximation in Garthwaite (2010). The step size shrinks when a proposal is rejected and expands when a proposal is accepted.

Gradient ascent is criticized for sometimes being relatively slow when close to the maximum, and its asymptotic rate of convergence is inferior to other methods. However, compared to other popular optimization algorithms such as Newton-Raphson, an advantage of the gradient ascent is that it works in infinite dimensions, requiring only sufficient computer memory. Although Newton-Raphson converges in fewer iterations, calculating the inverse of the negative Hessian matrix of second-derivatives is more computationally expensive and subject to singularities. Therefore, gradient ascent takes longer to converge, but is more generalizable.

When `Method="BFGS"`, the BFGS algorithm is used, which was proposed by Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970), independently. BFGS may be the most efficient and popular quasi-Newton optimiziation algorithm. As a quasi-Newton algorithm, the Hessian matrix is approximated using rank-one updates specified by (approximate) gradient evaluations. Since BFGS is very popular, there are many variations of it. This is a version by Nash that has been adapted from the Rvmmin package, and is used in the `optim` function of base R. The approximate Hessian is not guaranteed to converge to the Hessian. When BFGS is used, the approximate Hessian is not used to calculate the final covariance matrix.

When `Method="BHHH"`, the algorithm of Berndt et al. (1974) is used, which is commonly pronounced B-triple H. The BHHH algorithm is a quasi-Newton method that includes a step-size parameter, partial derivatives, and an approximation of a covariance matrix that is calculated as the inverse of the sum of the outer product of the gradient (OPG), calculated from each record. The OPG method becomes more costly with data sets with more records. Since partial derivatives must be calculated per record of data, the list of data has special requirements with this method, and must include design matrix `X`, and dependent variable `y` or `Y`. Records must be row-wise. An advantage of BHHH over NR (see below) is that the covariance matrix is necessarily positive definite, and gauranteed to provide an increase in LP each iteration (given a small enough step-size), even in convex areas. The covariance matrix is better approximated with larger data sample sizes, and when closer to the maximum of LP. Disadvantages of BHHH include that it can give small increases in LP, especially when far from the maximum or when LP is highly non-quadratic.

When `Method="CG"`, a nonlinear conjugate gradient algorithm is used. CG uses partial derivatives, but does not use the Hessian matrix or any approximation of it. CG usually requires more iterations to reach convergence than other algorithms that use the Hessian or an approximation. However, since the Hessian becomes computationally expensive as the dimension of the model grows, CG is applicable to large dimensional models when `CovEst="Hessian"` is avoided. CG was originally developed by Hestenes and Stiefel (1952), though this version is adapted from the `Rcgminu` function in package Rcgmin.

When `Method="DFP"`, the Davidon-Fletcher-Powell algorithm is used. DFP was the first popular, multidimensional, quasi-Newton optimization algorithm. The DFP update of an approximate Hessian matrix maintains symmetry and positive-definiteness. The approximate Hessian is not guaranteed to converge to the Hessian. When DFP is used, the approximate Hessian is not used to calculate the final covariance matrix. Although DFP is very effective, it was superseded by the BFGS algorithm.

When `Method="HAR"`, a hit-and-run algorithm with a multivariate proposal and adaptive length is used. The length parameter is adapted each iteration with the univariate version of the Robbins-Monro stochastic approximation in Garthwaite (2010). The length shrinks when a proposal is rejected and expands when a proposal is accepted. This is the same algorithm as the HARM or

Hit-And-Run Metropolis MCMC algorithm with adaptive length, except that a Metropolis step is not used.

When `Method="HJ"`, the Hooke-Jeeves (1961) algorithm is used. This was adapted from the `HJK` algorithm in package dfoptim. Hooke-Jeeves is a derivative-free, direct search method. Each iteration involves two steps: an exploratory move and a pattern move. The exploratory move explores local behavior, and the pattern move takes advantage of pattern direction. It is sometimes described as a hill-climbing algorithm. If the solution improves, it accepts the move, and otherwise rejects it. Step size decreases with each iteration. The decreasing step size can trap it in local maxima, where it gets stuck and convergences erroneously. Users are encouraged to attempt again after what seems to be convergence, starting from the latest point. Although getting stuck at local maxima can be problematic, the Hooke-Jeeves algorithm is also attractive because it is simple, fast, does not depend on derivatives, and is otherwise relatively robust.

When `Method="LBFGS"`, the limited-memory BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is called in `optim`, once per iteration.

When `Method="LM"`, the Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963) is used. Also known as the Levenberg-Marquardt Algorithm (LMA) or the Damped Least-Squares (DLS) method, LM is a trust region (not to be confused with TR below) quasi-Newton optimization algorithm that provides minimizes nonlinear least squares, and has been adapted here to maximize LP. LM uses partial derivatives and approximates the Hessian with outer-products. It is suitable for nonlinear optimization up to a few hundred parameters, but loses its efficiency in larger problems due to matrix inversion. LM is considered between the Gauss-Newton algorithm and gradient descent. When far from the solution, LM moves slowly like gradient descent, but is guaranteed to converge. When LM is close to the solution, LM becomes a damped Gauss-Newton method. This was adapted from the `lsqnonlin` algorithm in package pracma.

When `Method="NM"`, the Nelder-Mead (1965) algorithm is used. This was adapted from the `nelder_mead` function in package pracma. Nelder-Mead is a derivative-free, direct search method that is known to become inefficient in large-dimensional problems. As the dimension increases, the search direction becomes increasingly orthogonal to the steepest ascent (usually descent) direction. However, in smaller dimensions, it is a popular algorithm. At each iteration, three steps are taken to improve a simplex: reflection, extension, and contraction.

When `Method="NR"`, the Newton-Raphson optimization algorithm, also known as Newton's Method, is used. Newton-Raphson uses derivatives and a Hessian matrix. The algorithm is included for its historical significance, but is known to be problematic when starting values are far from the targets, and calculating and inverting the Hessian matrix can be computationally expensive. As programmed here, when the Hessian is problematic, it tries to use only the derivatives, and when that fails, a jitter is applied. Newton-Raphson should not be the first choice of the user, and BFGS should always be preferred.

When `Method="PSO"`, the Standard Particle Swarm Optimization 2007 algorithm is used. A swarm of particles is moved according to velocity, neighborhood, and the best previous solution. The neighborhood for each particle is a set of informing particles. PSO is derivative-free. PSO has been adapted from the `psoptim` function in package pso.

When `Method="Rprop"`, the approximate gradient is taken for each parameter in each iteration, and its sign is compared to the approximate gradient in the previous iteration. A weight element in a weight vector is associated with each approximate gradient. A weight element is multiplied by 1.2 when the sign does not change, or by 0.5 if the sign changes. The weight vector is the step size, and is constrained to the interval [0.001, 50], and initial weights are 0.0125. This is the resilient backpropagation algorithm, which is often denoted as the "Rprop-" algorithm of Riedmiller (1994).

When Method="SGD", a stochastic gradient descent algorithm is used that is designed only for big data, and gained popularity after successful use in the NetFlix competition. This algorithm has special requirements for the Model specification function and the Data list. See the "LaplacesDemon Tutorial" vignette for more information.

When Method="SOMA", a population of ten particles or individuals moves in the direction of the best particle, the leader. The leader does not move in each iteration, and a line-search is used for each non-leader, up to three times the difference in parameter values between each non-leader and leader. This algorithm is derivative-free and often considered in the family of evolution algorithms. Numerous model evaluations are performed per non-leader per iteration. This algorithm was adapted from package soma.

When Method="SPG", a Spectral Projected Gradient algorithm is used. SPG is a non-monotone algorithm that is suitable for high-dimensional models. The approximate gradient is used, but the Hessian matrix is not. When used with large models, CovEst="Hessian" should be avoided. SPG has been adapted from the spg function in package BB.

When Method="SR1", the Symmetric Rank-One (SR1) algorithm is used. SR1 is a quasi-Newton algorithm, and the Hessian matrix is approximated, often without being positive-definite. At the posterior modes, the true Hessian is usually positive-definite, but this is often not the case during optimization when the parameters have not yet reached the posterior modes. Other restrictions, including constraints, often result in the true Hessian being indefinite at the solution. For these reasons, SR1 often outperforms BFGS. The approximate Hessian is not guaranteed to converge to the Hessian. When SR1 is used, the approximate Hessian is not used to calculate the final covariance matrix.

When Method="TR", the Trust Region algorithm of Nocedal and Wright (1999) is used. The TR algorithm attempts to reach its objective in the fewest number of iterations, is therefore very efficient, as well as safe. The efficiency of TR is attractive when model evaluations are expensive. The Hessian is approximated each iteration, making TR best suited to models with small to medium dimensions, say up to a few hundred parameters. TR has been adapted from the trust function in package trust.

Value

LaplaceApproximation returns an object of class laplace that is a list with the following components:

Call	This is the matched call of LaplaceApproximation.
Converged	This is a logical indicator of whether or not LaplaceApproximation converged within the specified Iterations according to the supplied Stop.Tolerance criterion. Convergence does not indicate that the global maximum has been found, but only that the tolerance was less than or equal to the Stop.Tolerance criterion.
Covar	This covariance matrix is estimated according to the CovEst argument. The Covar matrix may be scaled and input into the Covar argument of the LaplacesDemon or PMC function for further estimation, or the diagonal of this matrix may be used to represent the posterior variance of the parameters, provided the algorithm converged and matrix inversion was successful. To scale this matrix for use with Laplace's Demon or PMC, multiply it by $2.38^2/d$, where d is the number of initial values.

`Deviance` This is a vector of the iterative history of the deviance in the `LaplaceApproximation` function, as it sought convergence.

`History` This is a matrix of the iterative history of the parameters in the `LaplaceApproximation` function, as it sought convergence.

`Initial.Values` This is the vector of initial values that was originally given to `LaplaceApproximation` in the `parm` argument.

`LML` This is an approximation of the logarithm of the marginal likelihood of the data (see the `LML` function for more information). When the model has converged and `sir=TRUE`, the NSIS method is used. When the model has converged and `sir=FALSE`, the LME method is used. This is the logarithmic form of equation 4 in Lewis and Raftery (1997). As a rough estimate of Kass and Raftery (1995), the LME-based LML is worrisome when the sample size of the data is less than five times the number of parameters, and `LML` should be adequate in most problems when the sample size of the data exceeds twenty times the number of parameters (p. 778). The LME is inappropriate with hierarchical models. However `LML` is estimated, it is useful for comparing multiple models with the `BayesFactor` function.

`LP.Final` This reports the final scalar value for the logarithm of the unnormalized joint posterior density.

`LP.Initial` This reports the initial scalar value for the logarithm of the unnormalized joint posterior density.

`Minutes` This is the number of minutes that `LaplaceApproximation` was running, and this includes the initial checks as well as drawing posterior samples and creating summaries.

`Monitor` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the monitored variables.

`Posterior` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the parameters.

`Step.Size.Final`
This is the final, scalar `Step.Size` value at the end of the `LaplaceApproximation` algorithm.

`Step.Size.Initial`
This is the initial, scalar `Step.Size`.

`Summary1` This is a summary matrix that summarizes the point-estimated posterior modes. Uncertainty around the posterior modes is estimated from the covariance matrix. Rows are parameters. The following columns are included: Mode, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Summary2` This is a summary matrix that summarizes the posterior samples drawn with sampling importance resampling (`SIR`) when `sir=TRUE`, given the point-estimated posterior modes and the covariance matrix. Rows are parameters. The following columns are included: Mode, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Tolerance.Final`
: This is the last `Tolerance` of the `LaplaceApproximation` algorithm.

`Tolerance.Stop`
: This is the `Stop.Tolerance` criterion.

Author(s)

Statisticat, LLC `<software@bayesian-inference.com>`

References

Azevedo-Filho, A. and Shachter, R. (1994). "Laplace's Method Approximations for Probabilistic Inference in Belief Networks with Continuous Variables". In "Uncertainty in Artificial Intelligence", Mantaras, R. and Poole, D., Morgan Kauffman, San Francisco, CA, p. 28–36.

Bernardo, J.M. and Smith, A.F.M. (2000). "Bayesian Theory". John Wiley \& Sons: West Sussex, England.

Berndt, E., Hall, B., Hall, R., and Hausman, J. (1974), "Estimation and Inference in Nonlinear Structural Models". *Annals of Economic and Social Measurement*, 3, p. 653–665.

Broyden, C.G. (1970). "The Convergence of a Class of Double Rank Minimization Algorithms: 2. The New Algorithm". Journal of the Institute of Mathematics and its Applications, 6, p.76–90.

Fletcher, R. (1970). "A New Approach to Variable Metric Algorithms". Computer Journal, 13(3), p. 317–322.

Garthwaite, P., Fan, Y., and Sisson, S. (2010). "Adaptive Optimal Scaling of Metropolis-Hastings Algorithms Using the Robbins-Monro Process."

Goldfarb, D. (1970). "A Family of Variable Metric Methods Derived by Variational Means". Mathematics of Computation, 24(109), p. 23–26.

Hestenes, M.R. and Stiefel, E. (1952). "Methods of Conjugate Gradients for Solving Linear Systems". *Journal of Research of the National Bureau of Standards*, 49(6), p. 409–436.

Hooke, R. and Jeeves, T.A. (1961). "'Direct Search' Solution of Numerical and Statistical Problems". *Journal of the Association for Computing Machinery*, 8(2), p. 212–229.

Kass, R.E. and Raftery, A.E. (1995). "Bayes Factors". *Journal of the American Statistical Association*, 90(430), p. 773–795.

Laplace, P. (1774). "Memoire sur la Probabilite des Causes par les Evenements." l'Academie Royale des Sciences, 6, 621–656. English translation by S.M. Stigler in 1986 as "Memoir on the Probability of the Causes of Events" in Statistical Science, 1(3), 359–378.

Laplace, P. (1814). "Essai Philosophique sur les Probabilites." English translation in Truscott, F.W. and Emory, F.L. (2007) from (1902) as "A Philosophical Essay on Probabilities". ISBN 1602063281, translated from the French 6th ed. (1840).

Levenberg, K. (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". *Quarterly of Applied Mathematics*, 2, p. 164–168.

Lewis, S.M. and Raftery, A.E. (1997). "Estimating Bayes Factors via Posterior Simulation with the Laplace-Metropolis Estimator". *Journal of the American Statistical Association*, 92, p. 648–655.

Marquardt, D. (1963). "An Algorithm for Least-Squares Estimation of Nonlinear Parameters". *SIAM Journal on Applied Mathematics*, 11(2), p. 431–441.

Nelder, J.A. and Mead, R. (1965). "A Simplex Method for Function Minimization". *The Computer Journal*, 7(4), p. 308–313.

Nocedal, J. and Wright, S.J. (1999). "Numerical Optimization". Springer-Verlag.

Riedmiller, M. (1994). "Advanced Supervised Learning in Multi-Layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms". *Computer Standards and Interfaces*, 16, p. 265–278.

Shanno, D.F. (1970). "Conditioning of quasi-Newton Methods for Function Minimization". Mathematics of Computation, 24(111), p. 647–650.

Tierney, L. and Kadane, J.B. (1986). "Accurate Approximations for Posterior Moments and Marginal Densities". *Journal of the American Statistical Association*, 81(393), p. 82–86.

Tierney, L., Kass. R., and Kadane, J.B. (1989). "Fully Exponential Laplace Approximations to Expectations and Variances of Nonpositive Functions". *Journal of the American Statistical Association*, 84(407), p. 710–716.

Zelinka, I. (2004). "SOMA - Self Organizing Migrating Algorithm". In: Onwubolu G.C. and Babu, B.V., editors. "New Optimization Techniques in Engineering". Springer: Berlin, Germany.

See Also

BayesFactor, BayesianBootstrap, IterativeQuadrature, LaplacesDemon, GIV, LML, optim, PMC, SIR, and VariationalBayes.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#####################  Load the LaplacesDemon Library  #####################
library(LaplacesDemon)

#############################  Demon Data  #############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,10]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

#########################  Data List Preparation  #########################
mon.names <- "mu[1]"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

#########################  Model Specification  #########################
Model <- function(parm, Data)
```

```
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=mu[1],
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

############################  Initial Values  ############################
#Initial.Values <- GIV(Model, MyData, PGF=TRUE)
Initial.Values <- rep(0,J+1)

Fit <- LaplaceApproximation(Model, Initial.Values, Data=MyData,
     Iterations=100, Method="NM", CPUs=1)
Fit
print(Fit)
#PosteriorChecks(Fit)
#caterpillar.plot(Fit, Parms="beta")
#plot(Fit, MyData, PDF=FALSE)
#Pred <- predict(Fit, Model, MyData, CPUs=1)
#summary(Pred, Discrep="Chi-Square")
#plot(Pred, Style="Covariates", Data=MyData)
#plot(Pred, Style="Density", Rows=1:9)
#plot(Pred, Style="Fitted")
#plot(Pred, Style="Jarque-Bera")
#plot(Pred, Style="Predictive Quantiles")
#plot(Pred, Style="Residual Density")
#plot(Pred, Style="Residuals")
#Levene.Test(Pred)
#Importance(Fit, Model, MyData, Discrep="Chi-Square")

#Fit$Covar is scaled (2.38^2/d) and submitted to LaplacesDemon as Covar.
#Fit$Summary[,1] is submitted to LaplacesDemon as Initial.Values.
#End
```

LaplacesDemon *Laplace's Demon*

Description

The LaplacesDemon function is the main function of Laplace's Demon. Given data, a model specification, and initial values, LaplacesDemon maximizes the logarithm of the unnormalized joint posterior density with MCMC and provides samples of the marginal posterior distributions, deviance, and other monitored variables.

The LaplacesDemon.hpc function extends LaplacesDemon to parallel chains for multicore or cluster high performance computing.

Usage

```
LaplacesDemon(Model, Data, Initial.Values, Covar=NULL, Iterations=10000,
     Status=100, Thinning=10, Algorithm="MWG", Specs=list(B=NULL),
     Debug=list(DB.chol=FALSE, DB.eigen=FALSE, DB.MCSE=FALSE,
     DB.Model=TRUE), LogFile="", ...)
LaplacesDemon.hpc(Model, Data, Initial.Values, Covar=NULL,
     Iterations=10000, Status=100, Thinning=10, Algorithm="MWG",
     Specs=list(B=NULL), Debug=list(DB.chol=FALSE, DB.eigen=FALSE,
     DB.MCSE=FALSE, DB.Model=TRUE), LogFile="", Chains=2, CPUs=2,
     Type="PSOCK", Packages=NULL, Dyn.libs=NULL)
```

Arguments

Model	This required argument receives the model from a user-defined function that must be named Model. The user-defined function is where the model is specified. LaplacesDemon passes two arguments to the model function, parms and Data, and receives five arguments from the model function: LP (the logarithm of the unnormalized joint posterior), Dev (the deviance), Monitor (the monitored variables), yhat (the variables for posterior predictive checks), and parm, the vector of parameters, which may be constrained in the model function. More information on the Model specification function may be found in the "LaplacesDemon Tutorial" vignette, and the is.model function. Many examples of model specification functions may be found in the "Examples" vignette.
Data	This required argument accepts a list of data. The list of data must contain mon.names which contains monitored variable names, and must contain parm.names which contains parameter names. The as.parm.names function may be helpful for preparing the data, and the is.data function may be helpful for checking data.
Initial.Values	For LaplacesDemon, this argument requires a vector of initial values equal in length to the number of parameters. For LaplacesDemon.hpc, this argument also accepts a vector, in which case the same initial values will be applied to all parallel chains, or the argument accepts a matrix in which each row is a parallel chain and the number of columns is equal in length to the number of parameters. When a matrix is supplied for LaplacesDemon.hpc, each parallel chain begins with its own initial values that are preferably dispersed. For both LaplacesDemon and LaplacesDemon.hpc, each initial value will be the starting point for an adaptive chain or a non-adaptive Markov chain of a parameter. Parameters are assumed to be continuous, unless specified to be discrete (see

dparm below), which is not accepted by all algorithms (see dcrmrf for an alternative). If all initial values are set to zero, then Laplace's Demon will attempt to optimize the initial values with the LaplaceApproximation function. After Laplace's Demon finishes updating, it may be desired to continue updating from where it left off. To continue, this argument should receive the last iteration of the previous update. For example, if the output object is called Fit, then Initial.Values=as.initial.values(Fit). Initial values may be generated randomly with the GIV function.

Covar — This argument defaults to NULL, but may otherwise accept a $K \times K$ proposal covariance matrix (where K is the number of dimensions or parameters), a variance vector, or a list of covariance matrices (for blockwise sampling in some algorithms). When the model is updated for the first time and prior variance or covariance is unknown, then Covar=NULL should be used. Some algorithms require covariance, some only require variance, and some require neither. Laplace's Demon automatically converts the user input to the required form. Once Laplace's Demon has finished updating, it may be desired to continue updating where it left off, in which case the proposal covariance matrix from the last run can be input into the next run. The covariance matrix may also be input from the LaplaceApproximation function, if used.

Iterations — This required argument accepts integers larger than 10, and determines the number of iterations that Laplace's Demon will update the parameters while searching for target distributions. The required amount of computer memory will increase with Iterations. If computer memory is exceeded, then all will be lost. The Combine function can be used later to combine multiple updates.

Status — This argument accepts an integer between 1 and the number of iterations, and indicates how often, in iterations, the user would like the status printed to the screen or log file. Usually, the following is reported: the number of iterations, the proposal type (for example, multivariate or componentwise, or mixture, or subset), and LP. For example, if a model is updated for 1,000 iterations and Status=200, then a status message will be printed at the following iterations: 200, 400, 600, 800, and 1,000.

Thinning — This argument accepts integers between 1 and the number of iterations, and indicates that every nth iteration will be retained, while the other iterations are discarded. If Thinning=5, then every 5th iteration will be retained. Thinning is performed to reduce autocorrelation and the number of marginal posterior samples.

Algorithm — This argument accepts the abbreviated name of the MCMC algorithm, which must appear in quotes. A list of MCMC algorithms appears below in the Details section, and the abbreviated name is in parenthesis.

Specs — This argument defaults to NULL, and accepts a list of specifications for the MCMC algorithm declared in the Algorithm argument. The specifications associated with each algorithm may be seen below in the examples, must appear in the order shown, and are described in the details section below.

Debug — This argument accepts a list of logical scalars that control whether or not errors or warnings are reported due to a try function or non-finite values. List components include DB.chol regarding chol, DB.eigen regarding eigen, DB.MCSE

regarding MCSE, and DB.Model regarding the Model specification function. Errors and warnings should be investigated, but do not necessarily indicate a faulty Model specification function or a bug in the software. For example, a sampler may make a proposal that would result in a matrix that is not positive definite, when it should be. This kind of error or warning is acceptable, provided the sampler handles it correctly by rejecting the proposal, and provided the Model specification function is not causing the issue. Oftentimes, blockwise sampling with carefully chosen blocks will mostly or completely eliminate errors or warnings that occur otherwise in larger, multivariate proposals. Similarly, debugged componentwise algorithms tend to provide more information than multivariate algorithms, since usually the parameter and both its current and proposed values may be reported. If confident in the Model specification function, and errors or warnings are produced frequently that are acceptable, then consider setting DB.Model=FALSE for cleaner output and faster sampling. If the Model specification function is not faulty and there is a bug in LaplacesDemon, then please report it with a bug description and reproducible code on https://github.com/LaplacesDemonR/LaplacesDemon/issues.

LogFile — This argument is used to specify a log file name in quotes in the working directory as a destination, rather than the console, for the output messages of cat and stop commands. It is helpful to assign a log file name when using multiple cores, such as with LaplacesDemon.hpc. Doing so allows the user to check the progress in the log. A number of log files are created, one for each chain, and one for the overall process.

Chains — This argument is required only for LaplacesDemon.hpc, and indicates the number of parallel chains.

CPUs — This argument is required for parallel independent or interactive chains in LaplacesDemon or LaplacesDemon.hpc, and indicates the number of central processing units (CPUs) of the computer or cluster. For example, when a user has a quad-core computer, CPUs=4.

Type — This argument defaults to "PSOCK" and uses the Simple Network of Workstations (SNOW) for parallelization. Alternatively, Type="MPI" may be specified to use Message Passing Interface (MPI) for parallelization.

Packages — This optional argument is for use with parallel independent or interacting chains, and defaults to NULL. This argument accepts a vector of package names to load into each parallel chain. If the Model specification depends on any packages, then these package names need to be in this vector.

Dyn.libs — This optional argument is for use with parallel independent or interacting chain, and defaults to NULL. This argument accepts a vector of the names of dynamic link libraries (shared objects) to load into each parallel chain. The libraries must be located in the working directory.

... — Additional arguments are unused.

Details

LaplacesDemon offers numerous MCMC algorithms for numerical approximation in Bayesian inference. The algorithms are

- Adaptive Directional Metropolis-within-Gibbs (ADMG)
- Adaptive Griddy-Gibbs (AGG)
- Adaptive Hamiltonian Monte Carlo (AHMC)
- Adaptive Metropolis (AM)
- Adaptive Metropolis-within-Gibbs (AMWG)
- Adaptive-Mixture Metropolis (AMM)
- Affine-Invariant Ensemble Sampler (AIES)
- Componentwise Hit-And-Run Metropolis (CHARM)
- Delayed Rejection Adaptive Metropolis (DRAM)
- Delayed Rejection Metropolis (DRM)
- Differential Evolution Markov Chain (DEMC)
- Elliptical Slice Sampler (ESS)
- Gibbs Sampler (Gibbs)
- Griddy-Gibbs (GG)
- Hamiltonian Monte Carlo (HMC)
- Hamiltonian Monte Carlo with Dual-Averaging (HMCDA)
- Hit-And-Run Metropolis (HARM)
- Independence Metropolis (IM)
- Interchain Adaptation (INCA)
- Metropolis-Adjusted Langevin Algorithm (MALA)
- Metropolis-Coupled Markov Chain Monte Carlo (MCMCMC)
- Metropolis-within-Gibbs (MWG)
- Multiple-Try Metropolis (MTM)
- No-U-Turn Sampler (NUTS)
- Oblique Hyperrectangle Slice Sampler (OHSS)
- Preconditioned Crank-Nicolson (pCN)
- Random Dive Metropolis-Hastings (RDMH)
- Random-Walk Metropolis (RWM)
- Reflective Slice Sampler (RSS)
- Refractive Sampler (Refractive)
- Reversible-Jump (RJ)
- Robust Adaptive Metropolis (RAM)
- Sequential Adaptive Metropolis-within-Gibbs (SAMWG)
- Sequential Metropolis-within-Gibbs (SMWG)
- Slice Sampler (Slice)
- Stochastic Gradient Langevin Dynamics (SGLD)
- Tempered Hamiltonian Monte Carlo (THMC)

- t-walk (twalk)
- Univariate Eigenvector Slice Sampler (UESS)
- Updating Sequential Adaptive Metropolis-within-Gibbs (USAMWG)
- Updating Sequential Metropolis-within-Gibbs (USMWG)

It is a goal for the documentation in the **LaplacesDemon** to be extensive. However, details of MCMC algorithms are best explored online at https://web.archive.org/web/20150206014000/http://www.bayesian-inference.com/mcmc, as well as in the "LaplacesDemon Tutorial" vignette, and the "Bayesian Inference" vignette. Algorithm specifications (`Specs`) are listed below:

- `A` is used in AFSS, HMCDA, MALA, NUTS, OHSS, and UESS. In MALA, it is the maximum acceptable value of the Euclidean norm of the adaptive parameters mu and sigma, and the Frobenius norm of the covariance matrix. In AFSS, HMCDA, NUTS, OHSS, and UESS, it is the number of initial, adaptive iterations to be discarded as burn-in.
- `Adaptive` is the iteration in which adaptation begins, and is used in AM, AMM, DRAM, INCA, and Refractive. Most of these algorithms adapt according to an observed covariance matrix, and should sample before beginning to adapt.
- `alpha.star` is the target acceptance rate in MALA and RAM, and is optional in CHARM and HARM. The recommended value for multivariate proposals is `alpha.star=0.234`, for componentwise proposals is `alpha.star=0.44`, and for MALA is `alpha.star=0.574`.
- `at` affects the traverse move in twalk. `at=6` is recommended. It helps when some parameters are highly correlated, and the correlation structure may change through the state-space. The traverse move is associated with an acceptance rate that decreases as the number of parameters increases, and is the reason that `n1` is used to select a subset of parameters each iteration. If adjusted, it is recommended to stay in the interval [2,10].
- `aw` affects the walk move in twalk, and `aw=1.5` is recommended. If adjusted, it is recommended to stay in the interval [0.3,2].
- `beta` is a scale parameter for AIES, and defaults to 2, or an autoregressive parameter for pCN.
- `bin.n` is the scalar size parameter for a binomial prior distribution of model size for the RJ algorithm.
- `bin.p` is the scalar probability parameter for a binomial prior distribution of model size for the RJ algorithm.
- `B` is a list of blocked parameters. Each component of the list represents a block of parameters, and contains a vector in which each element is the position of the associated parameter in parm.names. This function is optional in the AFSS, AMM, AMWG, ESS, HARM, MWG, RAM, RWM, Slice, and UESS algorithms. For more information on blockwise sampling, see the `Blocks` function.
- `Begin` indicates the time-period in which to begin updating (filtering or predicting) in the USAMWG and USMWG algorithms.
- `Bounds` is used in the Slice algorithm. It is a vector of length two with the lower and upper boundary of the slice. For continuous parameters, it is often set to negative and positive infinity, while for discrete parameters it is set to the minimum and maximum discrete values to be sampled. When blocks are used, this must be supplied as a list with the same number of list components as the number of blocks.

- delta is used in HMCDA, MALA, and NUTS. In HMCDA and NUTS, it is the target acceptance rate, and the recommended value is 0.65 in HMCDA and 0.6 in NUTS. In MALA, it is a constant in the bounded drift function, may be in the interval [1e-10,1000], and 1 is the default.
- Dist is the proposal distribution in RAM, and may either be Dist="t" for t-distributed or Dist="N" for normally-distributed.
- dparm accepts a vector of integers that indicate discrete parameters. This argument is for use with the AGG or GG algorithm.
- Dyn is a $T \times K$ matrix of dynamic parameters, where T is the number of time-periods and K is the number of dynamic parameters. Dyn is used by SAMWG, SMWG, USAMWG, and USMWG. Non-dynamic parameters are updated first in each sampler iteration, then dynamic parameters are updated in a random order in each time-period, and sequentially by time-period.
- epsilon is used in AHMC, HMC, HMCDA, MALA, NUTS, SGLD, and THMC. It is the step-size in all algorithms except MALA. It is a vector equal in length to the number of parameters in AHMC, HMC, and THMC. It is a scalar in HMCDA and NUTS. It is either a scalar or a vector equal in length to the number of iterations in SGLD. When epsilon=NULL in HMCDA or NUTS (only), a reasonable initial value is found. In MALA, it is a vector of length two. The first element is the acceptable minimum of adaptive scale sigma, and the second element is added to the diagonal of the covariance matrix for regularization.
- FC is used in Gibbs and accepts a function that receives two arguments: the vector of all parameters and the list of data (similar to the Model specification function). FC must return the updated vector of all parameters. The user specifies FC to calculate the full conditional distribution of one or more parameters.
- file is the quoted name of a numeric matrix of data, without headers, for SGLD. The big data set must be a .csv file. This matrix has Nr rows and Nc columns. Each iteration, SGLD will randomly select a block of rows, where the number of rows is specified by the size argument.
- Fit is an object of class demonoid in the USAMWG and USMWG algorithms. Posterior samples before the time-period specified in the Begin argument are not updated, and are used instead from Fit.
- gamma controls the step size in DEMC or the decay of adaptation in MALA and RAM. In DEMC, it is positive and defaults to $2.38/\sqrt{2J}$ when NULL, where J is the length of initial values. For RAM, it is in the interval (0.5,1], and 0.66 is recommended. For MALA, it is in the interval (1,Iterations), and defaults to 1.
- Grid accepts either a vector or a list of vectors of evenly-spaced points on a grid for the AGG or GG algorithm. When the argument is a vector, the same grid is applied to all parameters. When the argument is a list, each component in the list has a grid that is applied to the corresponding parameter. The algorithm will evaluate each continuous parameter at the latest value plus each point in the grid, or each discrete parameter (see dparm) at each grid point (which should be each discrete value).
- K is a scalar number of proposals in MTM.
- L is a scalar number of leapfrog steps in AHMC, HMC, and THMC. When L=1, the algorithm reduces to Langevin Monte Carlo (LMC).
- lambda is used in HMCDA and MCMCMC. In HMCDA, it is a scalar trajectory length. In MCMCMC, it is either a scalar that controls temperature spacing, or a vector of temperature spacings.

- `Lmax` is a scalar maximum for `L` (see above) in HMCDA and NUTS.
- `m` is used in the AFSS, AHMC, HMC, Refractive, RSS, Slice, THMC, and UESS algorithms. In AHMC, HMC, and THMC, it is a $J \times J$ mass matrix for J initial values. In AFSS and UESS, it is a scalar, and is the maximum number of steps for creating the slice interval. In Refractive and RSS, it is a scalar, and is the number of steps to take per iteration. In Slice, it is either a scalar or a list with as many list components as blocks. It must be an integer in [1,Inf], and indicates the maximum number of steps for creating the slice interval.
- `mu` is a vector that is equal in length to the initial values. This vector will be used as the mean of the proposal distribution, and is usually the posterior mode of a previously-updated `LaplaceApproximation`.
- `MWG` is used in Gibbs to specify a vector of parameters that are to receive Metropolis-within-Gibbs updates. Each element is an integer that indicates the parameter.
- `Nc` is either the number of (un-parallelized) parallel chains in DEMC (and must be at least 3) or the number of columns of big data in SGLD.
- `Nr` is the number of rows of big data in SGLD.
- `n` is the number of previous iterations in ADMG, AFSS, AMM, AMWG, OHSS, RAM, and UESS.
- `n1` affects the size of the subset of each set of points to adjust, and is used in twalk. It relates to the number of parameters, and `n1=4` is recommended. If adjusted, it is recommended to stay in the interval [2,20].
- `parm.p` is a vector of probabilities for parameter selection in the RJ algorithm, and must be equal in length to the number of initial values.
- `r` is a scalar used in the Refractive algorithm to indicate the ratio between r1 and r2.
- `Periodicity` specifies how often in iterations the adaptive algorithm should adapt, and is used by AHMC, AM, AMM, AMWG, DRAM, INCA, SAMWG, and USAMWG. If `Periodicity=10`, then the algorithm adapts every 10th iteration. A higher `Periodicity` is associated with an algorithm that runs faster, because it does not have to calculate adaptation as often, though the algorithm adapts less often to the target distributions, so it is a trade-off. It is recommended to use the lowest value that runs fast enough to suit the user, or provide sufficient adaptation.
- `selectable` is a vector of indicators of whether or not a parameter is selectable for variable selection in the RJ algorithm. Non-selectable parameters are assigned a zero, and are always in the model. Selectable parameters are assigned a one. This vector must be equal in length to the number of initial values.
- `selected` is a vector of indicators of whether or not each parameter is selected when the RJ algorithm begins, and must be equal in length to the number of initial values.
- `SIV` stands for secondary initial values and is used by twalk. `SIV` must be the same length as `Initial.Values`, and each element of these two vectors must be unique from each other, both before and after being passed to the `Model` function. `SIV` defaults to `NULL`, in which case values are generated with `GIV`.
- `size` is the number of rows of big data to be read into SGLD each iteration.
- `smax` is the maximum allowable tuning parameter sigma, the standard deviation of the conditional distribution, in the AGG algorithm.

- `Temperature` is used in the THMC algorithm to heat up the momentum in the first half of the leapfrog steps, and then cool down the momentum in the last half. `Temperature` must be positive. When greater than 1, THMC should explore more diffuse distributions, and may be helpful with multimodal distributions.
- `Type` is used in the Slice algorithm. It is either a scalar or a list with the same number of list components as blocks. This accepts `"Continuous"` for continuous parameters, `"Nominal"` for discrete parameters that are unordered, and `"Ordinal"` for discrete parameters that are ordered.
- `w` is used in AFSS, AMM, DEMC, Refractive, RSS, and Slice. It is a mixture weight for both the AMM and DEMC algorithms, and in these algorithms it is in the interval (0,1]. For AMM, it is recommended to use `w=0.05`, as per Roberts and Rosenthal (2009). The two mixture components in AMM are adaptive multivariate and static/symmetric univariate proposals. The mixture is determined at each iteration with mixture weight `w`. In the AMM algorithm, a higher value of `w` is associated with more static/symmetric univariate proposals, and a lower `w` is associated with more adaptive multivariate proposals. AMM will be unable to include the multivariate mixture component until it has accumulated some history, and models with more parameters will take longer to be able to use adaptive multivariate proposals. In DEMC, it indicates the probability that each iteration uses a snooker update, rather than a projection update, and the recommended default is `w=0.1`. In the Refractive algorithm, `w` is a scalar step size parameter. In AFSS, RSS, and the Slice algorithms, this is a step size interval for creating the slice interval. In AFSS and RSS, a scalar or vector equal in length the number of initial values is accepted. In Slice, a scalar or a list with a number of list components equal to the number of blocks is accepted.
- `Z` accepts a $T \times J$ matrix or $T \times J \times Nc$ array of thinned samples for T thinned iterations, J parameters, and Nc chains for DEMC. `Z` defaults to `NULL`. The matrix of thinned posterior samples from a previous run may be used, in which case the samples are copied across the chains.

Value

`LaplacesDemon` returns an object of class `demonoid`, and `LaplacesDemon.hpc` returns an object of class `demonoid.hpc` that is a list of objects of class `demonoid`, where the number of components in the list is the number of parallel chains. Each object of class `demonoid` is a list with the following components:

`Acceptance.Rate`
: This is the acceptance rate of the MCMC algorithm, indicating the percentage of iterations in which the proposals were accepted. For more information on acceptance rates, see the `AcceptanceRate` function.

`Algorithm`
: This reports the specific algorithm used.

`Call`
: This is the matched call of `LaplacesDemon`.

`Covar`
: This stores the $K \times K$ proposal covariance matrix (where K is the dimension or number of parameters), variance vector, or list of covariance matrices. If variance or covariance is used for adaptation, then this covariance is returned. Otherwise, the variance of the samples of each parameter is returned. If the model is updated in the future, then this vector, matrix, or list can be used to start the next update where the last update left off. Only the diagonal of this matrix is reported in the associated `print` function.

`CovarDHis` This $N \times K$ matrix stores the diagonal of the proposal covariance matrix of each adaptation in each of N rows for K dimensions, where the dimension is the number of parameters or length of the initial values vector. The proposal covariance matrix should change less over time. An exception is that the AHMC algorithm stores an algorithm specification here, which is not the diagonal of the proposal covariance matrix.

`Deviance` This is a vector of the deviance of the model, with a length equal to the number of thinned samples that were retained. Deviance is useful for considering model fit, and is equal to the sum of the log-likelihood for all rows in the data set, which is then multiplied by negative two.

`DIC1` This is a vector of three values: Dbar, pD, and DIC. Dbar is the mean deviance, pD is a measure of model complexity indicating the effective number of parameters, and DIC is the Deviance Information Criterion, which is a model fit statistic that is the sum of Dbar and pD. `DIC1` is calculated over all retained samples. Note that pD is calculated as `var(Deviance)/2` as in Gelman et al. (2004).

`DIC2` This is identical to `DIC1` above, except that it is calculated over only the samples that were considered by the `BMK.Diagnostic` to be stationary for all parameters. If stationarity (or a lack of trend) is not estimated for all parameters, then `DIC2` is set to missing values.

`Initial.Values` This is the vector of `Initial.Values`, which may have been optimized with the `IterativeQuadrature` or `LaplaceApproximation` function.

`Iterations` This reports the number of `Iterations` for updating.

`LML` This is an approximation of the logarithm of the marginal likelihood of the data (see the `LML` function for more information). `LML` is estimated only with stationary samples, and only with a non-adaptive algorithm, including Adaptive Griddy-Gibbs (AGG), Affine-Invariant Ensemble Sampler (AIES), Componentwise Hit-And-Run (CHARM), Delayed Rejection Metropolis (DRM), Elliptical Slice Sampling (ESS), Gibbs Sampler (Gibbs), Griddy-Gibbs (GG), Hamiltonian Monte Carlo (HMC), Hit-And-Run Metropolis (HARM), Independence Metropolis (IM), Metropolis-Coupled Markov Chain Monte Carlo (MCMCMC), Metropolis-within-Gibbs (MWG), Multiple-Try Metropolis, No-U-Turn Sampler (NUTS), Random Dive Metropolis-Hastings (RDMH), Random-Walk Metropolis (RWM), Reflective Slice Sampler (RSS), Refractive Sampler (Refractive), Reversible-Jump (RJ), Sequential Metropolis-within-Gibbs (SMWG), Slice Sampler (Slice), Stochastic Gradient Langevin Dynamics (SGLD), Tempered Hamiltonian Monte Carlo (THMC), or t-walk (twalk). `LML` is estimated with nonparametric self-normalized importance sampling (NSIS), given LL and the marginal posterior samples of the parameters. `LML` is useful for comparing multiple models with the `BayesFactor` function.

`Minutes` This indicates the number of minutes that `LaplacesDemon` was running, and includes the initial checks as well as time it took the `LaplaceApproximation` function, assessing stationarity, effective sample size (ESS), and creating summaries.

`Model` This contains the model specification `Model`.

`Monitor` This is a vector or matrix of one or more monitored variables, which are variables that were specified in the `Model` function to be observed as chains (or Markov chains, if `Adaptive=0`), but that were not deviance or parameters.

`Parameters` This reports the number of parameters.

`Posterior1` This is a matrix of marginal posterior distributions composed of thinned samples, with a number of rows equal to the number of thinned samples and a number of columns equal to the number of parameters. This matrix includes all thinned samples.

`Posterior2` This is a matrix equal to `Posterior1`, except that rows are included only if stationarity (a lack of trend) is indicated by the `BMK.Diagnostic` for all parameters. If stationarity did not occur, then this matrix is missing.

`Rec.BurnIn.Thinned`
This is the recommended burn-in for the thinned samples, where the value indicates the first row that was stationary across all parameters, and previous rows are discarded as burn-in. Samples considered as burn-in are discarded because they do not represent the target distribution and have not adequately forgotten the initial value of the chain (or Markov chain, if `Adaptive=0`).

`Rec.BurnIn.UnThinned`
This is the recommended burn-in for all samples, in case thinning will not be necessary.

`Rec.Thinning` This is the recommended value for the `Thinning` argument according to the autocorrelation in the thinned samples, and it is limited to the interval [1,1000].

`Specs` This is an optional list of algorithm specifications.

`Status` This is the value in the `Status` argument.

`Summary1` This is a matrix that summarizes the marginal posterior distributions of the parameters, deviance, and monitored variables over all samples in `Posterior1`. The following summary statistics are included: mean, standard deviation, MCSE (Monte Carlo Standard Error), ESS is the effective sample size due to autocorrelation, and finally the 2.5%, 50%, and 97.5% quantiles are reported. MCSE is essentially a standard deviation around the marginal posterior mean that is due to uncertainty associated with using MCMC. The acceptable size of the MCSE depends on the acceptable uncertainty associated around the marginal posterior mean. Laplace's Demon prefers to continue updating until each MCSE is less than 6.27% of each marginal posterior standard deviation (see the `MCSE` and `Consort` functions). The default `IMPS` method is used. Next, the desired precision of ESS depends on the user's goal, and Laplace's Demon prefers to continue until each ESS is at least 100, which should be enough to describe 95% boundaries of an approximately Gaussian distribution (see the `ESS` for more information).

`Summary2` This matrix is identical to the matrix in `Summary1`, except that it is calculated only on the stationary samples found in `Posterior2`. If universal stationarity was not estimated for the parameters, then this matrix is set to missing values.

`Thinned.Samples`
This is the number of thinned samples that were retained.

`Thinning` This is the value of the `Thinning` argument.

Author(s)

Statisticat, LLC., Silvere Vialet-Chabrand <silvere@vialet-chabrand.com>

References

Atchade, Y.F. (2006). "An Adaptive Version for the Metropolis Adjusted Langevin Algorithm with a Truncated Drift". *Methodology and Computing in Applied Probability*, 8, p. 235–254.

Bai, Y. (2009). "An Adaptive Directional Metropolis-within-Gibbs Algorithm". Technical Report in Department of Statistics at the University of Toronto.

Beskos, A., Roberts, G.O., Stuart, A.M., and Voss, J. (2008). "MCMC Methods for Diffusion Bridges". Stoch. Dyn., 8, p. 319–350.

Boyles, L.B. and Welling, M. (2012). "Refractive Sampling".

Craiu, R.V., Rosenthal, J., and Yang, C. (2009). "Learn From Thy Neighbor: Parallel-Chain and Regional Adaptive MCMC". *Journal of the American Statistical Assocation*, 104(488), p. 1454–1466.

Christen, J.A. and Fox, C. (2010). "A General Purpose Sampling Algorithm for Continuous Distributions (the t-walk)". *Bayesian Analysis*, 5(2), p. 263–282.

Dutta, S. (2012). "Multiplicative Random Walk Metropolis-Hastings on the Real Line". *Sankhya B*, 74(2), p. 315–342.

Duane, S., Kennedy, A.D., Pendleton, B.J., and Roweth, D. (1987). "Hybrid Monte Carlo". *Physics Letters*, B, 195, p. 216–222.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). "Bayesian Data Analysis, Texts in Statistical Science, 2nd ed.". Chapman and Hall, London.

Geman, S. and Geman, D. (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6), p. 721–741.

Geyer, C.J. (1991). "Markov Chain Monte Carlo Maximum Likelihood". In Keramidas, E.M. Computing Science and Statistics: Proceedings of the 23rd Symposium of the Interface. Fairfax Station VA: Interface Foundation. p. 156–163.

Goodman J, and Weare, J. (2010). "Ensemble Samplers with Affine Invariance". *Communications in Applied Mathematics and Computational Science*, 5(1), p. 65–80.

Green, P.J. (1995). "Reversible Jump Markov Chain Monte Carlo Computation and Bayesian Model Determination". *Biometrika*, 82, p. 711–732.

Haario, H., Laine, M., Mira, A., and Saksman, E. (2006). "DRAM: Efficient Adaptive MCMC". *Statistical Computing*, 16, p. 339–354.

Haario, H., Saksman, E., and Tamminen, J. (2001). "An Adaptive Metropolis Algorithm". *Bernoulli*, 7, p. 223–242.

Hoffman, M.D. and Gelman. A. (2012). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo". *Journal of Machine Learning Research*, p. 1–30.

Kass, R.E. and Raftery, A.E. (1995). "Bayes Factors". *Journal of the American Statistical Association*, 90(430), p. 773–795.

Lewis, S.M. and Raftery, A.E. (1997). "Estimating Bayes Factors via Posterior Simulation with the Laplace-Metropolis Estimator". *Journal of the American Statistical Association*, 92, p. 648–655.

Liu, J., Liang, F., and Wong, W. (2000). "The Multiple-Try Method and Local Optimization in Metropolis Sampling". *Journal of the American Statistical Association*, 95, p. 121–134.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., and Teller, E. (1953). "Equation of State Calculations by Fast Computing Machines". *Journal of Chemical Physics*, 21, p. 1087–1092.

Mira, A. (2001). "On Metropolis-Hastings Algorithms with Delayed Rejection". *Metron*, Vol. LIX, n. 3-4, p. 231–241.

Murray, I., Adams, R.P., and MacKay, D.J. (2010). "Elliptical Slice Sampling". *Journal of Machine Learning Research*, 9, p. 541–548.

Neal, R.M. (2003). "Slice Sampling" (with discussion). *Annals of Statistics*, 31(3), p. 705–767.

Ritter, C. and Tanner, M. (1992), "Facilitating the Gibbs Sampler: the Gibbs Stopper and the Griddy-Gibbs Sampler", *Journal of the American Statistical Association*, 87, p. 861–868.

Roberts, G.O. and Rosenthal, J.S. (2009). "Examples of Adaptive MCMC". *Computational Statistics and Data Analysis*, 18, p. 349–367.

Roberts, G.O. and Tweedie, R.L. (1996). "Exponential Convergence of Langevin Distributions and Their Discrete Approximations". *Bernoulli*, 2(4), p. 341–363.

Rosenthal, J.S. (2007). "AMCMC: An R interface for adaptive MCMC". *Computational Statistics and Data Analysis*, 51, p. 5467–5470.

Smith, R.L. (1984). "Efficient Monte Carlo Procedures for Generating Points Uniformly Distributed Over Bounded Region". *Operations Research*, 32, p. 1296–1308.

Ter Braak, C.J.F. and Vrugt, J.A. (2008). "Differential Evolution Markov Chain with Snooker Updater and Fewer Chains", *Statistics and Computing*, 18(4), p. 435–446.

Tibbits, M., Groendyke, C., Haran, M., Liechty, J. (2014). "Automated Factor Slice Sampling". *Journal of Computational and Graphical Statistics*, 23(2), p. 543–563.

Thompson, M.D. (2011). "Slice Sampling with Multivariate Steps". http://hdl.handle.net/1807/31955

Vihola, M. (2011). "Robust Adaptive Metropolis Algorithm with Coerced Acceptance Rate". *Statistics and Computing*. Springer, Netherlands.

Welling, M. and Teh, Y.W. (2011). "Bayesian Learning via Stochastic Gradient Langevin Dynamics". *Proceedings of the 28th International Conference on Machine Learning (ICML)*, p. 681–688.

See Also

AcceptanceRate, as.initial.values, as.parm.names, BayesFactor, Blocks, BMK.Diagnostic, Combine, Consort, dcrmrf, ESS, GIV, is.data, is.model, IterativeQuadrature, LaplaceApproximation, LaplacesDemon.RAM, LML, and MCSE.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#################### Load the LaplacesDemon Library ####################
library(LaplacesDemon)

############################## Demon Data ##############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
```

```
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

############################  Data List Preparation  ############################
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

############################  Model Specification  ############################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }
#library(compiler)
#Model <- cmpfun(Model) #Consider byte-compiling for more speed

set.seed(666)

############################  Initial Values  ############################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)

#############################################################################
# Examples of MCMC Algorithms                                               #
#############################################################################

####################  Automated Factor Slice Sampler  ####################
Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
     Algorithm="AFSS", Specs=list(A=Inf, B=NULL, m=100, n=0, w=1))
Fit
```

```
print(Fit)
#Consort(Fit)
#plot(BMK.Diagnostic(Fit))
#PosteriorChecks(Fit)
#caterpillar.plot(Fit, Parms="beta")
#BurnIn <- Fit$Rec.BurnIn.Thinned
#plot(Fit, BurnIn, MyData, PDF=FALSE)
#Pred <- predict(Fit, Model, MyData, CPUs=1)
#summary(Pred, Discrep="Chi-Square")
#plot(Pred, Style="Covariates", Data=MyData)
#plot(Pred, Style="Density", Rows=1:9)
#plot(Pred, Style="ECDF")
#plot(Pred, Style="Fitted")
#plot(Pred, Style="Jarque-Bera")
#plot(Pred, Style="Predictive Quantiles")
#plot(Pred, Style="Residual Density")
#plot(Pred, Style="Residuals")
#Levene.Test(Pred)
#Importance(Fit, Model, MyData, Discrep="Chi-Square")

#############  Adaptive Directional Metropolis-within-Gibbs  ##############
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="ADMG", Specs=list(n=0, Periodicity=50))

########################  Adaptive Griddy-Gibbs  ##########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AGG", Specs=list(Grid=GaussHermiteQuadRule(3)$nodes,
#     dparm=NULL, smax=Inf, CPUs=1, Packages=NULL, Dyn.libs=NULL))

##################  Adaptive Hamiltonian Monte Carlo  ####################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AHMC", Specs=list(epsilon=0.02, L=2, m=NULL,
#     Periodicity=10))

#########################  Adaptive Metropolis  ##########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AM", Specs=list(Adaptive=500, Periodicity=10))

##################  Adaptive Metropolis-within-Gibbs  ####################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AMWG", Specs=list(B=NULL, n=0, Periodicity=50))

######################  Adaptive-Mixture Metropolis  ####################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AMM", Specs=list(Adaptive=500, B=NULL, n=0,
#     Periodicity=10, w=0.05))
```

```
###################  Affine-Invariant Ensemble Sampler  ###################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="AIES", Specs=list(Nc=2*length(Initial.Values), Z=NULL,
#     beta=2, CPUs=1, Packages=NULL, Dyn.libs=NULL))

#################  Componentwise Hit-And-Run Metropolis  ##################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="CHARM", Specs=NULL)

############  Componentwise Hit-And-Run (Adaptive) Metropolis  #############
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="CHARM", Specs=list(alpha.star=0.44))

#################  Delayed Rejection Adaptive Metropolis  #################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="DRAM", Specs=list(Adaptive=500, Periodicity=10))

####################  Delayed Rejection Metropolis  ####################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="DRM", Specs=NULL)

##################  Differential Evolution Markov Chain  ##################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="DEMC", Specs=list(Nc=3, Z=NULL, gamma=NULL, w=0.1))

######################  Elliptical Slice Sampler  ######################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="ESS", Specs=list(B=NULL))

############################  Gibbs Sampler  ############################
### NOTE: Unlike the other samplers, Gibbs requires specifying a
### function (FC) that draws from full conditionals.
#FC <- function(parm, Data)
#     {
#     ### Parameters
#     beta <- parm[Data$pos.beta]
#     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
#     sigma2 <- sigma*sigma
#     ### Hyperparameters
#     betamu <- rep(0,length(beta))
#     betaprec <- diag(length(beta))/1000
#     ### Update beta
#     XX <- crossprod(Data$X)
#     Xy <- crossprod(Data$X, Data$y)
#     IR <- backsolve(chol(XX/sigma2 + betaprec), diag(length(beta)))
#     btilde <- crossprod(t(IR)) %*% (Xy/sigma2 + betaprec %*% betamu)
```

```
#     beta <- btilde + IR %*% rnorm(length(beta))
#     return(c(beta,sigma))
#     }
##library(compiler)
##FC <- cmpfun(FC) #Consider byte-compiling for more speed
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="Gibbs", Specs=list(FC=FC, MWG=pos.sigma))


############################  Griddy-Gibbs  ############################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="GG", Specs=list(Grid=seq(from=-0.1, to=0.1, len=5),
#     dparm=NULL, CPUs=1, Packages=NULL, Dyn.libs=NULL))

######################  Hamiltonian Monte Carlo  ######################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="HMC", Specs=list(epsilon=0.001, L=2, m=NULL))

############  Hamiltonian Monte Carlo with Dual-Averaging  ###############
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=1, Thinning=1,
#     Algorithm="HMCDA", Specs=list(A=500, delta=0.65, epsilon=NULL,
#     Lmax=1000, lambda=0.1))

######################  Hit-And-Run Metropolis  ######################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="HARM", Specs=NULL)

#################  Hit-And-Run (Adaptive) Metropolis  ##################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="HARM", Specs=list(alpha.star=0.234, B=NULL))

######################  Independence Metropolis  ######################
### Note: the mu and Covar arguments are populated from a previous Laplace
### Approximation.
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=Fit$Covar, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="IM",
#     Specs=list(mu=Fit$Summary1[1:length(Initial.Values),1]))

########################  Interchain Adaptation  ########################
#Initial.Values <- rbind(Initial.Values, GIV(Model, MyData, PGF=TRUE))
#Fit <- LaplacesDemon.hpc(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="INCA", Specs=list(Adaptive=500, Periodicity=10),
#     LogFile="MyLog", Chains=2, CPUs=2, Type="PSOCK", Packages=NULL,
#     Dyn.libs=NULL)
```

```
################  Metropolis-Adjusted Langevin Algorithm  #################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="MALA", Specs=list(A=1e7, alpha.star=0.574, gamma=1,
#          delta=1, epsilon=c(1e-6,1e-7)))

#############  Metropolis-Coupled Markov Chain Monte Carlo  ###############
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="MCMCMC", Specs=list(lambda=1, CPUs=2, Packages=NULL,
#     Dyn.libs=NULL))

######################  Metropolis-within-Gibbs  ########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="MWG", Specs=list(B=NULL))

######################  Multiple-Try Metropolis  ########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="MTM", Specs=list(K=4, CPUs=1, Packages=NULL, Dyn.libs=NULL))

#########################  No-U-Turn Sampler  ############################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=1, Thinning=1,
#     Algorithm="NUTS", Specs=list(A=500, delta=0.6, epsilon=NULL,
#     Lmax=Inf))

################  Oblique Hyperrectangle Slice Sampler  ################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="OHSS", Specs=list(A=Inf, n=0))

####################  Preconditioned Crank-Nicolson  ###################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="pCN", Specs=list(beta=0.1))

######################  Robust Adaptive Metropolis  ######################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="RAM", Specs=list(alpha.star=0.234, B=NULL, Dist="N",
#     gamma=0.66, n=0))

##################  Random Dive Metropolis-Hastings  ###################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="RDMH", Specs=NULL)

#########################  Refractive Sampler  ###########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="Refractive", Specs=list(Adaptive=1, m=2, w=0.1, r=1.3))
```

```
############################  Reversible-Jump  ############################
#bin.n <- J-1
#bin.p <- 0.2
#parm.p <- c(1, rep(1/(J-1),(J-1)), 1)
#selectable <- c(0, rep(1,J-1), 0)
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="RJ", Specs=list(bin.n=bin.n, bin.p=bin.p,
#          parm.p=parm.p, selectable=selectable,
#          selected=c(0,rep(1,J-1),0)))

########################  Random-Walk Metropolis  ########################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="RWM", Specs=NULL)

########################  Reflective Slice Sampler  ######################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="RSS", Specs=list(m=5, w=1e-5))

##############  Sequential Adaptive Metropolis-within-Gibbs  ##############
#NOTE: The SAMWG algorithm is only for state-space models (SSMs)
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="SAMWG", Specs=list(Dyn=Dyn, Periodicity=50))

##################  Sequential Metropolis-within-Gibbs  ##################
#NOTE: The SMWG algorithm is only for state-space models (SSMs)
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="SMWG", Specs=list(Dyn=Dyn))

############################  Slice Sampler  ############################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=1, Thinning=1,
#     Algorithm="Slice", Specs=list(B=NULL, Bounds=c(-Inf,Inf), m=100,
#     Type="Continuous", w=1))

#################  Stochastic Gradient Langevin Dynamics  #################
#NOTE: The Data and Model functions must be coded differently for SGLD.
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=10, Thinning=10,
#     Algorithm="SGLD", Specs=list(epsilon=1e-4, file="X.csv", Nr=1e4,
#     Nc=6, size=10))

###################  Tempered Hamiltonian Monte Carlo  ###################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="THMC", Specs=list(epsilon=0.001, L=2, m=NULL,
#     Temperature=2))
```

```
################################  t-walk  ###################################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="twalk", Specs=list(SIV=NULL, n1=4, at=6, aw=1.5))

#################  Univariate Eigenvector Slice Sampler  #################
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=1000, Status=100, Thinning=1,
#     Algorithm="UESS", Specs=list(A=Inf, B=NULL, m=100, n=0))

##########  Updating Sequential Adaptive Metropolis-within-Gibbs  #########
#NOTE: The USAMWG algorithm is only for state-space model updating
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=100000, Status=100, Thinning=100,
#     Algorithm="USAMWG", Specs=list(Dyn=Dyn, Periodicity=50, Fit=Fit,
#     Begin=T.m))

##############  Updating Sequential Metropolis-within-Gibbs  ##############
#NOTE: The USMWG algorithm is only for state-space model updating
#Fit <- LaplacesDemon(Model, Data=MyData, Initial.Values,
#     Covar=NULL, Iterations=100000, Status=100, Thinning=100,
#     Algorithm="USMWG", Specs=list(Dyn=Dyn, Fit=Fit, Begin=T.m))

#End
```

`LaplacesDemon.RAM` *LaplacesDemon RAM Estimate*

Description

This function estimates the random-access memory (RAM) required to update a given model and data with the `LaplacesDemon` function.

Warning: Unwise use of this function may crash a computer, so please read the details below.

Usage

```
LaplacesDemon.RAM(Model, Data, Iterations, Thinning, Algorithm="RWM")
```

Arguments

`Model`	This is a model specification function. For more information, see `LaplacesDemon`.
`Data`	This is a list of Data. For more information, see `LaplacesDemon`.
`Iterations`	This is the number of iterations for which `LaplacesDemon` would update. For more information, see `LaplacesDemon`.
`Thinning`	This is the amount of thinning applied to the chains in `LaplacesDemon`.For more information, see `LaplacesDemon`.
`Algorithm`	This argument accepts the name of the algorithm as a string, as entered in `LaplacesDemon`.For more information, see `LaplacesDemon`.

Details

The LaplacesDemon.RAM function uses the object.size function to estimate the size in MB of RAM required to update one chain in LaplacesDemon for a given model and data, and for a number of iterations and specified thinning. When RAM is exceeded, the computer will crash. This function can be useful when trying to estimate how many iterations to update a model without crashing the computer. However, when estimating the required RAM, LaplacesDemon.RAM actually creates several large objects, such as post (see below). If too many iterations are given as an argument to LaplacesDemon.RAM, for example, then it will crash the computer while trying to estimate the required RAM.

The best way to use this function is as follows. First, prepare the model specification and list of data. Second, observe how much RAM the computer is using at the moment, as well as the maximum available RAM. The majority of the difference of these two is the amount of RAM the computer may dedicate to updating the model. Next, use this function with a small number of iterations (important in some algorithms), and with few thinned samples (important in all algorithms). Note the estimated RAM. Increase the number of iterations and thinned samples, and again note the RAM. Continue to increase the number of iterations and thinned samples until, say, arbitrarily within 90% of the above-mentioned difference in RAM.

The computer operating system uses RAM, as does any other software running at the moment. R is currently using RAM, and other functions in the LaplacesDemon package, and any other package that is currently activated, are using RAM. There are numerous small objects that are not included in the returned list, that use RAM. For example, there may be a scalar called alpha for the acceptance probability, etc.

One potentially larger object that is not included, and depends on the algorithm, is a matrix used for estimating LML. Its use occurs with non-adaptive MCMC algorithms, only with enough globally stationary samples, and only when the ratio of parameters to samples is not excessive. If used, then the user should create a matrix of the appropriate dimensions and use the object.size function to estimate the RAM.

If the data is too large for RAM, then consider using either the BigData function or the SGLD algorithm in LaplacesDemon.

Value

LaplacesDemon.RAM returns a list with several components. Each component is an estimate in MB for an object. The list has the following components:

Covar	This is the estimated size in MB of RAM required for the covariance matrix, variance vector, or both (some algorithms store both internally, creating one from the other). Blocked covariance matrices are not considered at this time.
Data	This is the estimated size in MB of RAM required for the list of data.
Deviance	This is the estimated size in MB of RAM required for the deviance vector.
Initial.Values	This is the estimated size in MB of RAM required for the vector of initial values.
Model	This is the estimated size in MB of RAM required for the model specification function.
Monitor	This is the estimated size in MB of RAM required for the $N \times J$ matrix Monitor, where N is the number of thinned samples and J is the number of monitored variables.

post	This is the estimated size in MB of RAM required for a matrix of posterior samples. This matrix is used in some algorithms, and is not returned by LaplacesDemon.
Posterior1	This is the estimated size in MB of RAM required for the $N \times J$ matrix Posterior1, where N is the number of thinned samples and J is the number of initial values or parameters.
Posterior2	This is the estimated size in MB of RAM required for the $N \times J$ matrix Posterior2, where N is the number of globally stationary thinned samples and J is the number of initial values or parameters. Maximum RAM use is assumed here, so the same N is used, as in Posterior1.
Summary1	This is the estimated size in MB of RAM required for the summary table of all thinned posterior samples of parameters, deviance, and monitored variables.
Summary2	This is the estimated size in MB of RAM required for the summary table of all globally stationary thinned posterior samples of parameters, deviance, and monitored variables.
Total	This is the estimated size in MB of RAM required in total to update one chain in LaplacesDemon for a given model and data, and for a number of iterations and specified thinning.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

See Also

BigData, LaplacesDemon, LML, and object.size.

Levene.Test *Levene's Test*

Description

The Levene.Test function is a Bayesian form of Levene's test (Levene, 1960) of equality of variances.

Usage

```
Levene.Test(x, Method="U", G=NULL, Data=NULL)
```

Arguments

x	This required argument must be an object of class demonoid.ppc, iterquad.ppc, laplace.ppc, pmc.ppc, or vb.ppc.
Method	The method defaults to U for a univariate dependent variable (DV), y. When the DV is multivariate, Method="C" applies Levene's test to each column associated in Y. When Method="R", Levene's test is applied to each row associated in Y.

`G` This argument defaults to `NULL`, or is required to have the same dimensions as the DV. For example, if the DV is univariate, then G must have a length equal to y, which is usually represented with a length of N for the number of records or T for the number of time-periods. If the DV is multivariate, then `G` must be a matrix, like Y, and have the same number of rows and columns. The purpose of the `G` argument is to allow the user to specify each element of y or Y to be in a particular group, so the variance of the groups can be tested. As such, each element of `G` must consist of an integer from one to the number of groups desired to be tested. The reason this test allows this degree of specificity is so the user can specify groups, such as according to covariate levels. By default, 4 groups are specified, where the first quarter of the records are group 1 and the last quarter of the records are group 4.

`Data` This argument is required when the DV is multivariate, hence when `Method="C"` or `Method="R"`. The DV is required to be named Y.

Details

This function is a Bayesian form of Levene's test. Levene's test is used to assess the probability of the equality of residual variances in different groups. When residual variance does not differ by group, it is often called homoscedastic (or homoskedastic) residual variance. Homoskedastic residual variance is a common assumption. An advantage of Levene's test to other tests of homoskedastic residual variance is that Levene's test does not require normality of the residuals.

The `Levene.Test` function estimates the test statistic, W, as per Levene's test. This Bayesian form, however, estimates W from the observed residuals as W^{obs}, and W from residuals that are replicated from a homoskedastic process as W^{rep}. Further, W^{obs} and W^{rep} are estimated for each posterior sample. Finally, the probability that the distribution of W^{obs} is greater than the distribution of W^{rep} is reported (see below).

Value

The `Levene.Test` function returns a plot (or for multivariate Y, a series of plots), and a vector with a length equal to the number of Levene's tests conducted.

One plot is produced per univariate application of Levene's test. Each plot shows the test statistic W, both from the observed process (W.obs as a black density) and the replicated process (W.rep as a red line). The mean of W.obs is reported, along with its 95% quantile-based probability interval (see `p.interval`), the probability $p(W^{obs} > W^{rep})$, and the indicated results, either homoskedastic or heteroskedastic.

Each element of the returned vector is the probability $p(W^{obs} > W^{rep})$. When the probability is $p < 0.025$ or $p > 0.975$, heteroskedastic variance is indicated. Otherwise, the variances of the groups are assumed not to differ effectively.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Levene, H. (1960). "Robust Tests for Equality of Variances". In I. Olkins, S. G. Ghurye, W. Hoeffding, W. G. Madow, & H. B. Mann (Eds.), *Contributions to Probability and Statistics*, p. 278–292. Stanford University Press: Stanford, CA.

See Also

IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, p.interval, and VariationalBayes.

Examples

```
#First, update the model with IterativeQuadrature, LaplaceApproximation,
#  LaplacesDemon, PMC, or VariationalBayes.
#Then, use the predict function, creating, say, object Pred.
#Finally:
#Levene.Test(Pred)
```

LML *Logarithm of the Marginal Likelihood*

Description

This function approximates the logarithm of the marginal likelihood (LML), where the marginal likelihood is also called the integrated likelihood or the prior predictive distribution of $\mathbf{y}$ in Bayesian inference. The marginal likelihood is

$$p(\mathbf{y}) = \int p(\mathbf{y}|\Theta)p(\Theta)d\Theta$$

The prior predictive distribution indicates what $\mathbf{y}$ should look like, given the model, before $\mathbf{y}$ has been observed. The presence of the marginal likelihood of $\mathbf{y}$ normalizes the joint posterior distribution, $p(\Theta|\mathbf{y})$, ensuring it is a proper distribution and integrates to one (see is.proper). The marginal likelihood is the denominator of Bayes' theorem, and is often omitted, serving as a constant of proportionality. Several methods of approximation are available.

Usage

```
LML(Model=NULL, Data=NULL, Modes=NULL, theta=NULL, LL=NULL, Covar=NULL,
method="NSIS")
```

Arguments

Model	This is the model specification for the model that was updated either in IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, or VariationalBayes. This argument is used only with the LME method.
Data	This is the list of data passed to the model specification. This argument is used only with the LME method.

Modes	This is a vector of the posterior modes (or medians, in the case of MCMC). This argument is used only with the GD or LME methods.
theta	This is a matrix of posterior samples (parameters only), and is specified only with the GD, HME, or NSIS methods.
LL	This is a vector of MCMC samples of the log-likelihood, and is specified only with the GD, codeHME, or NSIS methods.
Covar	This argument accepts the covariance matrix of the posterior modes, and is used only with the GD or LME methods.
method	The method may be "GD", "HME", "LME", or "NSIS", and defaults to "NSIS". "GD" uses the Gelfand-Dey estimator, "HME" uses the Harmonic Mean Estimator, "LME" uses the Laplace-Metropolis Estimator, and "NSIS" uses nonparametric self-normalized importance sampling (NSIS).

Details

Generally, a user of LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, or VariationalBayes does not need to use the LML function, because these methods already include it. However, LML may be called by the user, should the user desire to estimate the logarithm of the marginal likelihood with a different method, or with non-stationary chains. The LaplacesDemon and LaplacesDemon.hpc functions only call LML when all parameters are stationary, and only with non-adaptive algorithms.

The GD method, where GD stands for Gelfand-Dey (1994), is a modification of the harmonic mean estimator (HME) that results in a more stable estimator of the logarithm of the marginal likelihood. This method is unbiased, simulation-consistent, and usually satisfies the Gaussian central limit theorem.

The HME method, where HME stands for harmonic mean estimator, of Newton-Raftery (1994) is the easiest, and therefore fastest, estimation of the logarithm of the marginal likelihood. However, it is an unreliable estimator and should be avoided, because small likelihood values can overly influence the estimator, variance is often infinite, and the Gaussian central limit theorem is usually not satisfied. It is included here for completeness. There is not a function in this package that uses this method by default. Given N samples, the estimator is $1/[\frac{1}{N}\sum_N \exp(-LL)]$.

The LME method uses the Laplace-Metropolis Estimator (LME), in which the estimation of the Hessian matrix is approximated numerically. It is the slowest method here, though it returns an estimate in more cases than the other methods. The supplied Model specification must be executed a number of times equal to $k^2 \times 4$, where k is the number of parameters. In large dimensions, this is very slow. The Laplace-Metropolis Estimator is inappropriate with hierarchical models. The IterativeQuadrature, LaplaceApproximation, and VariationalBayes functions use LME when it has converged and sir=FALSE, in which case it uses the posterior means or modes, and is itself Laplace Approximation.

The Laplace-Metropolis Estimator (LME) is the logarithmic form of equation 4 in Lewis and Raftery (1997). In a non-hierarchical model, the marginal likelihood may easily be approximated with the Laplace-Metropolis Estimator for model m as

$$p(\mathbf{y}|m) = (2\pi)^{d_m/2}|\Sigma_m|^{1/2}p(\mathbf{y}|\Theta_m, m)p(\Theta_m|m)$$

where d is the number of parameters and Σ is the inverse of the negative of the approximated Hessian matrix of second derivatives.

As a rough estimate of Kass and Raftery (1995), LME is worrisome when the sample size of the data is less than five times the number of parameters, and LME should be adequate in most problems when the sample size of the data exceeds twenty times the number of parameters (p. 778).

The NSIS method is essentially the MarginalLikelihood function in the MargLikArrogance package. After HME, this is the fastest method available here. The IterativeQuadrature, LaplaceApproximation, and VariationalBayes functions use NSIS when converged and sir=TRUE. The LaplacesDemon, LaplacesDemon.hpc, and PMC functions use NSIS. At least 301 stationary samples are required, and the number of parameters cannot exceed half the number of stationary samples.

Value

LML returns a list with two components:

LML	This is an approximation of the logarithm of the marginal likelihood (LML), which is notoriously difficult to estimate. For this reason, several methods are provided. The marginal likelihood is useful when comparing models, such as with Bayes factors in the BayesFactor function. When the method fails, NA is returned, and it is most likely that the joint posterior is improper (see is.proper).
VarCov	This is a variance-covariance matrix, and is the negative inverse of the Hessian matrix, if estimated. The GD, HME, and NSIS methods do not estimate VarCov, and return NA.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Gelfand, A.E. and Dey, D.K. (1994). "Bayesian Model Choice: Asymptotics and Exact Calculations". *Journal of the Royal Statistical Society*, Series B 56, p. 501–514.

Kass, R.E. and Raftery, A.E. (1995). "Bayes Factors". *Journal of the American Statistical Association*, 90(430), p. 773–795.

Lewis, S.M. and Raftery, A.E. (1997). "Estimating Bayes Factors via Posterior Simulation with the Laplace-Metropolis Estimator". *Journal of the American Statistical Association*, 92, p. 648–655.

Newton, M.A. and Raftery, A.E. (1994). "Approximate Bayesian Inference by the Weighted Likelihood Bootstrap". *Journal of the Royal Statistical Society*, Series B 3, p. 3–48.

See Also

BayesFactor, is.proper, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LaplacesDemon.hpc, PMC, and VariationalBayes.

Examples

```
### If a model object were created and called Fit, then:
#
### Applying HME to an object of class demonoid or pmc:
#LML(LL=Fit$Deviance*(-1/2), method="HME")
```

```
#
### Applying LME to an object of class demonoid:
#LML(Model, MyData, Modes=apply(Fit$Posterior1, 2, median), method="LME")
#
### Applying NSIS to an object of class demonoid
#LML(theta=Fit$Posterior1, LL=Fit$Deviance*-(1/2), method="NSIS")
#
### Applying LME to an object of class iterquad:
#LML(Model, MyData, Modes=Fit$Summary1[,1], method="LME")
#
### Applying LME to an object of class laplace:
#LML(Model, MyData, Modes=Fit$Summary1[,1], method="LME")
#
### Applying LME to an object of class vb:
#LML(Model, MyData, Modes=Fit$Summary1[,1], method="LME")
```

`log-log` *The log-log and complementary log-log functions*

Description

The log-log and complementary log-log functions, as well as the inverse functions, are provided.

Usage

```
cloglog(p)
invcloglog(x)
invloglog(x)
loglog(p)
```

Arguments

`x`	This is a vector of real values that will be transformed to the interval [0,1].
`p`	This is a vector of probabilities p in the interval [0,1] that will be transformed to the real line.

Details

The logit and probit links are symmetric, because the probabilities approach zero or one at the same rate. The log-log and complementary log-log links are asymmetric. Complementary log-log links approach zero slowly and one quickly. Log-log links approach zero quickly and one slowly. Either the log-log or complementary log-log link will tend to fit better than logistic and probit, and are frequently used when the probability of an event is small or large. A mixture of the two links, the log-log and complementary log-log is often used, where each link is weighted. The reason that logit is so prevalent is because logistic parameters can be interpreted as odds ratios.

Value

`cloglog` returns `x`, `invcloglog` and `invloglog` return probability `p`, and `loglog` returns `x`.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

LaplacesDemon

Examples

```
library(LaplacesDemon)
x <- -5:5
p <- invloglog(x)
x <- loglog(p)
```

---

logit *The logit and inverse-logit functions*

Description

The logit and inverse-logit (also called the logistic function) are provided.

Usage

```
invlogit(x)
logit(p)
```

Arguments

x	This object contains real values that will be transformed to the interval [0,1].
p	This object contains of probabilities p in the interval [0,1] that will be transformed to the real line.

Details

The `logit` function is the inverse of the sigmoid or logistic function, and transforms a continuous value (usually probability p) in the interval [0,1] to the real line (where it is usually the logarithm of the odds). The `logit` function is $\log(p/(1-p))$.

The `invlogit` function (called either the inverse logit or the logistic function) transforms a real number (usually the logarithm of the odds) to a value (usually probability p) in the interval [0,1]. The `invlogit` function is $\frac{1}{1+\exp(-x)}$.

If p is a probability, then $\frac{p}{1-p}$ is the corresponding odds, while the `logit` of p is the logarithm of the odds. The difference between the logits of two probabilities is the logarithm of the odds ratio. The derivative of probability p in a logistic function (such as `invlogit`) is: $\frac{d}{dx} = p(1-p)$.

In the LaplacesDemon package, it is common to re-parameterize a model so that a parameter that should be in an interval can be updated from the real line by using the `logit` and `invlogit` functions, though the `interval` function provides an alternative. For example, consider a parameter θ

that must be in the interval [0,1]. The algorithms in IterativeQuadrature, LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes are unaware of the desired interval, and may attempt θ outside of this interval. One solution is to have the algorithms update `logit(theta)` rather than `theta`. After `logit(theta)` is manipulated by the algorithm, it is transformed via `invlogit(theta)` in the model specification function, where $\theta \in [0, 1]$.

Value

`invlogit` returns probability `p`, and `logit` returns `x`.

See Also

interval, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, plogis, PMC, qlogis, and VariationalBayes.

Examples

```
library(LaplacesDemon)
x <- -5:5
p <- invlogit(x)
x <- logit(p)
```

LossMatrix *Loss Matrix*

Description

A loss matrix is useful in Bayesian decision theory for selecting the Bayes action, the optimal Bayesian decision, when there are a discrete set of possible choices (actions) and a discrete set of possible outcomes (states of the world). The Bayes action is the action that minimizes expected loss, which is equivalent to maximizing expected utility.

Usage

```
LossMatrix(L, p.theta)
```

Arguments

`L`	This required argument accepts a $S \times A$ matrix or $S \times A \times N$ array of losses, where S is the number of states of the world, A is the number of actions, and N is the number of samples. These losses have already been estimated, given a personal loss function. One or more personal losses has already been estimated for each combination of possible actions $a = 1, \ldots, A$ and possible states $s = 1, \ldots, S$.
`p.theta`	This required argument accepts a $S \times A$ matrix or $S \times A \times N$ array of state prior probabilities, where S is the number of states of the world, A is the number of actions, and N is the number of samples. The sum of each column must equal one.

Details

Bayesian inference is often tied to decision theory (Bernardo and Smith, 2000), and decision theory has long been considered the foundations of statistics (Savage, 1954).

Before using the `LossMatrix` function, the user should have already considered all possible actions (choices), states of the world (outcomes unknown at the time of decision-making), chosen a loss function $L(\theta, \alpha)$, estimated loss, and elicited prior probabilities $p(\theta|x)$.

Although possible actions (choices) for the decision-maker and possible states (outcomes) may be continuous or discrete, the loss matrix is used for discrete actions and states. An example of a continuous action may be that a decision-maker has already decided to invest, and the remaining, current decision is how much to invest. Likewise, an example of continuous states of the world (outcomes) may be how much profit or loss may occur after a given continuous unit of time.

The coded example provided below is taken from Berger (1985, p. 6-7) and described here. The set of possible actions for a decision-maker is to invest in bond ZZZ or alternatively in bond XXX, as it is called here. A real-world decision should include a mutually exhaustive list of actions, such as investing in neither, but perhaps the decision-maker has already decided to invest and narrowed the options down to these two bonds.

The possible states of the world (outcomes unknown at the time of decision-making) are considered to be two states: either the chosen bond will not default or it will default. Here, the loss function is a negative linear identity of money, and hence a loss in element `L[1,1]` of -500 is a profit of 500, while a loss in `L[2,1]` of 1,000 is a loss of 1,000.

The decision-maker's dilemma is that bond ZZZ may return a higher profit than bond XXX, however there is an estimated 10% chance, the prior probability, that bond ZZZ will default and return a substantial loss. In contrast, bond XXX is considered to be a sure-thing and return a steady but smaller profit. The Bayes action is to choose the first action and invest in bond ZZZ, because it minimizes expected loss, even though there is a chance of default.

A more realistic application of a loss matrix may be to replace the point-estimates of loss with samples given uncertainty around the estimated loss, and replace the point-estimates of the prior probability of each state with samples given the uncertainty of the probability of each state. The loss function used in the example is intuitive, but a more popular monetary loss function may be $-\log(E(W|R))$, the negative log of the expectation of wealth, given the return. There are many alternative loss functions.

Although isolated decision-theoretic problems exist such as the provided example, decision theory may also be applied to the results of a probability model (such as from `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`), or `VariationalBayes`, contingent on how a decision-maker is considering to use the information from the model. The statistician may pass the results of a model to a client, who then considers choosing possible actions, given this information. The statistician should further assist the client with considering actions, states of the world, then loss functions, and finally eliciting the client's prior probabilities (such as with the `elicit` function).

When the outcome is finally observed, the information from this outcome may be used to refine the priors of the next such decision. In this way, Bayesian learning occurs.

Value

The `LossMatrix` function returns a list with two components:

`BayesAction` This is a numeric scalar that indicates the action that minimizes expected loss.

`E.Loss` This is a vector of expected losses, one for each action.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Berger, J.O. (1985). "Statistical Decision Theory and Bayesian Analysis", Second Edition. Springer: New York, NY.

Bernardo, J.M. and Smith, A.F.M. (2000). "Bayesian Theory". John Wiley \& Sons: West Sussex, England.

Savage, L.J. (1954). "The Foundations of Statistics". John Wiley \& Sons: West Sussex, England.

See Also

`elicit`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, and `VariationalBayes`.

Examples

```
library(LaplacesDemon)
### Point-estimated loss and state probabilities
L <- matrix(c(-500,1000,-300,-300), 2, 2)
rownames(L) <- c("s[1]: !Defaults","s[2]: Defaults")
colnames(L) <- c("a[1]: Buy ZZZ", "a[2]: Buy XXX")
L
p.theta <- matrix(c(0.9, 0.1, 1, 0), 2, 2)
Fit <- LossMatrix(L, p.theta)

### Point-estimated loss and samples of state probabilities
L <- matrix(c(-500,1000,-300,-300), 2, 2)
rownames(L) <- c("s[1]: Defaults","s[2]: !Defaults")
colnames(L) <- c("a[1]: Buy ZZZ", "a[2]: Buy XXX")
L
p.theta <- array(runif(4000), dim=c(2,2,1000)) #Random probabilities,
#just for a quick example. And, since they must sum to one:
for (i in 1:1000) {
     p.theta[,,i] <- p.theta[,,i] / matrix(colSums(p.theta[,,i]),
          dim(p.theta)[1], dim(p.theta)[2], byrow=TRUE)}
Fit <- LossMatrix(L, p.theta)
Fit

### Point-estimates of loss may be replaced with samples as well.
```

`LPL.interval` *Lowest Posterior Loss Interval*

Description

This function returns the Lowest Posterior Loss (LPL) interval for one parameter, given samples from the density of its prior distribution and samples of the posterior distribution.

Usage

```
LPL.interval(Prior, Posterior, prob=0.95, plot=FALSE, PDF=FALSE)
```

Arguments

`Prior`	This is a vector of samples of the prior density.
`Posterior`	This is a vector of posterior samples.
`prob`	This is a numeric scalar in the interval (0,1) giving the Lowest Posterior Loss (LPL) interval, and defaults to 0.95, representing a 95% LPL interval.
`plot`	Logical. When `plot=TRUE`, two plots are produced. The top plot shows the expected posterior loss. The LPL region is shaded black, and the area outside the region is gray. The bottom plot shows LPL interval of θ on the kernel density of θ. Again, the LPL region is shaded black, and the outside area is gray. A vertical, red, dotted line is added at zero for both plots. The `plot` argument defaults to `FALSE`. The plot treats the distribution as if it were unimodal; disjoint regions are not estimated here. If multimodality should result in disjoint regions, then consider using HPD intervals in the `p.interval` function.
`PDF`	Logical. When `PDF=TRUE`, and only when `plot=TRUE`, plots are saved as a .pdf file in the working directory.

Details

The Lowest Posterior Loss (LPL) interval (Bernardo, 2005), or LPLI, is a probability interval based on intrinsic discrepancy loss between prior and posterior distributions. The expected posterior loss is the loss associated with using a particular value $\theta_i \in \theta$ of the parameter as the unknown true value of θ (Bernardo, 2005). Parameter values with smaller expected posterior loss should always be preferred. The LPL interval includes a region in which all parameter values have smaller expected posterior loss than those outside the region.

Although any loss function could be used, the loss function should be invariant under reparameterization. Any intrinsic loss function is invariant under reparameterization, but not necessarily invariant under one-to-one transformations of data **x**. When a loss function is also invariant under one-to-one transformations, it is usually also invariant when reduced to a sufficient statistic. Only an intrinsic loss function that is invariant when reduced to a sufficient statistic should be considered.

The intrinsic discrepancy loss is easily a superior loss function to the overused quadratic loss function, and is more appropriate than other popular measures, such as Hellinger distance, Kullback-Leibler divergence (`KLD`), and Jeffreys logarithmic divergence. The intrinsic discrepancy loss is

also an information-theory related divergence measure. Intrinsic discrepancy loss is a symmetric, non-negative loss function, and is a continuous, convex function. Intrinsic discrepancy loss was introduced by Bernardo and Rueda (2002) in a different context: hypothesis testing. Formally, it is:

$$\delta f(p_2, p_1) = min[\kappa(p_2|p_1), \kappa(p_1|p_2)]$$

where δ is the discrepancy, κ is the KLD, and p_1 and p_2 are the probability distributions. The intrinsic discrepancy loss is the loss function, and the expected posterior loss is the mean of the directed divergences.

The LPL interval is also called an intrinsic credible interval or intrinsic probability interval, and the area inside the interval is often called an intrinsic credible region or intrinsic probability region.

In practice, whether a reference prior or weakly informative prior (WIP) is used, the LPL interval is usually very close to the HPD interval, though the posterior losses may be noticeably different. If LPL used a zero-one loss function, then the HPD interval would be produced. An advantage of the LPL interval over HPD interval (see `p.interval`) is that the LPL interval is invariant to reparameterization. This is due to the invariant reparameterization property of reference priors. The quantile-based probability interval is also invariant to reparameterization. The LPL interval enjoys the same advantage as the HPD interval does over the quantile-based probability interval: it does not produce equal tails when inappropriate.

Compared with probability intervals, the LPL interval is slightly less convenient to calculate. Although the prior distribution is specified within the `Model` specification function, the user must specify it for the `LPL.interval` function as well. A comparison of the quantile-based probability interval, HPD interval, and LPL interval is available here: https://web.archive.org/web/20150214090353/http://www.bayesian-inference.com/credible.

Value

A matrix is returned with one row and two columns. The row represents the parameter and the column names are `"Lower"` and `"Upper"`. The elements of the matrix are the lower and upper bounds of the LPL interval.

Author(s)

Statisticat, LLC.

References

Bernardo, J.M. (2005). "Intrinsic Credible Regions: An Objective Bayesian Approach to Interval Estimation". *Sociedad de Estadistica e Investigacion Operativa*, 14(2), p. 317–384.

Bernardo, J.M. and Rueda, R. (2002). "Bayesian Hypothesis Testing: A Reference Approach". *International Statistical Review*, 70, p. 351–372.

See Also

KLD, `p.interval`, LaplacesDemon, and PMC.

Examples

```
library(LaplacesDemon)
#Although LPL is intended to be applied to output from LaplacesDemon or
#PMC, here is an example in which p(theta) ~ N(0,100), and
#p(theta | y) ~ N(1,10), given 1000 samples.
theta <- rnorm(1000,1,10)
LPL.interval(Prior=dnorm(theta,0,100^2), Posterior=theta, prob=0.95,
  plot=TRUE)
#A more practical example follows, but it assumes a model has been
#updated with LaplacesDemon or PMC, the output object is called Fit, and
#that the prior for the third parameter is normally distributed with
#mean 0 and variance 100:
#temp <- Fit$Posterior2[,3]
#names(temp) <- colnames(Fit$Posterior2)[3]
#LPL.interval(Prior=dnorm(temp,0,100^2), Posterior=temp, prob=0.95,
#     plot=TRUE, PDF=FALSE)
```

Math *Math Utility Functions*

Description

These are utility functions for math.

Usage

```
GaussHermiteQuadRule(N)
Hermite(x, N, prob=TRUE)
logadd(x, add=TRUE)
partial(Model, parm, Data, Interval=1e-6, Method="simple")
```

Arguments

N	This required argument accepts a positive integer that indicates the number of nodes.
x	This is a numeric vector.
add	Logical. This defaults to TRUE, in which case $\log(x+y)$ is performed. Otherwise, $\log(x-y)$ is performed.
Model	This is a model specification function. For more information, see LaplacesDemon.
parm	This is a vector parameters.
prob	Logical. This defaults to TRUE, which uses the probabilist's kernel for the Hermite polynomial. Otherwise, FALSE uses the physicist's kernel.
Data	This is a list of data. For more information, see LaplacesDemon.
Interval	This is the interval of numeric differencing.

`Method` This accepts a quoted string, and defaults to "simple", which is finite-differencing. Alternatively `Method="Richardson"` uses Richardson extrapolation, which is more accurate, but takes longer to calculate. Another method called automatic differentiation is currently unsupported, but is even more accurate, and takes even longer to calculate.

Details

The `GaussHermiteQuadRule` function returns nodes and weights for univariate Gauss-Hermite quadrature. The nodes and weights are obtained from a tridiagonal eigenvalue problem. Weights are calculated from the physicist's (rather than the probabilist's) kernel. This has been adapted from the GaussHermite function in the pracma package. The `GaussHermiteCubeRule` function is a multivariate version. This is used in the `IterativeQuadrature` function.

The `Hermite` function evaluates a Hermite polynomial of degree N at x, using either the probabilist's (`prob=TRUE`) or physicist's (`prob=FALSE`) kernel. This function was adapted from the `hermite` function in package EQL.

The `logadd` function performs addition (or subtraction) when the terms are logarithmic. The equations are:

$$\log(x+y) = \log(x) + \log(1 + \exp(\log(y) - \log(x)))$$

$$\log(x-y) = \log(x) + \log(1 - \exp(\log(y) - \log(x)))$$

The `partial` function estimates partial derivatives of parameters in a model specification with data, using either forward finite-differencing or Richardson extrapolation. In calculus, a partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant. Related functions include `Jacobian` which returns a matrix of first-order partial derivatives, and `Hessian`, which returns a matrix of second-order partial derivatives of the model specification function with respect to its parameters. The `partial` function is not intended to be called by the user, but is used by other functions. This is essentially the `grad` function in the numDeriv package, but defaulting to forward finite-differencing with a smaller interval.

Value

`logadd` returns the result of $\log(x+y)$ or $\log(x-y)$.

`partial` returns a vector of partial derivatives.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`GaussHermiteCubeRule`, `Hessian`, `IterativeQuadrature`, `Jacobian`, `LaplaceApproximation`, `LaplacesDemon`, and `VariationalBayes`.

Matrices *Matrix Utility Functions*

Description

These are utility functions for working with matrices.

Usage

```
as.indicator.matrix(x)
as.inverse(x)
as.parm.matrix(x, k, parm, Data, a=-Inf, b=Inf, restrict=FALSE, chol=FALSE)
as.positive.definite(x)
as.positive.semidefinite(x)
as.symmetric.matrix(x, k=NULL)
is.positive.definite(x)
is.positive.semidefinite(x)
is.square.matrix(x)
is.symmetric.matrix(x)
Cov2Cor(Sigma)
CovEstim(Model, parm, Data, Method="Hessian")
GaussHermiteCubeRule(N, dims, rule)
Hessian(Model, parm, Data, Interval=1e-6, Method="Richardson")
Jacobian(Model, parm, Data, Interval=1e-6, Method="simple")
logdet(x)
lower.triangle(x, diag=FALSE)
read.matrix(file, header=FALSE, sep=",", nrow=0, samples=0, size=0, na.rm=FALSE)
SparseGrid(J, K)
TransitionMatrix(theta.y=NULL, y.theta=NULL, p.theta=NULL)
tr(x)
upper.triangle(x, diag=FALSE)
```

Arguments

N	This required argument accepts a positive integer that indicates the number of nodes.
x	This is a matrix (though `as.symmetric.matrix` also accepts vectors).
J	This required argument indicates the dimension of the integral and accepts a positive integer.
k	For `as.parm.matrix`, this is a required argument, indicating the dimension of the matrix. For `as.symmetric.matrix`, this is an optional argument that specifies the dimension of the symmetric matrix. This applies only when `x` is a vector. It defaults to `NULL`, in which case it calculates `k <- (-1 + sqrt(1 + 8 * length(x)))/ 2`.
K	This required argument indicates the accuracy and accepts a positive integer. Larger values result in many more integration nodes.

`diag`	Logical. If `TRUE`, then the elements in the main diagonal are also returned.
`dims`	This required argument indicates the dimension of the integral and accepts a positive integer.
`Sigma`	This is a covariance matrix, Σ, and may be entered either as a matrix or vector.
`Model`	This is a model specification function. For more information, see `LaplacesDemon`.
`parm`	This is a vector of parameters passed to the model specification.
`Data`	This is the list of data passed to the model specification. For more information, see `LaplacesDemon`.
`a,b`	These optional arguments allow the elements of `x` to be bound to the interval $[a, b]$. For example, elements of a correlation matrix are in the interval $[-1, 1]$.
`restrict`	Logical. If `TRUE`, then `x[1,1]` is restricted to 1. This is useful in multinomial probit, for example. The variable, `LaplacesDemonMatrix`, is created in a new environment, `LDEnv` so `as.parm.matrix` can keep track of changes from iteration to iteration.
`rule`	This is an optional argument that accepts a univariate Gauss-Hermite quadrature rule. Usually, this argument is left empty. A rule may be supplied that differs from the traditional rule, such as when constraints have been observed, and one or more nodes and weights were adjusted.
`chol`	Logical. If `TRUE`, then x is an upper-triangular matrix.
`file`	This is the name of the file from which the numeric data matrix will be imported or read.
`header`	Logical. When `TRUE`, the first row of the file must contain names of the columns, and will be converted to the column names of the numeric matrix. When `FALSE`, the first row of the file contains data, not column names.
`Interval`	This accepts a small scalar number for precision.
`Method`	This accepts a quoted string. For `Hessian`, it defaults to `Method="Richardson"`, which uses Richardson extrapolation. For `Jacobian`, it defaults to `Method="simple"`, which uses finite-differencing. Richardson Richardson extrapolation is more accurate, but slower to calculate. Since error due to finite-differencing propagates through to higher derivatives, finite-differencing should not be used when approximating a Hessian matrix. Another method called automatic differentiation is not currently available here, but should be more accurate, though even slower to calculate. Another popular alternative is to use the `BayesianBootstrap` on the data. For `CovEstim`, this accepts `Method="Hessian"`, `Method="Identity"` (which simply assigns an identity matrix), `Method="OPG"` (which calculates the sum of outer products of record-level gradients), or `Method="Sandwich"`, which is the sandwich estimator and combines the Hessian and OPG estimates.
`nrow`	This is the number of rows of the numeric matrix, and defaults to `nrow=0`. If the number is known, the function will perform noticeably faster when it does not have to check.
`p.theta`	This accepts a matrix of prior probabilities for a transition matrix, and defaults to `NULL`. If used, each row must sum to 1.
`samples`	This is the number of samples to take from the numeric matrix. When `samples=0`, sampling is not performed and the entire matrix is returned.

sep	This argument indicates a character with which it will separate fields when creating column vectors. For example, a read a comma-separated file (.csv), use sep=",".
size	This is the batch size to be used only when reading a numeric matrix that is larger than the available computer memory (RAM), and only when samples is greater than zero. Sampling of a big data matrix is performed by first determining the records to keep, and then reading batches, one by one, and keeping the matching records.
theta.y	This accepts a vector of posterior samples of a discrete Markov chain, and defaults to NULL. If used, the order of the samples affects the transition probability.
na.rm	Logical. When TRUE, rows with missing values are removed from the matrix after it is read. Rather than removing missing values, the user may consider imputing missing values inside the model, or before the model with the MISS function. Examples of within-model imputation may be found in the accompanying "Examples" vignette.
y.theta	This accepts a vector of data that are samples of a discrete distribution, and defaults to NULL. If used, the order of the samples affects the transition probability.

Details

The as.indicator.matrix function creates an indicator matrix from a vector. This function is useful for converting a discrete vector into a matrix in which each column represents one of the discrete values, and each occurence of that value in the related column is indicated by a one, and is otherwise filled with zeroes. This function is similar to the class.ind function in the nnet package.

The as.inverse function returns the matrix inverse of x. The solve function in base R also returns the matrix inverse, but solve can return a matrix that is not symmetric, and can fail due to singularities. The as.inverse function tries to use the solve function to return a matrix inverse, and when it fails due to a singularity, as.inverse uses eigenvalue decomposition (in which eigenvalues below a tolerance are replaced with the tolerance), and coerces the result to a symmetric matrix. This is similar to the solvcov function in the fpc package.

The as.parm.matrix function prepares a correlation, covariance, or precision matrix in two important ways. First, as.parm.matrix obtains the parameters for the matrix specified in the x argument by matching the name of the matrix in the x argument with any parameters in parm, given the parameter names in the Data listed in parm.names. These obtained parameters are organized into a matrix as the elements of the upper-triangular, including the diagonal. A copy is made, without the diagonal, and the lower-triangular is filled in, completing the matrix. Second, as.parm.matrix checks for positive-definiteness. If matrix x is positive-definite, then the matrix is stored as a variable called LaplacesDemonMatrix in a new environment called LDEnv. If matrix x is not positive-definite, then LaplacesDemonMatrix in LDEnv is sought as a replacement. If this variable exists, then it is used to replace the matrix. If not, then the matrix is replaced with an identity matrix. Back in the model specification, after using as.parm.matrix, it is recommended that the user also pass the resulting matrix back into the parm vector, so the sampler or algorithm knows that the elements of the matrix have changed.

The as.positive.definite function returns the nearest positive-definite matrix for a matrix that is square and symmetric (Higham, 2002). This version is intended only for covariance and precision matrices, and has been optimized for speed. A more extensible function is nearPD in the matrixcalc package, which is also able to work with correlation matrices, and matrices that are asymmetric.

The `as.positive.semidefinite` function iteratively seeks to return a square, symmetric matrix that is at least positive-semidefinite, by replacing each negative eigenvalue and calculating its projection. This is intended only for covariance and precision matrices. A similar function is `makePsd` in the RTAQ package, though it is not iterative, and returns matrices that fail a logical check with `is.positive.semidefinite`.

The `as.symmetric.matrix` function accepts either a vector or matrix, and returns a symmetric matrix. In the case of a vector, it can be either all elements of the matrix, or the lower triangular. In the case of a `x` being entered as a matrix, this function tolerates non-finite values in one triangle (say, the lower), as long as the corresponding element is finite in the other (say, the upper) triangle.

The `Cov2Cor` function converts a covariance matrix into a correlation matrix, and accepts the covariance matrix either in matrix or vector form. This function may be useful inside a model specification and also with converting posterior draws of the elements of a covariance matrix to a correlation matrix. `Cov2Cor` is an expanded form of the `cov2cor` function in the `stats` package, where `Cov2Cor` is also able to accept and return a vectorized matrix.

The `CovEstim` function estimates a covariance matrix with one of several methods. This is mainly used by `LaplaceApproximation`, where the `parm` argument receives the posterior modes. See the `CovEst` argument for more details.

The `GaussHermiteCubeRule` function returns a matrix of nodes and a vector of weights for a `dims`-dimensional integral given N univariate nodes. The number of multivariate nodes will differ from the number of univariate nodes. This function is for use with multivariate quadrature, often called cubature. This has been adapted from the `multiquad` function in the NominalLogisticBiplot package. The `GaussHermiteQuadRule` function is a univariate version. A customized univariate `rule` may be supplied when constraints necessitate that one or more nodes and weights had to be altered.

The `Hessian` returns a symmetric, Hessian matrix, which is a matrix of second partial derivatives. The estimation of the Hessian matrix is approximated numerically using Richardson extrapolation by default. This is a slow function. This function is not intended to be called by the user, but is made available here. This is essentially the `hessian` function from the numDeriv package, adapted to Laplace's Demon.

The `is.positive.definite` function is a logical test of whether or not a matrix is positive-definite. A $k \times k$ symmetric matrix $\mathbf{X}$ is positive-definite if all of its eigenvalues are positive ($\lambda_i > 0, i \in k$). All main-diagonal elements must be positive. The determinant of a positive-definite matrix is always positive, so a positive-definite matrix is always nonsingular. Non-symmetric, positive-definite matrices exist, but are not considered here.

The `is.positive.semidefinite` function is a logical test of whether or not a matrix is positive-semidefinite. A kxk symmetric matrix $\mathbf{X}$ is positive-semidefinite if all of its eigenvalues are non-negative ($\lambda_i \geq 0, i \in k$).

The `is.square.matrix` function is a logical test of whether or not a matrix is square. A square matrix is a matrix with the same number of rows and columns, and is usually represented as a $k \times k$ matrix $\mathbf{X}$.

The `is.symmetric.matrix` function is a logical test of whether or not a matrix is symmetric. A symmetric matrix is a square matrix that is equal to its transpose, $\mathbf{X} = \mathbf{X}^T$. For example, where i indexes rows and j indexes columns, $\mathbf{X}_{i,j} = \mathbf{X}_{j,i}$. This differs from the `isSymmetric` function in base R that is inexact, using `all.equal`.

The `Jacobian` function estimates the Jacobian matrix, which is a matrix of all first-order partial derivatives of the `Model`. The Jacobian matrix is estimated by default with forward finite-differencing, or optionally with Richardson extrapolation. This function is not intended to be called

by the user, but is made available here. This is essentially the `jacobian` function from the numDeriv package, adapted to LaplacesDemon.

The `logdet` function returns the logarithm of the determinant of a positive-definite matrix via the Cholesky decomposition. The determinant is a value associated with a square matrix, and was used historically to *determine* if a system of linear equations has a unique solution. The term *determinant* was introduced by Gauss, where Laplace referred to it as the resultant. When the determinant is zero, the matrix is singular and non-invertible; there are either no solutions or many solutions. A unique solution exists when the determinant is non-zero. The `det` function in base R works well for small matrices, but can return erroneously return zero in larger matrices. It is better to work with the log-determinant.

The `lower.triangle` function returns a vector of the lower triangular elements of a matrix, and the diagonal is included when `diag=TRUE`.

The `read.matrix` function is provided here as one of many convenient ways to read a numeric matrix into R. The most common method of storing data in R is the data frame, because it is versatile. For example, a data frame may contain character, factor, and numeric variables together. For iterative estimation, common in Bayesian inference, the data frame is much slower than the numeric matrix. For this reason, the LaplacesDemon package does not use data frames, and has not traditionally accepted character or factor data. The `read.matrix` function returns either an entire numeric matrix, or row-wise samples from a numeric matrix. Samples may be taken from a matrix that is too large for available computer memory (RAM), such as with big data.

The `SparseGrid` function returns a sparse grid for a J-dimensional integral with accuracy K, given Gauss-Hermite quadrature rules. A grid of order eqnK provides an exact result for a polynomial of total order of $2K - 1$ or less. `SparseGrid` returns a matrix of nodes and a vector of weights. A sparse grid is more efficient than the full grid in the `GaussHermiteCubeRule` function. This has been adapted from the SparseGrid package.

The `TransitionMatrix` function has several uses. A user may supply a vector of marginal posterior samples of a discrete Markov chain as `theta.y`, and an observed posterior transition matrix is returned. Otherwise, a user may supply data (`y.theta`) and/or a prior (`p.theta`), in which case a posterior transition matrix is returned. A common row-wise prior is the dirichlet distribution. Transition probabilities are from row element to column element.

The `tr` function returns the trace of a matrix. The trace of a matrix is the sum of the elements in the main diagonal of a square matrix. For example, the trace of a $k \times k$ matrix $\mathbf{X}$, is $\sum_{k=1} \mathbf{X}_{k,k}$.

The `upper.triangle` function returns a vector of the lower triangular elements of a matrix, and the diagonal is included when `diag=TRUE`.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Higham, N.J. (2002). "Computing the Nearest Correlation Matrix - a Problem from Finance". *IMA Journal of Numerical Analysis*, 22, p. 329–343.

See Also

BayesianBootstrap, Cov2Prec, cov2cor, ddirichlet, GaussHermiteQuadRule, isSymmetric, LaplaceApproximation, LaplacesDemon, lower.tri, MISS, Prec2Cov, solve, and upper.tri.

MCSE *Monte Carlo Standard Error*

Description

Monte Carlo Standard Error (MCSE) is an estimate of the inaccuracy of Monte Carlo samples, usually regarding the expectation of posterior samples, $\mathrm{E}(\theta)$, from Monte Carlo or Markov chain Monte Carlo (MCMC) algorithms, such as with the LaplacesDemon or LaplacesDemon.hpc functions. MCSE approaches zero as the number of independent posterior samples approaches infinity. MCSE is essentially a standard deviation around the posterior mean of the samples, $\mathrm{E}(\theta)$, due to uncertainty associated with using an MCMC algorithm, or Monte Carlo methods in general.

The acceptable size of the MCSE depends on the acceptable uncertainty associated around the marginal posterior mean, $\mathrm{E}(\theta)$, and the goal of inference. It has been argued that MCSE is generally unimportant when the goal of inference is θ rather than $\mathrm{E}(\theta)$ (Gelman et al., 2004, p. 277), and that a sufficient ESS is more important. Others perceive MCSE to be a vital part of reporting any Bayesian model, and as a stopping rule (Flegal et al., 2008).

In LaplacesDemon, MCSE is part of the posterior summaries because it is easy to estimate, and Laplace's Demon prefers to continue updating until each MCSE is less than 6.27% of its associated marginal posterior standard deviation (for more information on this stopping rule, see the Consort function), since MCSE has been demonstrated to be an excellent stopping rule.

Acceptable error may be specified, if known, in the MCSS (Monte Carlo Sample Size) function to estimate the required number of posterior samples.

MCSE is a univariate function that is often applied to each marginal posterior distribution. A multivariate form is not included. By chance alone due to multiple independent tests, 5% of the parameters should indicate unacceptable MSCEs, even when acceptable. Assessing convergence is difficult.

Usage

```
MCSE(x, method="IMPS", batch.size="sqrt", warn=FALSE)
MCSS(x, a)
```

Arguments

x	This is a vector of posterior samples for which MCSE or MCSS will be estimated.
a	This is a scalar argument of acceptable error for the mean of x, and a must be positive. As acceptable error decreases, the required number of samples increases.

`method`	This is an optional argument for the method of MCSE estimation, and defaults to Geyer's `"IMPS"` method. Optional methods include `"sample.variance"` and `"batch.mean"`. Note that `"batch.mean"` is recommended only when the number of posterior samples is at least 1,000.
`batch.size`	This is an optional argument that corresponds only with `method="batch.means"`, and determines either the size of the batches (accepting a numerical argument) or the method of creating the size of batches, which is either `"sqrt"` or `"cuberoot"`, and refers to the length of `x`. The default argument is `"sqrt"`.
`warn`	Logical. If `warn=TRUE`, then a warning is provided with `method="batch.means"` whenever posterior sample size is less than 1,000, or a warning is produced when more autcovariance is recommended with `method="IMPS"`.

Details

The default method for estimating MCSE is Geyer's Initial Monotone Positive Sequence (IMPS) estimator (Geyer, 1992), which takes the asymptotic variance into account and is time-series based. This method goes by other names, such as Initial Positive Sequence (IPS).

The simplest method for estimating MCSE is to modify the formula for standard error, $\sigma(\mathbf{x})/\sqrt{N}$, to account for non-independence in the sequence $\mathbf{x}$ of posterior samples. Non-independence is estimated with the `ESS` function for Effective Sample Size (see the `ESS` function for more details), where $M = ESS(\mathbf{x})$, and MCSE is $\sigma(\mathbf{x})/\sqrt{M}$. Although this is the fastest and easiest method of estimation, it does not incorporate an estimate of the asymptotic variance of $\mathbf{x}$.

The batch means method (Jones et al., 2006; Flegal et al., 2008) separates elements of $\mathbf{x}$ into batches and estimates MCSE as a function of multiple batches. This method is excellent, but is not recommended when the number of posterior samples is less than 1,000. These journal articles also assert that MCSE is a better stopping rule than MCMC convergence diagnostics.

The `MCSS` function estimates the required number of posterior samples, given the user-specified acceptable error, posterior samples `x`, and the observed variance (rather than asymptotic variance). Due to the observed variance, this is a rough estimate.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Flegal, J.M., Haran, M., and Jones, G.L. (2008). "Markov chain Monte Carlo: Can We Trust the Third Significant Figure?". *Statistical Science*, 23, p. 250–260.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). "Bayesian Data Analysis, Texts in Statistical Science, 2nd ed.". Chapman and Hall, London.

Geyer, C.J. (1992). "Practical Markov Chain Monte Carlo". *Statistical Science*, 7, 4, p. 473–483.

Jones, G.L., Haran, M., Caffo, B.S., and Neath, R. (2006). "Fixed-Width Output Analysis for Markov chain Monte Carlo". *Journal of the American Statistical Association*, 101(1), p. 1537–1547.

See Also

`Consort`, `ESS`, `LaplacesDemon`, and `LaplacesDemon.hpc`.

Examples

```
library(LaplacesDemon)
x <- rnorm(1000)
MCSE(x)
MCSE(x, method="batch.means")
MCSE(x, method="sample.variance")
MCSS(x, a=0.01)
```

`MinnesotaPrior` *Minnesota Prior*

Description

The Minnesota prior, also called the Litterman prior, is a shrinkage prior for autoregressive parameters in vector autoregressive (VAR) models. There are many variations of the Minnesota prior. This Minnesota prior is calculated as presented in Lutkepohl (2005, p. 225), and returns one or more prior covariance matrices in an array.

Usage

```
MinnesotaPrior(J, lags=c(1,2), lambda=1, theta=0.5, sigma)
```

Arguments

`J`	This is the scalar number of time-series in the VAR.
`lags`	This accepts an integer vector of lags of the autoregressive parameters. The lags are not required to be successive.
`lambda`	This accepts a scalar, positive-only hyperparameter that controls how tightly the parameter of the first lag is concentrated around zero. A smaller value results in smaller diagonal variance. When equal to zero, the posterior equals the prior and data is not influential. When equal to infinity, no shrinkage occurs and posterior expectations are closest to estimates from ordinary least squares (OLS). It has been asserted that as the number, J, of time-series increases, this hyperparameter should decrease.
`theta`	This accepts a scalar hyperparameter in the interval [0,1]. When one, off-diagonal elements have variance similar or equal to diagonal elements. When zero, off-diagonal elements have zero variance. A smaller value is associated with less off-diagonal variance.
`sigma`	This accepts a vector of length J of residual standard deviations of the dependent variables given the expectations.

Details

The Minnesota prior was introduced in Doan, Litterman, and Sims (1984) as a shrinkage prior for autoregressive parameters in vector autoregressive (VAR) models. The Minnesota prior was reviewed in Litterman (1986), and numerous variations have been presented since. This is the version of the Minnesota prior as described in Lutkepohl (2005, p. 225) for stationary time-series.

Given one or more $J \times J$ matrices of autoregressive parameters in a VAR model, the user specifies two tuning hyperparameters for the Minnesota prior: `lambda` and `theta`. Each iteration of the numerical approximation algorithm, the latest vector of residual standard deviation parameters is supplied to the `MinnesotaPrior` function, which then returns an array that contains one or more prior covariance matrices for the autoregressive parameters. Multiple prior covariance matrices are returned when multiple lags are specified. The tuning hyperparameters, `lambda` and `theta`, can be estimated from the data via hierarchical Bayes.

It is important to note that the Minnesota prior does not technically return a covariance matrix, because the matrix is not symmetric, and therefore not positive-definite. For this reason, a Minnesota prior covariance matrix should not be supplied as a covariance matrix to a multivariate normal distribution, such as with the `dmvn` function, though it would be accepted and then (incorrectly) converted to a symmetric matrix. Instead, `dnormv` should be used for element-wise evaluation.

While the Minnesota prior is used to specify the prior covariance for VAR autoregressive parameters, prior means are often all set to zero, or sometimes the first lag is set to an identity matrix.

An example is provided in the Examples vignette.

Value

This function returns a $J \times J \times L$ array for J time-series and L lags.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Doan, T., Litterman, R.B. and Sims, C.A. (1984). "Forecasting and Conditional Projection using Realistic Prior Distributions". *Econometric Reviews*, 3, p. 1–144.

Litterman, R.B. (1986). "Forecasting with Bayesian Vector Autoregressions - Five Years of Experience". *Journal of Business & Economic Statistics*, 4, p. 25–38.

Lutkepohl, H. (2005). "New Introduction to Multiple Time Series Analysis". Springer, Germany.

See Also

`dmvn`, `dnormv`, and `LaplacesDemon`.

`MISS` *Multiple Imputation Sequential Sampling*

Description

This function performs multiple imputation (MI) on a numeric matrix by sequentially sampling variables with missing values, given all other variables in the data set.

Usage

```
MISS(X, Iterations=100, Algorithm="GS", Fit=NULL, verbose=TRUE)
```

Arguments

`X`	This required argument accepts a numeric matrix of data that contains both observed and missing values. Data set **X** must not have any rows or columns that are completely missing. **X** must not have any constants. The user must apply any data transformations appropriate for these models. Missing values are assumed to be Missing At Random (MAR).
`Iterations`	This is the number of iterations to perform sequential sampling via MCMC algorithms.
`Algorithm`	The MCMC algorithm defaults to the Gibbs Sampler (GS).
`Fit`	This optional argument accepts an object of class `miss`. When supplied, `MISS` will continue where it left off, provided the user does not change the algorithm (different methods are used with different algortihms, so model parameters will not match). In short, changing algorithms requires starting from scratch.
`verbose`	Logical. When `FALSE`, only the iteration prints to the console. When `TRUE`, which is the default, both the iteration and which variable is being imputed are printed to the console.

Details

Imputation is a family of statistical methods for replacing missing values with estimates. Introduced by Rubin and Schenker (1986) and Rubin (1987), Multiple Imputation (MI) is a family of imputation methods that includes multiple estimates, and therefore includes variability of the estimates.

The Multiple Imputation Sequential Sampler (MISS) function performs MI by determining the type of variable and therefore the sampler for each variable, and then sequentially progresses through each variable in the data set that has missing values, updating its prediction of those missing values given all other variables in the data set each iteration.

MI is best performed within a model, where it is called full-likelihood imputation. Examples may be found in the "Examples" vignette. However, sometimes it is impractical to impute within a model when there are numerous missing values and a large number of parameters are therefore added. As an alternative, MI may be performed on the data set before the data is passed to the model, such as in the `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, or `VariationalBayes` function. This is less desirable, but MISS is available for MCMC-based MI in this case.

Missing values are initially set to column means for continuous variables, and are set to one for discrete variables.

MISS uses the following methods and MCMC algorithms:

Missing values of continuous variables are estimated with a ridge-stabilized linear regression Gibbs sampler.

Missing values of binary variables that have only 0 or 1 for values are estimated either with a binary robit (t-link logistic regression model) Gibbs sampler of Albert and Chib (1993).

Missing values of discrete variables with 3 or more (ordered or unordered) discrete values are considered continuous.

In the presence of big data, it is suggested that the user sequentially read in batches of data that are small enough to be manageable, and then apply the MISS function to each batch. Each batch should be representative of the whole, of course.

It is common for multiple imputation functions to handle variable transformations. MISS does not transform variables, but imputes what it gets. For example, if a user has a variable that should be positive only, then it is recommended here that the user log-transform the variable, pass the data set to MISS, and when finished, exponentiate both the observed and imputed values of that variable.

The `CenterScale` function should also be considered to speed up convergence.

It is hoped that MISS is helpful, though it is not without limitation and there are numerous alternatives outside of the `LaplacesDemon` package. If MISS does not fulfill the needs of the user, then the following packages are recommended: Amelia, mi, or mice. MISS emphasizes MCMC more than these alternatives, though MISS is not as extensive. When a data set does not have a simple structure, such as merely continuous or binary or unordered discrete, the `LaplacesDemon` function should be considered, where a user can easily specify complicated structures such as multilevel, spatial or temporal dependence, and more.

Matrix inversions are required in the Gibbs sampler. Matrix inversions become more cumbersome as the number J of variables increases.

If a large number of iterations is used, then the user may consider studying the imputations for approximate convergence with the `BMK.Diagnostic` function, by supplying the transpose of codeFit$Imp. In the presence of numerous missing values, say more than 100, the user may consider iterating through the study of the imputations of 100 missing values at a time.

Value

This function returns an object of class `miss` that is a list with five components:

`Algorithm`	This indicates which algorithm was selected.
`Imp`	This is a $M \times T$ matrix of M missing values and T iterations that contains imputations.
`parm`	This is a list of length J for J variables, and each component of the list contains parameters associated with the prediction of missing values for that variable.
`PostMode`	This is a vector of posterior modes. If the user intends to replace missing values in a data set with only one estimate per missing values (single, not multiple imputation), then this vector contains these values.
`Type`	This is a vector of length J for J variables that indicates the type of each variable, as MISS will consider it. When `Type=1`, the variable is considered to be continuous. When `Type=2`, only two discrete values (0 and 1) were found.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Albert, J.H. and Chib, S. (1993). "Bayesian Analysis of Binary and Polychotomous Response Data". *Journal of the American Statistical Association*, 88(422), p. 669–679.

Rubin, D.B. (1987). "Multiple Imputation for Nonresponse in Surveys". John Wiley and Sons: New York, NY.

Rubin, D.B. and Schenker, N. (1986). "Multiple Imputation for Interval Estimation from Simple Random Samples with Ignorable Nonresponse". *Journal of the American Statistical Association*, 81, p. 366–374.

See Also

ABB, BMK.Diagnostic, CenterScale, IterativeQuadrature LaplaceApproximation, LaplacesDemon, and VariationalBayes.

Examples

```
#library(LaplacesDemon)
### Create Data
#N <- 20 #Number of Simulated Records
#J <- 5 #Number of Simulated Variables
#pM <- 0.25 #Percent Missing
#Sigma <- as.positive.definite(matrix(runif(J*J),J,J))
#X <- rmvn(N, rep(0,J), Sigma)
#m <- sample.int(N*J, round(pM*N*J))
#X[m] <- NA
#head(X)

### Begin Multiple Imputation
#Fit <- MISS(X, Iterations=100, Algorithm="GS", verbose=TRUE)
#Fit
#summary(Fit)
#plot(Fit)
#plot(BMK.Diagnostic(t(Fit$Imp)))

### Continue Updating if Necessary
#Fit <- MISS(X, Iterations=100, Algorithm="GS", Fit, verbose=TRUE)
#summary(Fit)
#plot(Fit)
#plot(BMK.Diagnostic(t(Fit$Imp)))

### Replace Missing Values in Data Set with Posterior Modes
#Ximp <- X
#Ximp[which(is.na(X))] <- Fit$PostMode

### Original and Imputed Data Sets
#head(X)
```

```
#head(Ximp)
#summary(X)
#summary(Ximp)

### or Multiple Data Sets, say 3
#Ximp <- array(X, dim=c(nrow(X), ncol(X), 3))
#for (i in 1:3) {
#     Xi <- X
#     Xi[which(is.na(X))] <- Fit$Imp[,sample.int(ncol(Fit$Imp), 1)]
#     Ximp[,,i] <- Xi}
#head(X)
#head(Ximp[,,1])
#head(Ximp[,,2])
#head(Ximp[,,3])

#End
```

Mode *The Mode(s) of a Vector*

Description

The mode is a measure of central tendency. It is the value that occurs most frequently, or in a continuous probability distribution, it is the value with the most density. A distribution may have no modes (such as with a constant, or in a uniform distribution when no value occurs more frequently than any other), or one or more modes.

Usage

```
is.amodal(x, min.size=0.1)
is.bimodal(x, min.size=0.1)
is.multimodal(x, min.size=0.1)
is.trimodal(x, min.size=0.1)
is.unimodal(x, min.size=0.1)
Mode(x)
Modes(x, min.size=0.1)
```

Arguments

x	This is a vector in which a mode (or modes) will be sought.
min.size	This is the minimum size that can be considered a mode, where size means the proportion of the distribution between areas of increasing kernel density estimates.

Details

The is.amodal function is a logical test of whether or not x has a mode. If x has a mode, then TRUE is returned, otherwise FALSE.

The is.bimodal function is a logical test of whether or not x has two modes. If x has two modes, then TRUE is returned, otherwise FALSE.

The is.multimodal function is a logical test of whether or not x has multiple modes. If x has multiple modes, then TRUE is returned, otherwise FALSE.

The is.trimodal function is a logical test of whether or not x has three modes. If x has three modes, then TRUE is returned, otherwise FALSE.

The is.unimodal function is a logical test of whether or not x has one mode. If x has one mode, then TRUE is returned, otherwise FALSE.

The Mode function returns the most frequent value when x is discrete. If x is a constant, then it is considered amodal, and NA is returned. If multiple modes exist, this function returns only the mode with the highest density, or if two or more modes have the same density, then it returns the first mode found. Otherwise, the Mode function returns the value of x associated with the highest kernel density estimate, or the first one found if multiple modes have the same density.

The Modes function is a simple, deterministic function that differences the kernel density of x and reports a number of modes equal to half the number of changes in direction, although the min.size function can be used to reduce the number of modes returned, and defaults to 0.1, eliminating modes that do not have at least 10% of the distributional area. The Modes function returns a list with three components: modes, modes.dens, and size. The elements in each component are ordered according to the decreasing density of the modes. The modes component is a vector of the values of x associated with the modes. The modes.dens component is a vector of the kernel density estimates at the modes. The size component is a vector of the proportion of area underneath each mode.

The IterativeQuadrature, LaplaceApproximation, and VariationalBayes functions characterize the marginal posterior distributions by posterior modes (means) and variance. A related topic is MAP or maximum *a posteriori* estimation.

Otherwise, the results of Bayesian inference tend to report the posterior mean or median, along with probability intervals (see p.interval and LPL.interval), rather than posterior modes. In many types of models, such as mixture models, the posterior may be multimodal. In such a case, the usual recommendation is to choose the highest mode if feasible and possible. However, the highest mode may be uncharacteristic of the majority of the posterior.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

IterativeQuadrature, LaplaceApproximation, LaplacesDemon, LPL.interval, p.interval, and VariationalBayes.

Examples

```
library(LaplacesDemon)
### Below are distributions with different numbers of modes.
```

```
x <- c(1,1) #Amodal
x <- c(1,2,2,2,3) #Unimodal
x <- c(1,2) #Bimodal
x <- c(1,3,3,3,3,4,4,4,4,4) #min.size affects the answer
x <- c(1,1,3,3,3,3,4,4,4,4,4) #Trimodal

### And for each of the above, the functions below may be applied.
Mode(x)
Modes(x)
is.amodal(x)
is.bimodal(x)
is.multimodal(x)
is.trimodal(x)
is.unimodal(x)
```

`Model.Specification.Time`
Model Specification Time

Description

The `Model.Spec.Time` function returns the time in minutes to evaluate a model specification a number of times, as well as the evaluations per minute, and componentwise iterations per minute.

Usage

```
Model.Spec.Time(Model, Initial.Values, Data, n=1000)
```

Arguments

`Model`	This requried argument is a model specification function. For more information, see `LaplacesDemon`.
`Initial.Values`	This required argument is a vector of initial values for the parameters.
`Data`	This required argument is a list of data. For more information, see `LaplacesDemon`.
`n`	This is the number of evaluations of the model specification, and accuracy increases with `n`.

Details

The largest single factor to affect the run-time of an algorithm – whether it is with `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, or `VariationalBayes` – is the time that it takes to evaluate the model specification. This has also been observed in Rosenthal (2007). It is highly recommended that a user of the `LaplacesDemon` package should attempt to reduce the run-time of the model specification, usually by testing alternate forms of code for speed. This is especially true with big data, such as with the `BigData` function.

Every function in the LaplacesDemon package is byte-compiled, which is a recent option in R. This reduces run-time, thanks to Tierney's compiler package in base R. The model specification,

however, is specified by the user, and should be byte-compiled. The reduction in run-time may range from mild to dramatic, depending on the model. It is highly recommended that users concerned with run-time activate the compiler package and use the `cmpfun` function, as per the example below.

A model specification function that is optimized for speed and involves many records may result in a model update run-time that is considerably less than other popular MCMC-based software algorithms that loop through records, even when those algorithms are coded in `C` or other fast languages. For a comparison, see the "Laplace's Demon Tutorial" vignette.

However, if a model specification function in the LaplacesDemon package is not fully vectorized (contains `for` loops and `apply` functions), then run-time will typically be slower than other popular MCMC-based software algorithms.

The speed of calculating the model specification function is affected by parameter constraints, such as with the `interval` function. Parameter constraints may add considerable variability to the speed of this calculation, and usually more variation occurs with initial values that are far from the target distributions.

Distributions in the `LaplacesDemon` package usually have logical checks to ensure correctness. These checks may slow the calculation of the density, for example. If the user is confident these checks are unnecessary for their model, then the user may copy the code to a new function name and comment-out the checks to improve speed.

When speed is of paramount importance, a user is encouraged to experiment with writing the model specification function in another language such as in `C++` with the `Rcpp` package, and calling drop-in replacement functions from within the `Model` function, or re-writing the model function entirely in `C++`. For an introduction to including `C++` in **LaplacesDemon**, see `https://web.archive.org/web/20150227225556/http://www.bayesian-inference.com/softwarearticlescppsugar`.

When a model specification function is computationally expensive, another approach to reduce run-time may be for the user to write parallelized code within the model, splitting up difficult tasks and assigning each to a separate CPU.

Another use for `Model.Spec.Time` is to allow the user to make an informed decision about which MCMC algorithm to select, given the speed of their model specification. For example, the Adaptive Metropolis-within-Gibbs (AMWG) of Roberts and Rosenthal (2009) is currently the adaptive MCMC algorithm of choice in general in many cases, but this choice is conditional on run-time. While other MCMC algorithms in `LaplacesDemon` evaluate the model specification function once per iteration, componentwise algorithms such as in the MWG family evaluate it once per parameter per iteration, significantly increasing run-time per iteration in large models. The `Model.Spec.Time` function may forewarn the user of the associated run-time, and if it should be decided to go with an alternate MCMC algorithm, such as AMM, in which each element of its covariance matrix must stabilize for the chains to become stationary. AMM, for example, will require many more iterations of burn-in (for more information, see the `burnin` function), but with numerous iterations, allows more thinning. A general recommendation may be to use AMWG when `Componentwise.Iters.per.Minute` >= 1000, but this is subjective and best determined by each user for each model. A better decision may be made by comparing MCMC algorithms with the `Juxtapose` function for a particular model.

Following are a few common suggestions for increasing the speed of `R` code in the model specification function. There are often exceptions to these suggestions, so case-by-case experimentation is also suggested.

- Replace exponents with multiplied terms, such as `x^2` with `x*x`.

- Replace `mean(x)` with `sum(x)/length(x)`.
- Replace parentheses (when possible) with curly brackets, such as `x*(a+b)` with `x*{a+b}`.
- Replace `tcrossprod(Data$X, t(beta))` with `Data$X %*% beta` when there are few predictors, and avoid `tcrossprod(beta, Data$X)`, which is always slowest.
- Vectorize functions and eliminate `apply` and `for` functions. There are often specialized functions. For example, `max.col(X)` is faster than `apply(X, 1, which.max)`.

When seeking speed, things to consider beyond the LaplacesDemon package are the Basic Linear Algebra System (BLAS) and hardware. The ATLAS (Automatically Tuned Linear Algebra System) should be worth installing (and there are several alternatives), but this discussion is beyond the scope of this documentation. Lastly, the speed at which the computer can process iterations is limited by its hardware, and more should be considered than merely the CPU (Central Processing Unit). Again, though, this is beyond the scope of this documentation.

Value

The `Model.Spec.Time` function returns a list with three components:

`Time` This is the time in minutes to evaluate the model specification `n` times.

`Evals.per.Minute`
This is the number of evaluations of the model specification per minute: `n / Time`. This is also the number of iterations per minute in an algorithm that is not componentwise, where one evaluation occurs per iteration.

`Componentwise.Iters.per.Minute`
This is the number of iterations per minute in a componentwise MCMC algorithm, such as AMWG or MWG. It is the evaluations per minute divided by the number of parameters, since an evaluation must occur for each parameter, for each iteration.

Author(s)

Statisticat, LLC.

References

Roberts, G.O. and Rosenthal, J.S. (2009). "Examples of Adaptive MCMC". *Computational Statistics and Data Analysis*, 18, p. 349–367.

See Also

`.C`, `.Fortran`,

`apply`, `BigData`, `interval`, `IterativeQuadrature`, `Juxtapose`, `LaplaceApproximation`, `LaplacesDemon`, `max.col`, `PMC`, `system.time`, and `VariationalBayes`.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#####################  Load the LaplacesDemon Library  #####################
library(LaplacesDemon)

##############################  Demon Data  ##############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) {X[,j] <- CenterScale(X[,j])}

########################  Data List Preparation  ########################
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) return(c(rnormv(Data$J,0,10), rhalfcauchy(1,5)))
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

########################  Model Specification  ########################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log of Prior Densities
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

set.seed(666)

#########################  Initial Values  #########################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)

#########################  Model.Spec.Time  #########################
### Not byte-compiled
Model.Spec.Time(Model, Initial.Values, MyData)
### Byte-compiled
library(compiler)
Model <- cmpfun(Model)
```

```
Model.Spec.Time(Model, Initial.Values, MyData)
```

`p.interval` *Probability Interval*

Description

This function returns one or more probability intervals of posterior samples.

Usage

```
p.interval(obj, HPD=TRUE, MM=TRUE, prob=0.95, plot=FALSE, PDF=FALSE, ...)
```

Arguments

`obj`	This can be either a vector or matrix of posterior samples, or an object of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb`. If it is an object of class `demonoid`, then it will use only stationary posterior samples and monitored target distributions (automatically discarding the burn-in; if stationarity does not exist, then it will use all samples).
`HPD`	Logical. This argument defaults to `TRUE`, in which case one or more Highest Posterior Density (HPD) intervals is returned. When `FALSE`, one or more quantile-based probability intervals is returned.
`MM`	Logical. This argument defaults to `TRUE`, in which case each column vector is checked for multimodality, and if found, the multimodal form of a Highest Posterior Density (HPD) interval is additionally estimated, even when `HPD=FALSE`.
`prob`	This is a numeric scalar in the interval (0,1) giving the target probability interval, and defaults to 0.95, representing a 95% probability interval. A 95% probability interval, for example, is an interval that contains 95% of a posterior probability distribution.
`plot`	Logical. When `plot=TRUE`, each kernel density is plotted and shaded gray, and the area under the curve within the probability interval is shaded black. If the kernel density is considered to be multimodal, then up to three intervals are shaded black. A vertical, red, dotted line is added at zero. The `plot` argument defaults to `FALSE`.
`PDF`	Logical. When `PDF=TRUE`, and only when `plot=TRUE`, plots are saved as a .pdf file in the working directory.
`...`	Additional arguments are unused.

Details

A probability interval, also called a credible interval or Bayesian confidence interval, is an interval in the domain of a posterior probability distribution. When generalized to multivariate forms, it is called a probability region (or credible region), though some sources refer to a probability region (or credible region) as the area within the probability interval. Bivariate probability regions may be plotted with the `joint.pr.plot` function.

The p.interval function may return different probability intervals: a quantile-based probability interval, a unimodal Highest Posterior Density (HPD) interval, and multimodal HPD intervals. Another type of probability interval is the Lowest Posterior Loss (LPL) interval, which is calculated with the LPL.interval function.

The quantile-based probability interval is used most commonly, possibly because it is simple, the fastest to calculate, invariant under transformation, and more closely resembles the frequentist confidence interval. The lower and upper bounds of the quantile-based probability interval are calculated with the quantile function. A 95% quantile-based probability interval reports the values of the posterior probability distribution that indicate the 2.5% and 97.5% quantiles, which contain the central 95% of the distribution. The quantile-based probability interval is centered around the median and has equal-sized tails.

The HPD (highest posterior density) interval is identical to the quantile-based probability interval when the posterior probability distribution is unimodal and symmetric. Otherwise, the HPD interval is the smallest interval, because it is estimated as the interval that contains the highest posterior density. Unlike the quantile-based probability interval, the HPD interval could be one-tailed or two-tailed, whichever is more appropriate. However, unlike the quantile-based interval, the HPD interval is not invariant to reparameterization (Bernardo, 2005).

The unimodal HPD interval is estimated from the empirical CDF of the sample for each parameter (or deviance or monitored variable) as the shortest interval for which the difference in the ECDF values of the end-points is the user-specified probability width. This assumes the distribution is not severely multimodal.

As an example, imagine an exponential posterior distribution. A quantile-based probability interval would report the highest density region near zero to be outside of its interval. In contrast, the unimodal HPD interval is recommended for such skewed posterior distributions.

When MM=TRUE, the is.multimodal function is applied to each column vector after the unimodal interval (either quantile-based or HPD) is estimated. If multimodality is found, then multimodal HPD intervals are estimated with kernel density and printed to the screen as a character string. The original unimodal intervals are returned in the output matrix, because the matrix is constrained to have a uniform number of columns per row, and because multimodal HPD intervals may be disjoint.

Disjoint multimodal HPD intervals have multiple intervals for one posterior probability distribution. An example may be when there is a bimodal, Gaussian distribution with means -10 and 10, variances of 1 and 1, and a 95% probability interval is specified. In this case, there is not enough density between these two distant modes to have only one probability interval.

The user should also consider LPL.interval, since it is invariant to reparameterization like the quantile-based probability interval, but could be one- or two-tailed, whichever is more appropriate, like the HPD interval. A comparison of the quantile-based probability interval, HPD interval, and LPL interval is available here: https://web.archive.org/web/20150214090353/http://www.bayesian-inference.com/credible.

Value

A matrix is returned with rows corresponding to the parameters (or deviance or monitored variables), and columns "Lower" and "Upper". The elements of the matrix are the unimodal probability intervals. The attribute "Probability" is the user-selected probability width. If MM=TRUE and multimodal posterior distributions are found, then multimodal HPD intervals are printed to the screen in a character string.

Author(s)

Statisticat, LLC

References

Bernardo, J.M. (2005). "Intrinsic Credible Regions: An Objective Bayesian Approach to Interval Estimation". *Sociedad de Estadistica e Investigacion Operativa*, 14(2), p. 317–384.

See Also

`is.multimodal`, `IterativeQuadrature`, `joint.pr.plot`, `LaplaceApproximation`, `LaplacesDemon`, `LPL.interval`, `PMC`, and `VariationalBayes`.

Examples

```
##First, update the model with the LaplacesDemon function.
##Then
#p.interval(Fit, HPD=TRUE, MM=TRUE, prob=0.95)
```

`plot.bmk` *Plot Hellinger Distances*

Description

This function plots Hellinger distances in an object of class `bmk`.

Usage

```
## S3 method for class 'bmk'
plot(x, col=colorRampPalette(c("black","red"))(100),
     title="", PDF=FALSE, Parms=NULL, ...)
```

Arguments

`x`	This required argument is an object of class `bmk`. See the `BMK.Diagnostic` function for more information.
`col`	This argument specifies the colors of the cells. By default, the `colorRampPalette` function colors large Hellinger distances as `red`, small as `black`, and provides 100 color gradations.
`title`	This argument specifies the title of the plot, and the default does not include a title.
`PDF`	Logical. When `TRUE`, the plot is saved as a .pdf file.

Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
...	Additional arguments are unused.

Details

The plot.bmk function plots the Hellinger distances in an object of class bmk. This is useful for quickly finding portions of chains with large Hellinger distances, which indicates non-stationarity and non-convergence.

See Also

BMK.Diagnostic

Examples

```
library(LaplacesDemon)
N <- 1000 #Number of posterior samples
J <- 10 #Number of parameters
Theta <- matrix(runif(N*J),N,J)
colnames(Theta) <- paste("beta[", 1:J, "]", sep="")
for (i in 2:N) {Theta[i,1] <- Theta[i-1,1] + rnorm(1)}
HD <- BMK.Diagnostic(Theta, batches=10)
plot(HD, title="Hellinger distance between batches")
```

---

plot.demonoid *Plot samples from the output of Laplace's Demon*

Description

This may be used to plot, or save plots of, samples in an object of class demonoid or demonoid.hpc. Plots include a trace plot, density plot, autocorrelation or ACF plot, and if an adaptive algorithm was used, the absolute difference in the proposal variance, or the value of epsilon, across adaptations.

Usage

```
## S3 method for class 'demonoid'
plot(x, BurnIn=0, Data, PDF=FALSE, Parms, FileName, ...)
## S3 method for class 'demonoid.hpc'
plot(x, BurnIn=0, Data, PDF=FALSE, Parms, FileName, ...)
```

Arguments

x	This required argument is an object of class demonoid or demonoid.hpc.
BurnIn	This argument requires zero or a positive integer that indicates the number of thinned samples to discard as burn-in for the purposes of plotting. For more information on burn-in, see burnin.
Data	This required argument must receive the list of data that was supplied to LaplacesDemon to create the object of class demonoid.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
FileName	This argument accepts a string and save the plot under the specified name. If PDF=FALSE this argument in unused. By default, FileName = paste0("laplacesDemon-plot_", format(Sys.time(), "yyyy-mm-dd_h:m:s"), ".pdf")
...	Additional arguments are unused.

Details

The plots are arranged in a 3×3 matrix. Each row represents a parameter, the deviance, or a monitored variable. The left column displays trace plots, the middle column displays kernel density plots, and the right column displays autocorrelation (ACF) plots.

Trace plots show the thinned history of the chain or Markov chain, with its value in the y-axis moving by thinned sample across the x-axis. A chain or Markov chain with good properties does not suggest a trend upward or downward as it progresses across the x-axis (it should appear stationary), and it should mix well, meaning it should appear as though random samples are being taken each time from the same target distribution. Visual inspection of a trace plot cannot verify convergence, but apparent non-stationarity or poor mixing can certainly suggest non-convergence. A red, smoothed line also appears to aid visual inspection.

Kernel density plots depict the marginal posterior distribution. Although there is no distributional assumption about this density, kernel density estimation uses Gaussian basis functions.

Autocorrelation plots show the autocorrelation or serial correlation between values of thinned samples at nearby thinned samples. Samples with autocorrelation do not violate any assumption, but are inefficient because they reduce the effective sample size (ESS), and indicate that the chain is not mixing well, since each value is influenced by values that are previous and nearby. The x-axis indicates lags with respect to thinned samples, and the y-axis represents autocorrelation. The ideal autocorrelation plot shows perfect correlation at zero lag, and quickly falls to zero autocorrelation for all other lags.

If an adaptive algorithm was used, then the distribution of absolute differences in the proposal variances, or the value of epsilon, is plotted across adaptations. The proposal variance, or epsilon,

should change less as the adaptive algorithm approaches the target distributions. The absolute differences in the proposal variance plot should approach zero. This is called the condition of diminishing adaptation. If it is not approaching zero, then consider using a different adaptive MCMC algorithm. The following quantiles are plotted for absolute changes proposal variance: 0.025, 0.500, and 0.975.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

See Also

burnin, ESS, LaplacesDemon, and LaplacesDemon.hpc.

Examples

```
### See the LaplacesDemon function for an example.
```

---

plot.demonoid.ppc *Plots of Posterior Predictive Checks*

Description

This may be used to plot, or save plots of, samples in an object of class demonoid.ppc. A variety of plots is provided.

Usage

```
## S3 method for class 'demonoid.ppc'
plot(x, Style=NULL, Data=NULL, Rows=NULL,
     PDF=FALSE, ...)
```

Arguments

x	This required argument is an object of class demonoid.ppc.
Style	This optional argument specifies one of several styles of plots, and defaults to NULL (which is the same as "Density"). Styles of plots are indicated in quotes. Optional styles include "Covariates", "Covariates, Categorical DV", "Density", "DW", "DW, Multivariate, C", "ECDF", "Fitted", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera", "Jarque-Bera, Multivariate, C", "Mardia", "Predictive Quantiles", "Residual Density", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R". Details are given below.

Data	This optional argument accepts the data set used when updating the model. Data is required only with certain plot styles, including "Covariates", "Covariates, Categorical DV", "DW, Multivariate, C", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera, Multivariate, C", "Mardia", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R".
Rows	This optional argument is for a vector of row numbers that specify the records associated by row in the object of class demonoid.ppc. Only these rows are plotted. The default is to plot all rows. Some plots do not allow rows to be specified.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
...	Additional arguments are unused.

Details

This function can be used to produce a variety of posterior predictive plots, and the style of plot is selected with the Style argument. Below are some notes on the styles of plots.

Covariates requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is not categorical.

Covariates, Categorical DV requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is categorical.

Density plots show the kernel density of the posterior predictive distribution for each selected row of y (all are selected by default). A vertical red line indicates the position of the observed y along the x-axis. When the vertical red line is close to the middle of a normal posterior predictive distribution, then there is little discrepancy between y and the posterior predictive distribution. When the vertical red line is in the tail of the distribution, or outside of the kernel density altogether, then there is a large discrepancy between y and the posterior predictive distribution. Large discrepancies may be considered outliers, and moreover suggest that an improvement in model fit should be considered.

DW plots the distributions of the Durbin-Watson (DW) test statistics (Durbin and Watson, 1950), both observed (d^{obs} as a transparent, black density) and replicated (d^{rep} as a transparent, red density). The distribution of d^{obs} is estimated from the model, and d^{rep} is simulated from normal residuals without autocorrelation, where the number of simulations are the same as the observed number. This DW test may be applied to the residuals of univariate time-series models (or otherwise ordered residuals) to detect first-order autocorrelation. Autocorrelated residuals are not independent. The DW test is applicable only when the residuals are normally-distributed, higher-order autocorrelation is not present, and y is not used also as a lagged predictor. The DW test statistic, d^{obs}, occurs in the interval (0,4), where 0 is perfect positive autocorrelation, 2 is no autocorrelation, and 4 is perfect negative autocorrelation. The following summary is reported on the plot: the mean of d^{obs} (and its 95% probability interval), the probability that $d^{obs} > d^{rep}$, and whether or not autocorrelation is found. Positive autocorrelation is reported when the observed process is greater than the replicated process in 2.5% of the samples, and negative autocorrelation is reported when the observed process is greater than the replicated process in 97.5% of the samples.

`DW, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Durbin-Watson test, as in `DW` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for first-order autocorrelation.

`ECDF` (Empirical Cumulative Distribution Function) plots compare the ECDF of y with three ECDFs of yhat based on the 2.5%, 50% (median), and 97.5% of its distribution. The ECDF(y) is defined as the proportion of values less than or equal to y. This plot is appropriate when y is univariate and at least ordinal.

`Fitted` plots compare y with the probability interval of its replicate, and provide loess smoothing. This plot is appropriate when y is univariate and not categorical.

`Fitted, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each column-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen column-wise.

`Fitted, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each row-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen row-wise.

`Jarque-Bera` plots the distributions of the Jarque-Bera (JB) test statistics (Jarque and Bera, 1980), both observed (JB^{obs} as a transparent black density) and replicated (JB^{rep} as a transparent red density). The distribution of JB^{obs} is estimated from the model, and JB^{rep} is simulated from normal residuals, where the number of simulations are the same as the observed number. This Jarque-Bera test may be applied to the residuals of univariate models to test for normality. The Jarque-Bera test does not test normality per se, but whether or not the distribution has kurtosis and skewness that match a normal distribution, and is therefore a test of the moments of a normal distribution. The following summary is reported on the plot: the mean of JB^{obs} (and its 95% probability interval), the probability that $JB^{obs} > JB^{rep}$, and whether or not normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples.

`Jarque-Bera, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Jarque-Bera test, as in `Jarque-Bera` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for normality.

`Mardia` plots the distributions of the skewness (K3) and kurtosis (K4) test statistics (Mardia, 1970), both observed ($K3^{obs}$ and $K4^{obs}$ as transparent black density) and replicated ($K3^{rep}$ and $K4^{rep}$ as transparent red density). The distributions of $K3^{obs}$ and $K4^{obs}$ are estimated from the model, and both $K3^{rep}$ $K4^{rep}$ are simulated from multivariate normal residuals, where the number of simulations are the same as the observed number. This Mardia's test may be applied to the residuals of multivariate models to test for multivariate normality. Mardia's test does not test for multivariate normality per se, but whether or not the distribution has kurtosis and skewness that match a multivariate normal distribution, and is therefore a test of the moments of a multivariate normal distribution. The following summary is reported on the plots: the means of $K3^{obs}$ and $K4^{obs}$ (and the associated 95% probability intervals), the probabilities that $K3^{obs} > K3^{rep}$ and $K4^{obs} > K4^{rep}$, and whether or not multivariate normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples. `Mardia`

requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. Y must be a $N \times P$ matrix of N records and P variables. Source code was modified from the deprecated package QRMlib.

Predictive Quantiles plots compare y with the predictive quantile (PQ) of its replicate. This may be useful in looking for patterns with outliers. Instances outside of the gray lines are considered outliers.

Residual Density plots the residual density of the median of the samples. A vertical red line occurs at zero. This plot may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when y is univariate and continuous.

Residual Density, Multivariate C requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are column-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen column-wise.

Residual Density, Multivariate R requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are row-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen row-wise.

Residuals plots compare y with its residuals. The probability interval is plotted as a line. This plot is appropriate when y is univariate.

Residuals, Multivariate, C requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are plots of each column-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen column-wise.

Residuals, Multivariate, R requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are plots of each row-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen row-wise.

Space-Time by Space requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude, longitude, S, and T. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one time-series plot per point s in space, for a total of S plots. Therefore, these are time-series plots for each point s in space across T time-periods. See Time-Series plots below.

Space-Time by Time requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude, longitude, S, and T. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one spatial plot per time-period, and T plots will be produced. See Spatial plots below.

Spatial requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude and longitude. This spatial plot shows yrep plotted according to its coordinates, and is color-coded so that higher values of yrep become more red, and lower values become more yellow.

Spatial Uncertainty requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude and longitude. This spatial plot shows

the probability interval of yrep plotted according to its coordinates, and is color-coded so that wider probability intervals become more red, and lower values become more yellow.

`Time-Series` plots compare y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is univariate and ordered by time.

`Time-Series, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by column in Y.

`Time-Series, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each row-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by row in Y, such as is typically true in panel models.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Durbin, J., and Watson, G.S. (1950). "Testing for Serial Correlation in Least Squares Regression, I." *Biometrika*, 37, p. 409–428.

Jarque, C.M. and Bera, A.K. (1980). "Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals". *Economics Letters*, 6(3), p. 255–259.

Mardia, K.V. (1970). "Measures of Multivariate Skewness and Kurtosis with Applications". *Biometrika*, 57(3), p. 519–530.

See Also

`LaplacesDemon` and `predict.demonoid`.

Examples

```
### See the LaplacesDemon function for an example.
```

`plot.importance` *Plot Variable Importance*

Description

This may be used to plot variable importance with BPIC, predictive concordance, a discrepancy statistic, or the L-criterion regarding an object of class `importance`.

Usage

```
## S3 method for class 'importance'
plot(x, Style="BPIC", ...)
```

Arguments

x	This required argument is an object of class importance.
Style	When Style="BPIC", BPIC is shown, and BPIC is the default. Otherwise, predictive concordance is plotted when Style="Concordance", a discrepancy statistic is plotted when Style="Discrep", or the L-criterion is plotted when Style="L-criterion".
...	Additional arguments are unused.

Details

The x-axis is either BPIC (Ando, 2007), predictive concordance (Gelfand, 1996), a discrepancy statistic (Gelman et al., 1996), or the L-criterion (Laud and Ibrahim, 1995) of the Importance function (depending on the Style argument), and variables are on the y-axis. A more important variable is associated with a dot that is plotted farther to the right. For more information on variable importance, see the Importance function.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

See Also

Importance

---

plot.iterquad *Plot the output of* IterativeQuadrature

Description

This may be used to plot, or save plots of, the iterated history of the parameters and, if posterior samples were taken, density plots of parameters and monitors in an object of class iterquad.

Usage

```
## S3 method for class 'iterquad'
plot(x, Data, PDF=FALSE, Parms, ...)
```

Arguments

x	This required argument is an object of class iterquad.
Data	This required argument must receive the list of data that was supplied to IterativeQuadrature to create the object of class iterquad.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
...	Additional arguments are unused.

Details

The plots are arranged in a 2×2 matrix. The purpose of the iterated history plots is to show how the value of each parameter and the deviance changed by iteration as the IterativeQuadrature attempted to fit a normal distribution to the marginal posterior distributions.

The plots on the right show several densities, described below.

- The transparent black density is the normalized quadrature weights for non-standard normal distributions, M. For multivariate quadrature, there are often multiple weights at a given node, and the average M is shown. Vertical black lines indicate the nodes.
- The transparent red density is the normalized LP weights. For multivariate quadrature, there are often multiple weights at a given node, and the average normalized and weighted LP is shown. Vertical red lines indicate the nodes.
- The transparent green density is the normal density implied given the conditional mean and conditional variance.
- The transparent blue density is the kernel density estimate of posterior samples generated with Sampling Importance Resampling. This is plotted only if the algorithm converged, and if sir=TRUE.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

IterativeQuadrature

Examples

```
### See the IterativeQuadrature function for an example.
```

`plot.iterquad.ppc` *Plots of Posterior Predictive Checks*

Description

This may be used to plot, or save plots of, samples in an object of class `iterquad.ppc`. A variety of plots is provided.

Usage

```
## S3 method for class 'iterquad.ppc'
plot(x, Style=NULL, Data=NULL,  Rows=NULL,
     PDF=FALSE, ...)
```

Arguments

`x`	This required argument is an object of class `iterquad.ppc`.
`Style`	This optional argument specifies one of several styles of plots, and defaults to `NULL` (which is the same as `"Density"`). Styles of plots are indicated in quotes. Optional styles include `"Covariates"`, `"Covariates, Categorical DV"`, `"Density"`, `"DW"`, `"DW, Multivariate, C"`, `"ECDF"`, `"Fitted"`, `"Fitted, Multivariate, C"`, `"Fitted, Multivariate, R"`, `"Jarque-Bera"`, `"Jarque-Bera, Multivariate, C"`, `"Mardia"`, `"Predictive Quantiles"`, `"Residual Density"`, `"Residual Density, Multivariate, C"`, `"Residual Density, Multivariate, R"`, `"Residuals"`, `"Residuals, Multivariate, C"`, `"Residuals, Multivariate, R"`, `"Space-Time by Space"`, `"Space-Time by Time"`, `"Spatial"`, `"Spatial Uncertainty"`, `"Time-Series"`, `"Time-Series, Multivariate, C"`, and `"Time-Series, Multivariate, R"`. Details are given below.
`Data`	This optional argument accepts the data set used when updating the model. Data is required only with certain plot styles, including `"Covariates"`, `"Covariates, Categorical DV"`, `"DW, Multivariate, C"`, `"Fitted, Multivariate, C"`, `"Fitted, Multivariate, R"`, `"Jarque-Bera, Multivariate, C"`, `"Mardia"`, `"Residual Density, Multivariate, C"`, `"Residual Density, Multivariate, R"`, `"Residuals, Multivariate, C"`, `"Residuals, Multivariate, R"`, `"Space-Time by Space"`, `"Space-Time by Time"`, `"Spatial"`, `"Spatial Uncertainty"`, `"Time-Series, Multivariate, C"`, and `"Time-Series, Multivariate, R"`.
`Rows`	This optional argument is for a vector of row numbers that specify the records associated by row in the object of class `iterquad.ppc`. Only these rows are plotted. The default is to plot all rows. Some plots do not allow rows to be specified.
`PDF`	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
`...`	Additional arguments are unused.

Details

This function can be used to produce a variety of posterior predictive plots, and the style of plot is selected with the `Style` argument. Below are some notes on the styles of plots.

`Covariates` requires `Data` to be specified, and also requires that the covariates are named `X` or `x`. A plot is produced for each covariate column vector against yhat, and is appropriate when y is not categorical.

`Covariates, Categorical DV` requires `Data` to be specified, and also requires that the covariates are named `X` or `x`. A plot is produced for each covariate column vector against yhat, and is appropriate when y is categorical.

`Density` plots show the kernel density of the posterior predictive distribution for each selected row of y (all are selected by default). A vertical red line indicates the position of the observed y along the x-axis. When the vertical red line is close to the middle of a normal posterior predictive distribution, then there is little discrepancy between y and the posterior predictive distribution. When the vertical red line is in the tail of the distribution, or outside of the kernel density altogether, then there is a large discrepancy between y and the posterior predictive distribution. Large discrepancies may be considered outliers, and moreover suggest that an improvement in model fit should be considered.

`DW` plots the distributions of the Durbin-Watson (DW) test statistics (Durbin and Watson, 1950), both observed (d^{obs} as a transparent, black density) and replicated (d^{rep} as a transparent, red density). The distribution of d^{obs} is estimated from the model, and d^{rep} is simulated from normal residuals without autocorrelation, where the number of simulations are the same as the observed number. This DW test may be applied to the residuals of univariate time-series models (or otherwise ordered residuals) to detect first-order autocorrelation. Autocorrelated residuals are not independent. The DW test is applicable only when the residuals are normally-distributed, higher-order autocorrelation is not present, and y is not used also as a lagged predictor. The DW test statistic, d^{obs}, occurs in the interval (0,4), where 0 is perfect positive autocorrelation, 2 is no autocorrelation, and 4 is perfect negative autocorrelation. The following summary is reported on the plot: the mean of d^{obs} (and its 95% probability interval), the probability that $d^{obs} > d^{rep}$, and whether or not autocorrelation is found. Positive autocorrelation is reported when the observed process is greater than the replicated process in 2.5% of the samples, and negative autocorrelation is reported when the observed process is greater than the replicated process in 97.5% of the samples.

`DW, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Durbin-Watson test, as in `DW` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for first-order autocorrelation.

`ECDF` (Empirical Cumulative Distribution Function) plots compare the ECDF of y with three ECDFs of yhat based on the 2.5%, 50% (median), and 97.5% of its distribution. The ECDF(y) is defined as the proportion of values less than or equal to y. This plot is appropriate when y is univariate and at least ordinal.

`Fitted` plots compare y with the probability interval of its replicate, and provide loess smoothing. This plot is appropriate when y is univariate and not categorical.

`Fitted, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each column-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen column-wise.

`Fitted, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each row-wise vector of y in Y with

its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen row-wise.

`Jarque-Bera` plots the distributions of the Jarque-Bera (JB) test statistics (Jarque and Bera, 1980), both observed (JB^{obs} as a transparent black density) and replicated (JB^{rep} as a transparent red density). The distribution of JB^{obs} is estimated from the model, and JB^{rep} is simulated from normal residuals, where the number of simulations are the same as the observed number. This Jarque-Bera test may be applied to the residuals of univariate models to test for normality. The Jarque-Bera test does not test normality per se, but whether or not the distribution has kurtosis and skewness that match a normal distribution, and is therefore a test of the moments of a normal distribution. The following summary is reported on the plot: the mean of JB^{obs} (and its 95% probability interval), the probability that $JB^{obs} > JB^{rep}$, and whether or not normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples.

`Jarque-Bera, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Jarque-Bera test, as in `Jarque-Bera` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for normality.

`Mardia` plots the distributions of the skewness (K3) and kurtosis (K4) test statistics (Mardia, 1970), both observed ($K3^{obs}$ and $K4^{obs}$ as transparent black density) and replicated ($K3^{rep}$ and $K4^{rep}$ as transparent red density). The distributions of $K3^{obs}$ and $K4^{obs}$ are estimated from the model, and both $K3^{rep}$ $K4^{rep}$ are simulated from multivariate normal residuals, where the number of simulations are the same as the observed number. This Mardia's test may be applied to the residuals of multivariate models to test for multivariate normality. Mardia's test does not test for multivariate normality per se, but whether or not the distribution has kurtosis and skewness that match a multivariate normal distribution, and is therefore a test of the moments of a multivariate normal distribution. The following summary is reported on the plots: the means of $K3^{obs}$ and $K4^{obs}$ (and the associated 95% probability intervals), the probabilities that $K3^{obs} > K3^{rep}$ and $K4^{obs} > K4^{rep}$, and whether or not multivariate normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples. `Mardia` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. `Y` must be a $N \times P$ matrix of N records and P variables. Source code was modified from the deprecated package QRMlib.

`Predictive Quantiles` plots compare y with the predictive quantile (PQ) of its replicate. This may be useful in looking for patterns with outliers. Instances outside of the gray lines are considered outliers.

`Residual Density` plots the residual density of the median of the samples. A vertical red line occurs at zero. This plot may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when y is univariate and continuous.

`Residual Density, Multivariate C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are column-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen column-wise.

`Residual Density, Multivariate R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are row-wise plots of residual density, given

the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen row-wise.

`Residuals` plots compare y with its residuals. The probability interval is plotted as a line. This plot is appropriate when y is univariate.

`Residuals, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each column-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen column-wise.

`Residuals, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each row-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen row-wise.

`Space-Time` by `Space` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one time-series plot per point s in space, for a total of S plots. Therefore, these are time-series plots for each point s in space across T time-periods. See `Time-Series` plots below.

`Space-Time` by `Time` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one spatial plot per time-period, and T plots will be produced. See `Spatial` plots below.

`Spatial` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows yrep plotted according to its coordinates, and is color-coded so that higher values of yrep become more red, and lower values become more yellow.

`Spatial Uncertainty` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows the probability interval of yrep plotted according to its coordinates, and is color-coded so that wider probability intervals become more red, and lower values become more yellow.

`Time-Series` plots compare y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is univariate and ordered by time.

`Time-Series, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by column in Y.

`Time-Series, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each row-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by row in Y, such as is typically true in panel models.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Durbin, J., and Watson, G.S. (1950). "Testing for Serial Correlation in Least Squares Regression, I." *Biometrika*, 37, p. 409–428.

Jarque, C.M. and Bera, A.K. (1980). "Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals". *Economics Letters*, 6(3), p. 255–259.

Mardia, K.V. (1970). "Measures of Multivariate Skewness and Kurtosis with Applications". *Biometrika*, 57(3), p. 519–530.

See Also

`IterativeQuadrature` and `predict.iterquad`.

Examples

```
### See the IterativeQuadrature function for an example.
```

`plot.juxtapose` *Plot MCMC Juxtaposition*

Description

This may be used to plot a juxtaposition of MCMC algorithms according either to `IAT` or ISM (Independent Samples per Minute).

Usage

```
## S3 method for class 'juxtapose'
plot(x, Style="ISM", ...)
```

Arguments

`x`	This required argument is an object of class `juxtapose`.
`Style`	This argument accepts either `IAT` or `ISM`, and defaults to `ISM`.
`...`	Additional arguments are unused.

Details

When `Style="IAT"`, the medians and 95% probability intervals of the integrated autocorrelation times (IATs) of MCMC algorithms are displayed in a caterpillar plot. The best, or least inefficient, MCMC algorithm is the algorithm with the lowest IAT.

When `Style="ISM"`, the medians and 95% probability intervals of the numbers of independent samples per minute (ISM) of MCMC algorithms are displayed in a caterpillar plot. The best, or least inefficient, MCMC algorithm is the algorithm with the highest ISM.

For more information, see the `Juxtapose` function.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Juxtapose

---

plot.laplace *Plot the output of* LaplaceApproximation

Description

This may be used to plot, or save plots of, the iterated history of the parameters and, if posterior samples were taken, density plots of parameters and monitors in an object of class laplace.

Usage

```
## S3 method for class 'laplace'
plot(x, Data, PDF=FALSE, Parms, ...)
```

Arguments

x	This required argument is an object of class laplace.
Data	This required argument must receive the list of data that was supplied to LaplaceApproximation to create the object of class laplace.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of metacharacters, though these are acceptable: ".", "[", and "]".
...	Additional arguments are unused.

Details

The plots are arranged in a 2×2 matrix. The purpose of the iterated history plots is to show how the value of each parameter and the deviance changed by iteration as the LaplaceApproximation attempted to maximize the logarithm of the unnormalized joint posterior density. If the algorithm converged, and if sir=TRUE in LaplaceApproximation, then plots are produced of selected parameters and all monitored variables.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

LaplaceApproximation

Examples

```
### See the LaplaceApproximation function for an example.
```

---

plot.laplace.ppc *Plots of Posterior Predictive Checks*

Description

This may be used to plot, or save plots of, samples in an object of class laplace.ppc. A variety of plots is provided.

Usage

```
## S3 method for class 'laplace.ppc'
plot(x, Style=NULL, Data=NULL,  Rows=NULL,
     PDF=FALSE, ...)
```

Arguments

x	This required argument is an object of class laplace.ppc.
Style	This optional argument specifies one of several styles of plots, and defaults to NULL (which is the same as "Density"). Styles of plots are indicated in quotes. Optional styles include "Covariates", "Covariates, Categorical DV", "Density", "DW", "DW, Multivariate, C", "ECDF", "Fitted", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera", "Jarque-Bera, Multivariate, C", "Mardia", "Predictive Quantiles", "Residual Density", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R". Details are given below.
Data	This optional argument accepts the data set used when updating the model. Data is required only with certain plot styles, including "Covariates", "Covariates, Categorical DV", "DW, Multivariate, C", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera, Multivariate, C", "Mardia", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R".

Rows	This optional argument is for a vector of row numbers that specify the records associated by row in the object of class laplace.ppc. Only these rows are plotted. The default is to plot all rows. Some plots do not allow rows to be specified.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
...	Additional arguments are unused.

Details

This function can be used to produce a variety of posterior predictive plots, and the style of plot is selected with the Style argument. Below are some notes on the styles of plots.

Covariates requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is not categorical.

Covariates, Categorical DV requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is categorical.

Density plots show the kernel density of the posterior predictive distribution for each selected row of y (all are selected by default). A vertical red line indicates the position of the observed y along the x-axis. When the vertical red line is close to the middle of a normal posterior predictive distribution, then there is little discrepancy between y and the posterior predictive distribution. When the vertical red line is in the tail of the distribution, or outside of the kernel density altogether, then there is a large discrepancy between y and the posterior predictive distribution. Large discrepancies may be considered outliers, and moreover suggest that an improvement in model fit should be considered.

DW plots the distributions of the Durbin-Watson (DW) test statistics (Durbin and Watson, 1950), both observed (d^{obs} as a transparent, black density) and replicated (d^{rep} as a transparent, red density). The distribution of d^{obs} is estimated from the model, and d^{rep} is simulated from normal residuals without autocorrelation, where the number of simulations are the same as the observed number. This DW test may be applied to the residuals of univariate time-series models (or otherwise ordered residuals) to detect first-order autocorrelation. Autocorrelated residuals are not independent. The DW test is applicable only when the residuals are normally-distributed, higher-order autocorrelation is not present, and y is not used also as a lagged predictor. The DW test statistic, d^{obs}, occurs in the interval (0,4), where 0 is perfect positive autocorrelation, 2 is no autocorrelation, and 4 is perfect negative autocorrelation. The following summary is reported on the plot: the mean of d^{obs} (and its 95% probability interval), the probability that $d^{obs} > d^{rep}$, and whether or not autocorrelation is found. Positive autocorrelation is reported when the observed process is greater than the replicated process in 2.5% of the samples, and negative autocorrelation is reported when the observed process is greater than the replicated process in 97.5% of the samples.

DW, Multivariate, C requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Durbin-Watson test, as in DW above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for first-order autocorrelation.

ECDF (Empirical Cumulative Distribution Function) plots compare the ECDF of y with three ECDFs of yhat based on the 2.5%, 50% (median), and 97.5% of its distribution. The ECDF(y) is defined as the proportion of values less than or equal to y. This plot is appropriate when y is univariate and at least ordinal.

`Fitted` plots compare y with the probability interval of its replicate, and provide loess smoothing. This plot is appropriate when y is univariate and not categorical.

`Fitted, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each column-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen column-wise.

`Fitted, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each row-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen row-wise.

`Jarque-Bera` plots the distributions of the Jarque-Bera (JB) test statistics (Jarque and Bera, 1980), both observed (JB^{obs} as a transparent black density) and replicated (JB^{rep} as a transparent red density). The distribution of JB^{obs} is estimated from the model, and JB^{rep} is simulated from normal residuals, where the number of simulations are the same as the observed number. This Jarque-Bera test may be applied to the residuals of univariate models to test for normality. The Jarque-Bera test does not test normality per se, but whether or not the distribution has kurtosis and skewness that match a normal distribution, and is therefore a test of the moments of a normal distribution. The following summary is reported on the plot: the mean of JB^{obs} (and its 95% probability interval), the probability that $JB^{obs} > JB^{rep}$, and whether or not normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples.

`Jarque-Bera, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Jarque-Bera test, as in `Jarque-Bera` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for normality.

`Mardia` plots the distributions of the skewness (K3) and kurtosis (K4) test statistics (Mardia, 1970), both observed ($K3^{obs}$ and $K4^{obs}$ as transparent black density) and replicated ($K3^{rep}$ and $K4^{rep}$ as transparent red density). The distributions of $K3^{obs}$ and $K4^{obs}$ are estimated from the model, and both $K3^{rep}$ $K4^{rep}$ are simulated from multivariate normal residuals, where the number of simulations are the same as the observed number. This Mardia's test may be applied to the residuals of multivariate models to test for multivariate normality. Mardia's test does not test for multivariate normality per se, but whether or not the distribution has kurtosis and skewness that match a multivariate normal distribution, and is therefore a test of the moments of a multivariate normal distribution. The following summary is reported on the plots: the means of $K3^{obs}$ and $K4^{obs}$ (and the associated 95% probability intervals), the probabilities that $K3^{obs} > K3^{rep}$ and $K4^{obs} > K4^{rep}$, and whether or not multivariate normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples. `Mardia` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. `Y` must be a $N \times P$ matrix of N records and P variables. Source code was modified from the deprecated package QRMlib.

`Predictive Quantiles` plots compare y with the predictive quantile (PQ) of its replicate. This may be useful in looking for patterns with outliers. Instances outside of the gray lines are considered outliers.

`Residual Density` plots the residual density of the median of the samples. A vertical red line occurs at zero. This plot may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when y is univariate and continuous.

`Residual Density, Multivariate C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are column-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen column-wise.

`Residual Density, Multivariate R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are row-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen row-wise.

`Residuals` plots compare y with its residuals. The probability interval is plotted as a line. This plot is appropriate when y is univariate.

`Residuals, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each column-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen column-wise.

`Residuals, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each row-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen row-wise.

`Space-Time` by `Space` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one time-series plot per point s in space, for a total of S plots. Therefore, these are time-series plots for each point s in space across T time-periods. See `Time-Series` plots below.

`Space-Time` by `Time` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one spatial plot per time-period, and T plots will be produced. See `Spatial` plots below.

`Spatial` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows yrep plotted according to its coordinates, and is color-coded so that higher values of yrep become more red, and lower values become more yellow.

`Spatial Uncertainty` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows the probability interval of yrep plotted according to its coordinates, and is color-coded so that wider probability intervals become more red, and lower values become more yellow.

`Time-Series` plots compare y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is univariate and ordered by time.

`Time-Series, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by column in Y.

`Time-Series, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each row-wise time-series in Y

with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by row in Y, such as is typically true in panel models.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Durbin, J., and Watson, G.S. (1950). "Testing for Serial Correlation in Least Squares Regression, I." *Biometrika*, 37, p. 409–428.

Jarque, C.M. and Bera, A.K. (1980). "Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals". *Economics Letters*, 6(3), p. 255–259.

Mardia, K.V. (1970). "Measures of Multivariate Skewness and Kurtosis with Applications". *Biometrika*, 57(3), p. 519–530.

See Also

`LaplaceApproximation` and `predict.laplace`.

Examples

```
### See the LaplaceApproximation function for an example.
```

---

plot.miss *Plot samples from the output of MISS*

Description

This may be used to plot, or save plots of, samples in an object of class `miss`. Plots include a trace plot, density plot, and autocorrelation or ACF plot.

Usage

```
## S3 method for class 'miss'
plot(x, PDF=FALSE, ...)
```

Arguments

`x`	This required argument is an object of class `miss`.
`PDF`	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
`...`	Additional arguments are unused.

Details

The plots are arranged in a 3×3 matrix. Each row represents the predictive distribution of a missing value. The left column displays trace plots, the middle column displays kernel density plots, and the right column displays autocorrelation (ACF) plots.

Trace plots show the thinned history of the predictive distribution, with its value in the y-axis moving by iteration across the x-axis. Simulations of a predictive distribution with good properties do not suggest a trend upward or downward as it progresses across the x-axis (it should appear stationary), and it should mix well, meaning it should appear as though random samples are being taken each time from the same target distribution. Visual inspection of a trace plot cannot verify convergence, but apparent non-stationarity or poor mixing can certainly suggest non-convergence. A red, smoothed line also appears to aid visual inspection.

Kernel density plots depict the marginal posterior distribution. There is no distributional assumption about this density.

Autocorrelation plots show the autocorrelation or serial correlation between sampled values at nearby iterations. Samples with autocorrelation do not violate any assumption, but are inefficient because they reduce the effective sample size (ESS), and indicate that the chain is not mixing well, since each value is influenced by values that are previous and nearby. The x-axis indicates lags with respect to samples by iteration, and the y-axis represents autocorrelation. The ideal autocorrelation plot shows perfect correlation at zero lag, and quickly falls to zero autocorrelation for all other lags.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

See Also

MISS.

Examples

```
### See the MISS function for an example.
```

---

plot.pmc *Plot samples from the output of PMC*

Description

This may be used to plot, or save plots of, samples in an object of class pmc. Plots include a trace plot and density plot for parameters, a density plot for deviance and monitored variables, and convergence plots.

Usage

```
## S3 method for class 'pmc'
plot(x, BurnIn=0, Data, PDF=FALSE, Parms, ...)
```

Arguments

x	This required argument is an object of class pmc.
BurnIn	This argument requires zero or a positive integer that indicates the number of iterations to discard as burn-in for the purposes of plotting.
Data	This required argument must receive the list of data that was supplied to PMC to create the object of class pmc.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
...	Additional arguments are unused.

Details

The plots are arranged in a 2×2 matrix. Each row represents a parameter, the deviance, or a monitored variable. For parameters, the left column displays trace plots and the right column displays kernel density plots.

Trace plots show the history of the distribution of independent importance samples. When multiple mixture components are used, each mixture component has a different color. These plots are unavailable for the deviance and monitored variables.

Kernel density plots depict the marginal posterior distribution. Although there is no distributional assumption about this density, kernel density estimation uses Gaussian basis functions.

Following these plots are three plots for convergence. First, ESSN (red) and perplexity (black) are plotted by iteration. Convergence occurs when both of these seem to stabilize, and higher is better. The second plot shows the distribution of the normalized importance weights by iteration. The third plot appears only when multiple mixture components are used. The third plot displays the probabilities of each mixture component by iteration. Although the last two plots are not formally convergence plots, they are provided so the user can verify the distribution of importance weights and the mixture probabilities have become stable.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

ESS and PMC.

Examples

```
### See the PMC function for an example.
```

plot.pmc.ppc *Plots of Posterior Predictive Checks*

Description

This may be used to plot, or save plots of, samples in an object of class pmc.ppc. A variety of plots is provided.

Usage

```
## S3 method for class 'pmc.ppc'
plot(x, Style=NULL, Data=NULL, Rows=NULL,
     PDF=FALSE, ...)
```

Arguments

x	This required argument is an object of class pmc.ppc.
Style	This optional argument specifies one of several styles of plots, and defaults to NULL (which is the same as "Density"). Styles of plots are indicated in quotes. Optional styles include "Covariates", "Covariates, Categorical DV", "Density", "DW", "DW, Multivariate, C", "ECDF", "Fitted", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera", "Jarque-Bera, Multivariate, C", "Mardia", "Predictive Quantiles", "Residual Density", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R". Details are given below.
Data	This optional argument accepts the data set used when updating the model. Data is required only with certain plot styles, including "Covariates", "Covariates, Categorical DV", "DW, Multivariate, C", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera, Multivariate, C", "Mardia", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R".
Rows	This optional argument is for a vector of row numbers that specify the records associated by row in the object of class pmc.ppc. Only these rows are plotted. The default is to plot all rows. Some plots do not allow rows to be specified.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
...	Additional arguments are unused.

Details

This function can be used to produce a variety of posterior predictive plots, and the style of plot is selected with the `Style` argument. Below are some notes on the styles of plots.

`Covariates` requires `Data` to be specified, and also requires that the covariates are named `X` or `x`. A plot is produced for each covariate column vector against yhat, and is appropriate when y is not categorical.

`Covariates, Categorical DV` requires `Data` to be specified, and also requires that the covariates are named `X` or `x`. A plot is produced for each covariate column vector against yhat, and is appropriate when y is categorical.

`Density` plots show the kernel density of the posterior predictive distribution for each selected row of y (all are selected by default). A vertical red line indicates the position of the observed y along the x-axis. When the vertical red line is close to the middle of a normal posterior predictive distribution, then there is little discrepancy between y and the posterior predictive distribution. When the vertical red line is in the tail of the distribution, or outside of the kernel density altogether, then there is a large discrepancy between y and the posterior predictive distribution. Large discrepancies may be considered outliers, and moreover suggest that an improvement in model fit should be considered.

`DW` plots the distributions of the Durbin-Watson (DW) test statistics (Durbin and Watson, 1950), both observed (d^{obs} as a transparent, black density) and replicated (d^{rep} as a transparent, red density). The distribution of d^{obs} is estimated from the model, and d^{rep} is simulated from normal residuals without autocorrelation, where the number of simulations are the same as the observed number. This DW test may be applied to the residuals of univariate time-series models (or otherwise ordered residuals) to detect first-order autocorrelation. Autocorrelated residuals are not independent. The DW test is applicable only when the residuals are normally-distributed, higher-order autocorrelation is not present, and y is not used also as a lagged predictor. The DW test statistic, d^{obs}, occurs in the interval (0,4), where 0 is perfect positive autocorrelation, 2 is no autocorrelation, and 4 is perfect negative autocorrelation. The following summary is reported on the plot: the mean of d^{obs} (and its 95% probability interval), the probability that $d^{obs} > d^{rep}$, and whether or not autocorrelation is found. Positive autocorrelation is reported when the observed process is greater than the replicated process in 2.5% of the samples, and negative autocorrelation is reported when the observed process is greater than the replicated process in 97.5% of the samples.

`DW, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Durbin-Watson test, as in `DW` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for first-order autocorrelation.

`ECDF` (Empirical Cumulative Distribution Function) plots compare the ECDF of y with three ECDFs of yhat based on the 2.5%, 50% (median), and 97.5% of its distribution. The ECDF(y) is defined as the proportion of values less than or equal to y. This plot is appropriate when y is univariate and at least ordinal.

`Fitted` plots compare y with the probability interval of its replicate, and provide loess smoothing. This plot is appropriate when y is univariate and not categorical.

`Fitted, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each column-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen column-wise.

`Fitted, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each row-wise vector of y in Y with

its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen row-wise.

`Jarque-Bera` plots the distributions of the Jarque-Bera (JB) test statistics (Jarque and Bera, 1980), both observed (JB^{obs} as a transparent black density) and replicated (JB^{rep} as a transparent red density). The distribution of JB^{obs} is estimated from the model, and JB^{rep} is simulated from normal residuals, where the number of simulations are the same as the observed number. This Jarque-Bera test may be applied to the residuals of univariate models to test for normality. The Jarque-Bera test does not test normality per se, but whether or not the distribution has kurtosis and skewness that match a normal distribution, and is therefore a test of the moments of a normal distribution. The following summary is reported on the plot: the mean of JB^{obs} (and its 95% probability interval), the probability that $JB^{obs} > JB^{rep}$, and whether or not normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples.

`Jarque-Bera, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Jarque-Bera test, as in `Jarque-Bera` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for normality.

`Mardia` plots the distributions of the skewness (K3) and kurtosis (K4) test statistics (Mardia, 1970), both observed ($K3^{obs}$ and $K4^{obs}$ as transparent black density) and replicated ($K3^{rep}$ and $K4^{rep}$ as transparent red density). The distributions of $K3^{obs}$ and $K4^{obs}$ are estimated from the model, and both $K3^{rep}$ $K4^{rep}$ are simulated from multivariate normal residuals, where the number of simulations are the same as the observed number. This Mardia's test may be applied to the residuals of multivariate models to test for multivariate normality. Mardia's test does not test for multivariate normality per se, but whether or not the distribution has kurtosis and skewness that match a multivariate normal distribution, and is therefore a test of the moments of a multivariate normal distribution. The following summary is reported on the plots: the means of $K3^{obs}$ and $K4^{obs}$ (and the associated 95% probability intervals), the probabilities that $K3^{obs} > K3^{rep}$ and $K4^{obs} > K4^{rep}$, and whether or not multivariate normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples. `Mardia` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. `Y` must be a $N \times P$ matrix of N records and P variables. Source code was modified from the deprecated package QRMlib.

`Predictive Quantiles` plots compare y with the predictive quantile (PQ) of its replicate. This may be useful in looking for patterns with outliers. Instances outside of the gray lines are considered outliers.

`Residual Density` plots the residual density of the median of the samples. A vertical red line occurs at zero. This plot may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when y is univariate and continuous.

`Residual Density, Multivariate C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are column-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen column-wise.

`Residual Density, Multivariate R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are row-wise plots of residual density, given

the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen row-wise.

`Residuals` plots compare y with its residuals. The probability interval is plotted as a line. This plot is appropriate when y is univariate.

`Residuals, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each column-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen column-wise.

`Residuals, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These are plots of each row-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen row-wise.

`Space-Time` by `Space` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one time-series plot per point s in space, for a total of S plots. Therefore, these are time-series plots for each point s in space across T time-periods. See `Time-Series` plots below.

`Space-Time` by `Time` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude`, `longitude`, `S`, and `T`. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one spatial plot per time-period, and T plots will be produced. See `Spatial` plots below.

`Spatial` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows yrep plotted according to its coordinates, and is color-coded so that higher values of yrep become more red, and lower values become more yellow.

`Spatial Uncertainty` requires `Data` to be specified, and also requires that the following variables exist in the data set with exactly these names: `latitude` and `longitude`. This spatial plot shows the probability interval of yrep plotted according to its coordinates, and is color-coded so that wider probability intervals become more red, and lower values become more yellow.

`Time-Series` plots compare y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is univariate and ordered by time.

`Time-Series, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by column in Y.

`Time-Series, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each row-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by row in Y, such as is typically true in panel models.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Durbin, J., and Watson, G.S. (1950). "Testing for Serial Correlation in Least Squares Regression, I." *Biometrika*, 37, p. 409–428.

Jarque, C.M. and Bera, A.K. (1980). "Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals". *Economics Letters*, 6(3), p. 255–259.

Mardia, K.V. (1970). "Measures of Multivariate Skewness and Kurtosis with Applications". *Biometrika*, 57(3), p. 519–530.

See Also

PMC and predict.pmc.

Examples

```
### See the PMC function for an example.
```

---

plot.vb *Plot the output of* VariationalBayes

Description

This may be used to plot, or save plots of, the iterated history of the parameters and variances, and if posterior samples were taken, density plots of parameters and monitors in an object of class vb.

Usage

```
## S3 method for class 'vb'
plot(x, Data, PDF=FALSE, Parms, ...)
```

Arguments

x	This required argument is an object of class vb.
Data	This required argument must receive the list of data that was supplied to VariationalBayes to create the object of class vb.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters for plotting. This argument defaults to NULL and selects every parameter for plotting. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".
...	Additional arguments are unused.

Details

The plots are arranged in a 3×3 matrix. The purpose of the iterated history plots is to show how the value of each parameter, variance, and the deviance changed by iteration as the VariationalBayes attempted to maximize the logarithm of the unnormalized joint posterior density. If the algorithm converged, and if sir=TRUE in VariationalBayes, then plots are produced of selected parameters and all monitored variables.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

VariationalBayes

Examples

```
### See the VariationalBayes function for an example.
```

---

plot.vb.ppc *Plots of Posterior Predictive Checks*

Description

This may be used to plot, or save plots of, samples in an object of class vb.ppc. A variety of plots is provided.

Usage

```
## S3 method for class 'vb.ppc'
plot(x, Style=NULL, Data=NULL,  Rows=NULL,
     PDF=FALSE, ...)
```

Arguments

x	This required argument is an object of class vb.ppc.
Style	This optional argument specifies one of several styles of plots, and defaults to NULL (which is the same as "Density"). Styles of plots are indicated in quotes. Optional styles include "Covariates", "Covariates, Categorical DV", "Density", "DW", "DW, Multivariate, C", "ECDF", "Fitted", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera", "Jarque-Bera, Multivariate, C", "Mardia", "Predictive Quantiles", "Residual Density", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R". Details are given below.

Data	This optional argument accepts the data set used when updating the model. Data is required only with certain plot styles, including "Covariates", "Covariates, Categorical DV", "DW, Multivariate, C", "Fitted, Multivariate, C", "Fitted, Multivariate, R", "Jarque-Bera, Multivariate, C", "Mardia", "Residual Density, Multivariate, C", "Residual Density, Multivariate, R", "Residuals, Multivariate, C", "Residuals, Multivariate, R", "Space-Time by Space", "Space-Time by Time", "Spatial", "Spatial Uncertainty", "Time-Series, Multivariate, C", and "Time-Series, Multivariate, R".
Rows	This optional argument is for a vector of row numbers that specify the records associated by row in the object of class vb.ppc. Only these rows are plotted. The default is to plot all rows. Some plots do not allow rows to be specified.
PDF	This logical argument indicates whether or not the user wants Laplace's Demon to save the plots as a .pdf file.
...	Additional arguments are unused.

Details

This function can be used to produce a variety of posterior predictive plots, and the style of plot is selected with the Style argument. Below are some notes on the styles of plots.

Covariates requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is not categorical.

Covariates, Categorical DV requires Data to be specified, and also requires that the covariates are named X or x. A plot is produced for each covariate column vector against yhat, and is appropriate when y is categorical.

Density plots show the kernel density of the posterior predictive distribution for each selected row of y (all are selected by default). A vertical red line indicates the position of the observed y along the x-axis. When the vertical red line is close to the middle of a normal posterior predictive distribution, then there is little discrepancy between y and the posterior predictive distribution. When the vertical red line is in the tail of the distribution, or outside of the kernel density altogether, then there is a large discrepancy between y and the posterior predictive distribution. Large discrepancies may be considered outliers, and moreover suggest that an improvement in model fit should be considered.

DW plots the distributions of the Durbin-Watson (DW) test statistics (Durbin and Watson, 1950), both observed (d^{obs} as a transparent, black density) and replicated (d^{rep} as a transparent, red density). The distribution of d^{obs} is estimated from the model, and d^{rep} is simulated from normal residuals without autocorrelation, where the number of simulations are the same as the observed number. This DW test may be applied to the residuals of univariate time-series models (or otherwise ordered residuals) to detect first-order autocorrelation. Autocorrelated residuals are not independent. The DW test is applicable only when the residuals are normally-distributed, higher-order autocorrelation is not present, and y is not used also as a lagged predictor. The DW test statistic, d^{obs}, occurs in the interval (0,4), where 0 is perfect positive autocorrelation, 2 is no autocorrelation, and 4 is perfect negative autocorrelation. The following summary is reported on the plot: the mean of d^{obs} (and its 95% probability interval), the probability that $d^{obs} > d^{rep}$, and whether or not autocorrelation is found. Positive autocorrelation is reported when the observed process is greater than the replicated process in 2.5% of the samples, and negative autocorrelation is reported when the observed process is greater than the replicated process in 97.5% of the samples.

`DW, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Durbin-Watson test, as in `DW` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for first-order autocorrelation.

`ECDF` (Empirical Cumulative Distribution Function) plots compare the ECDF of y with three ECDFs of yhat based on the 2.5%, 50% (median), and 97.5% of its distribution. The ECDF(y) is defined as the proportion of values less than or equal to y. This plot is appropriate when y is univariate and at least ordinal.

`Fitted` plots compare y with the probability interval of its replicate, and provide loess smoothing. This plot is appropriate when y is univariate and not categorical.

`Fitted, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each column-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen column-wise.

`Fitted, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exists in the data set with exactly that name. These plots compare each row-wise vector of y in Y with its replicates and provide loess smoothing. This plot is appropriate when Y is multivariate, not categorical, and desired to be seen row-wise.

`Jarque-Bera` plots the distributions of the Jarque-Bera (JB) test statistics (Jarque and Bera, 1980), both observed (JB^{obs} as a transparent black density) and replicated (JB^{rep} as a transparent red density). The distribution of JB^{obs} is estimated from the model, and JB^{rep} is simulated from normal residuals, where the number of simulations are the same as the observed number. This Jarque-Bera test may be applied to the residuals of univariate models to test for normality. The Jarque-Bera test does not test normality per se, but whether or not the distribution has kurtosis and skewness that match a normal distribution, and is therefore a test of the moments of a normal distribution. The following summary is reported on the plot: the mean of JB^{obs} (and its 95% probability interval), the probability that $JB^{obs} > JB^{rep}$, and whether or not normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples.

`Jarque-Bera, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise vector of residuals with a univariate Jarque-Bera test, as in `Jarque-Bera` above. This plot is appropriate when Y is multivariate, not categorical, and residuals are desired to be tested column-wise for normality.

`Mardia` plots the distributions of the skewness (K3) and kurtosis (K4) test statistics (Mardia, 1970), both observed ($K3^{obs}$ and $K4^{obs}$ as transparent black density) and replicated ($K3^{rep}$ and $K4^{rep}$ as transparent red density). The distributions of $K3^{obs}$ and $K4^{obs}$ are estimated from the model, and both $K3^{rep}$ $K4^{rep}$ are simulated from multivariate normal residuals, where the number of simulations are the same as the observed number. This Mardia's test may be applied to the residuals of multivariate models to test for multivariate normality. Mardia's test does not test for multivariate normality per se, but whether or not the distribution has kurtosis and skewness that match a multivariate normal distribution, and is therefore a test of the moments of a multivariate normal distribution. The following summary is reported on the plots: the means of $K3^{obs}$ and $K4^{obs}$ (and the associated 95% probability intervals), the probabilities that $K3^{obs} > K3^{rep}$ and $K4^{obs} > K4^{rep}$, and whether or not multivariate normality is indicated. Non-normality is reported when the observed process is greater than the replicated process in either 2.5% or 97.5% of the samples. `Mardia`

requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. Y must be a $N \times P$ matrix of N records and P variables. Source code was modified from the deprecated package QRMlib.

Predictive Quantiles plots compare y with the predictive quantile (PQ) of its replicate. This may be useful in looking for patterns with outliers. Instances outside of the gray lines are considered outliers.

Residual Density plots the residual density of the median of the samples. A vertical red line occurs at zero. This plot may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when y is univariate and continuous.

Residual Density, Multivariate C requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are column-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen column-wise.

Residual Density, Multivariate R requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are row-wise plots of residual density, given the median of the samples. These plots may be useful for inspecting a distributional assumption of residual variance. This plot is appropriate when Y is multivariate, continuous, and densities are desired to be seen row-wise.

Residuals plots compare y with its residuals. The probability interval is plotted as a line. This plot is appropriate when y is univariate.

Residuals, Multivariate, C requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are plots of each column-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen column-wise.

Residuals, Multivariate, R requires Data to be specified, and also requires that variable Y exist in the data set with exactly that name. These are plots of each row-wise vector of residuals. The probability interval is plotted as a line. This plot is appropriate when Y is multivariate, not categorical, and the residuals are desired to be seen row-wise.

Space-Time by Space requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude, longitude, S, and T. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one time-series plot per point s in space, for a total of S plots. Therefore, these are time-series plots for each point s in space across T time-periods. See Time-Series plots below.

Space-Time by Time requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude, longitude, S, and T. These space-time plots compare the S x T matrix Y with the S x T matrix Yrep, producing one spatial plot per time-period, and T plots will be produced. See Spatial plots below.

Spatial requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude and longitude. This spatial plot shows yrep plotted according to its coordinates, and is color-coded so that higher values of yrep become more red, and lower values become more yellow.

Spatial Uncertainty requires Data to be specified, and also requires that the following variables exist in the data set with exactly these names: latitude and longitude. This spatial plot shows

the probability interval of yrep plotted according to its coordinates, and is color-coded so that wider probability intervals become more red, and lower values become more yellow.

`Time-Series` plots compare y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is univariate and ordered by time.

`Time-Series, Multivariate, C` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each column-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by column in Y.

`Time-Series, Multivariate, R` requires `Data` to be specified, and also requires that variable `Y` exist in the data set with exactly that name. These plots compare each row-wise time-series in Y with its replicate, including the median and probability interval quantiles. This plot is appropriate when y is multivariate and each time-series is indexed by row in Y, such as is typically true in panel models.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Durbin, J., and Watson, G.S. (1950). "Testing for Serial Correlation in Least Squares Regression, I." *Biometrika*, 37, p. 409–428.

Jarque, C.M. and Bera, A.K. (1980). "Efficient Tests for Normality, Homoscedasticity and Serial Independence of Regression Residuals". *Economics Letters*, 6(3), p. 255–259.

Mardia, K.V. (1970). "Measures of Multivariate Skewness and Kurtosis with Applications". *Biometrika*, 57(3), p. 519–530.

See Also

`predict.vb` and `VariationalBayes`.

Examples

```
### See the VariationalBayes function for an example.
```

`plotMatrix` *Plot a Numerical Matrix*

Description

This function plots a numerical matrix, and is often used to plot the following matrices: correlation, covariance, distance, and precision.

Usage

```
plotMatrix(x, col=colorRampPalette(c("red","black","green"))(100),
     cex=1, circle=TRUE, order=FALSE, zlim=NULL, title="", PDF=FALSE, ...)
```

Arguments

x	This required argument is a numerical matrix, or an object of class bayesfactor, demonoid, iterquad, laplace, pmc, posteriorchecks, or vb. See more information below regarding these classes. One component of a blocked proposal covariance matrix must be pointed to explicitly, rather than to the object of class demonoid.
col	This argument specifies the colors of the circles. By default, the colorRampPalette function colors strong positive correlation as green, zero correlation as black, and strong negative correlation as red, and provides 100 color gradations.
cex	When circle=TRUE, this argument specifies the size of the marginal text, the names of the parameters or variables, and defaults to 1.
circle	Logical. When TRUE, each element in the numeric matrix is represented with a circle, and a larger circle is assigned to elements that are farther from zero. Also, when TRUE, the gradation scale does not appear to the right of the plot.
order	Logical. This argument defaults to FALSE, and presents the parameters or variables in the same order as in the numeric matrix. When TRUE, the parameters or variables are ordered using principal components analysis.
zlim	When circle=FALSE, the gradation scale may be constrained to an interval by zlim, such as zlim=c(-1,1), and only values within the interval are plotted.
title	This argument specifies the title of the plot, and the default does not include a title. When x is of class posteriorchecks, the title is changed to Posterior Correlation.
PDF	Logical. When TRUE, the plot is saved as a .pdf file.
...	Additional arguments are unused.

Details

The plotMatrix function produces one of two styles of plots, depending on the circle argument. A $K \times K$ numeric matrix of K parameters or variables is plotted. The plot is a matrix of the same dimensions, in which each element is colored (and sized, when circle=TRUE) according to its value.

Although plotMatrix does not provide the same detail as a numeric matrix, it is easier to discover elements of interest according to color (and size when circle=TRUE).

The plotMatrix function is not inherently Bayesian, and does not include uncertainty in matrices. Nonetheless, it is included because it is a useful graphical presentation of a numeric matrices, and is recommended to be used with the posterior correlation matrix in an object of class posteriorchecks.

When x is an object of class bayesfactor, matrix B is plotted. When x is an object of class demonoid (if it is a matrix), iterquad, laplace, pmc, or vb, the covariance matrix Covar is plotted. When x is an object of class posteriorchecks, the posterior correlation matrix is plotted.

This is a modified version of the circle.corr function of Taiyun Wei.

Author(s)

Taiyun Wei

See Also

PosteriorChecks

Examples

```
library(LaplacesDemon)
### Although it is most commonly used with an object of class
### posteriorchecks, it is applied here to a different correlation matrix.
data(mtcars)
plotMatrix(cor(mtcars), col=colorRampPalette(c("green","gray10","red"))(100),
     cex=1, circle=FALSE, order=TRUE)
plotMatrix(cor(mtcars), col=colorRampPalette(c("green","gray10","red"))(100),
     cex=1, circle=TRUE, order=TRUE)
```

---

plotSamples *Plot Samples*

Description

This function provides basic plots that are extended to include samples.

Usage

```
plotSamples(X, Style="KDE", LB=0.025, UB=0.975, Title=NULL)
```

Arguments

X	This required argument is a $N \times S$ numerical matrix of N records and S samples.
Style	This argument accepts the following quoted strings: "barplot", "dotchart", "hist", "KDE", or "Time-Series". It defaults to Style="KDE".
LB	This argument accepts the lower bound of a probability interval, which must be in the interval [0,0.5).
UB	This argument accepts the upper bound of a probability interval, which must be in the interval (0.5,1].
Title	This argument defaults to NULL, and otherwise accepts a quoted string that will be the title of the plot.

Details

The plotSamples function extends several basic plots from points to samples. For example, it is common to use the hist function to plot a histogram from a column vector. However, the user may desire to plot a histogram of a column vector that was sampled numerous times, rather than a simple column vector, in which a (usually 95%) probability interval is also plotted to show the uncertainty around the sampled median of each bin in the histogram.

The plotSamples function extends the barplot, dotchart, and hist functions to include uncertainty due to samples. The KDE style of plot is added so that a probability interval is shown around

a sampled kernel density estimate of a distribution, and the Time-Series style of plot is added so that a probability interval is shown around a sampled univariate time-series.

For each style of plot, three quantiles are plotted: the lower bound (LB), median, and upper bound (UB).

One of many potential Bayesian applications is to examine the uncertainty in a predictive distribution.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

Examples

```
#library(LaplacesDemon)
#N <- 100
#S <- 100
#X <- matrix(rnorm(N*S),N,S)
#rownames(X) <- 1:100
#plotSamples(X, Style="barplot", LB=0.025, UB=0.975)
#plotSamples(X[1:10,], Style="dotchart", LB=0.025, UB=0.975)
#plotSamples(X, Style="hist", LB=0.025, UB=0.975)
#plotSamples(X, Style="KDE", LB=0.025, UB=0.975)
#plotSamples(X, Style="Time-Series", LB=0.025, UB=0.975)
```

PMC *Population Monte Carlo*

Description

The PMC function updates a model with Population Monte Carlo. Given a model specification, data, and initial values, PMC maximizes the logarithm of the unnormalized joint posterior density and provides samples of the marginal posterior distributions, deviance, and other monitored variables.

Usage

```
PMC(Model, Data, Initial.Values, Covar=NULL, Iterations=10, Thinning=1,
alpha=NULL, M=1, N=1000, nu=9, CPUs=1, Type="PSOCK")
```

Arguments

Model	This is a model specification function. For more information, see LaplacesDemon.
Initial.Values	This is either a vector initial values, one for each of K parameters, or in the case of a mixture of M components, this is a $M \times K$ matrix of initial values. If all initial values are zero in this vector, or in the first row of a matrix, then LaplaceApproximation is used to optimize initial values, in which case all mixture components receive the same initial values and covariance matrix from the object of class laplace. Parameters must be continuous.

`Data`	This is a list of data. For more information, see `LaplacesDemon`.
`Covar`	This is a $K \times K$ covariance matrix for K parameters, or for multiple mixture components, this is a $K \times K \times M$ array of M covariance matrices, where M is the number of mixture components. `Covar` defaults to `NULL`, in which case a scaled identity matrix (with the same scale as in `LaplacesDemon`) is applied to all mixture components.
`Iterations`	This is the number of iterations during which PMC will update the model. Updating the model for only one iteration is the same as applying non-adaptive importance sampling.
`Thinning`	This is the number by which the posterior is thinned. To have 1,000 posterior samples with `M=3` mixture components and `N=10000` samples each, `Thinning=30`. For more information, see the `Thin` function.
`alpha`	This is a vector of length M, the number of mixture components. α is the probability of each mixture component. The default value is `NULL`, which assigns an equal probability to each component.
`M`	This is the number M of multivariate t distribution mixture components.
`N`	This is the number N of samples per mixture component. The required number of samples increases with the number K of parameters. These samples are also called walkers or particles.
`nu`	This is the degrees of freedom parameter ν for the multivariate t distribution for each mixture component. If a multivariate normal distribution is preferred, then set $\nu > 1e4$.
`CPUs`	This argument is required for parallel processing, and and indicates the number of central processing units (CPUs) of the computer or cluster. For example, when a user has a quad-core computer, `CPUs=4`.
`Type`	This argument defaults to `"PSOCK"` and uses the Simple Network of Workstations (SNOW) for parallelization. Alternatively, `Type="MPI"` may be specified to use Message Passing Interface (MPI) for parallelization.

Details

The `PMC` function uses the adaptive importance sampling algorithm of Wraith et al. (2009), also called Mixture PMC or M-PMC (Cappe et al., 2008). Iterative adaptive importance sampling was introduced in the 1980s. Modern PMC was introduced (Cappe et al., 2004), and extended to multivariate Gaussian or t-distributed mixtures (Cappe et al., 2008). This version uses a multivariate t distribution for each mixture component, and also allows a multivariate normal distribution when the degrees of freedom, $\nu > 1e4$. At each iteration, a mixture distribution is sampled with importance sampling, and the samples (or populations) are adapted to improve the importance sampling. Adaptation is a variant of EM (Expectation-Maximization). The sample is self-normalized, and is an example of self-normalized importance sampling (SNIS), or self-importance sampling. The vector α contains the probability of each mixture component. These, as well as multivariate t distribution mixture parameters (except ν), are adapted each iteration.

Advantages of PMC over MCMC include:

- It is difficult to assess convergence of MCMC chains, and this is not necessary in PMC (Wraith et al., 2009).

- MCMC chains have autocorrelation that effectively reduces posterior samples. PMC produces independent samples that are not reduced with autocorrelation.
- PMC has been reported to produce samples with less variance than MCMC.
- It is difficult to parallelize MCMC. Posterior samples from parallel chains can be pooled when all chains have converged, but until this occurs, parallelization is unhelpful. PMC, on the other hand, can parallelize the independent, Monte Carlo samples during each iteration and reduce run-time as the number of processors increases. Currently, PMC is not parallelized here.
- The multivariate mixture in PMC can represent a multimodal posterior, where MCMC with parallel chains may be used to identify a multimodal posterior, but probably will not yield combined samples that proportionally represent it.

Disadvantages of PMC, compared to MCMC, include:

- In PMC, the required number of samples at each iteration increases quickly with respect to an increase in parameters. MCMC is more suitable for models with large numbers of parameters, and therefore, MCMC is more generalizable.
- PMC is more sensitive to initial values than MCMC, especially as the number of parameters increases.
- PMC is more sensitive to the initial covariance matrix (or matrices for mixture components) than adaptive MCMC. PMC requires more information about the target distributions before updating. The covariance matrix from a converged iterative quadrature algorithm, Laplace Approximation, or Variational Bayes may be required (see `IterativeQuadrature`, `LaplaceApproximation`, or `VariationalBayes` for more information).

Since PMC requires better initial information than iterative quadrature, Laplace Approximation, MCMC, and Variational Bayes, it is not recommended to begin updating a model that has little prior information with PMC, especially when the model has more than a few parameters. Instead, iterative quadrature, Laplace Approximation, MCMC, or Variational Bayes should be used. However, once convergence is found or assumed, it is recommended to attempt to update the model with PMC, given the latest parameters and convariance matrix from iterative quadrature, Laplace Approximation, MCMC, or Variational Bayes. Used in this way, PMC may improve the model fit obtained with MCMC and should reduce the variance of the marginal posterior distributions, which is desirable for predictive modeling.

Convergence is assessed by observing two outputs: normalized effective sample size (`ESSN`) and normalized perplexity (`Perplexity`). These are described below. PMC is considered to have converged when these diagnostics stabilize (Wraith et al., 2009), or when the normalized perplexity becomes sufficiently close to 1 (Cappe et al., 2008). If they do not stabilize, then it is suggested to begin PMC again with a larger number `N` of samples, and possibly with different initial values and covariance matrix or matrices. `IterativeQuadrature`, `LaplaceApproximation`, or `VariationalBayes` may be helpful to provide better starting values for `PMC`.

If a message appears that warns about 'bad weights', then `PMC` is attempting to work with an iteration in which importance weights are problematic. If this occurs in the first iteration, then all importance weights are set to $1/N$. If this occurs in other iterations, then the information from the previous iteration is used instead and different draws are made from that importance distribution. This may allow `PMC` to eventually adapt successfully to the target. If not, the user is advised to begin again with a larger number N of samples, and possibly different initial values and covariance matrix or matrices, as above. PMC can experience difficulty when it begins with poor initial conditions.

The user may combine samples from previous iterations with samples from the latest iteration for inference, if the algorithm converged before the last iteration. Currently, a function is not provided for combining previous samples.

Value

The returned object is an object of class `pmc` with the following components:

`alpha`	This is a $M \times T$ matrix of the probabilities of mixture components, where M is the number of mixture components and T is the number of iterations.
`Call`	This is the matched call of `PMC`.
`Covar`	This stores the $K \times K \times T \times M$ proposal covariance matrix in an array, where K is the dimension or number of parameters or initial values, T is the number of iterations, and M is the number of mixture components. If the model is updated in the future, then the latest covariance matrix for each mixture component can be extracted and used to start the next update where the last update left off.
`Deviance`	This is a vector of the deviance of the model, with a length equal to the number of thinned samples that were retained. Deviance is useful for considering model fit, and is equal to the sum of the log-likelihood for all rows in the data set, which is then multiplied by negative two.
`DIC`	This is a vector of three values: Dbar, pD, and DIC. Dbar is the mean deviance, pD is a measure of model complexity indicating the effective number of parameters, and DIC is the Deviance Information Criterion, which is a model fit statistic that is the sum of Dbar and pD. `DIC` is calculated over the thinned samples. Note that pD is calculated as `var(Deviance)/2` as in Gelman et al. (2004).
`ESSN`	This is a vector of length T that contains the normalized effective sample size (ESSN) per iteration across T iterations. ESSN is used as a convergence diagnostic. ESSN is normalized between zero and one, and can be interpreted as the proportion of samples with non-zero weight. Higher is better.
`Initial.Values`	This is the vector or matrix of `Initial.Values`.
`Iterations`	This reports the number of `Iterations` for updating.
`LML`	This is an approximation of the logarithm of the marginal likelihood of the data (see the `LML` function for more information). `LML` is estimated with nonparametric self-normalized importance sampling (NSIS), given LL and the marginal posterior samples of the parameters. `LML` is useful for comparing multiple models with the `BayesFactor` function.
`M`	This reports the number of mixture components.
`Minutes`	This indicates the number of minutes that `PMC` was running, and this includes the initial checks as well as time it took to perform final sampling and create summaries.
`Model`	This contains the model specification `Model`.
`N`	This is the number of un-thinned samples per mixture component.

itemnuThis is the degrees of freedom parameter ν for each multivariate t distribution in each mixture component.

`Mu` This is a $T \times K \times M$ array of means for the importance sampling distribution across T iterations, K parameters, and M mixture components.

`Monitor` This is a $S \times J$ matrix of thinned samples of monitored variables, where S is the number of thinned samples and J is the number of monitored variables.

`Parameters` This reports the number K of parameters.

`Perplexity` This is a vector of length T that contains the normalized perplexity per iteration across T iterations, and is used as a convergence diagnostic. Perplexity is an approximation of the negative of the Kullback-Leibler divergence (see KLD) between the target and the importance function. Perplexity is normalized between zero and one, and a higher normalized perplexity relates to less divergence, so higher is better. A normalized perplexity that is close to one indicates good agreement between the target density and the importance function. This is based on the Shannon entropy of the normalized importance weights, which is used frequently to measure the quality of importance samples.

`Posterior1` This is an $N \times K \times T \times M$ array of un-thinned posterior samples across N samples, K parameters, T iterations, and M mixture components.

`Posterior2` This is a $S \times K$ matrix of thinned posterior samples, where S is the number of thinned samples and K is the number of parameters.

`Summary` This is a matrix that summarizes the marginal posterior distributions of the parameters, deviance, and monitored variables from thinned samples. The following summary statistics are included: mean, standard deviation, MCSE (Monte Carlo Standard Error), ESS is the effective sample size due to autocorrelation, and finally the 2.5%, 50%, and 97.5% quantiles are reported. MCSE is essentially a standard deviation around the marginal posterior mean that is due to uncertainty associated with using Monte Carlo sampling. The acceptable size of the MCSE depends on the acceptable uncertainty associated around the marginal posterior mean. The default `IMPS` method is used. Next, the desired precision of ESS depends on the user's goal.

`Thinned.Samples`
This is the number of thinned samples in `Posterior2`.

`Thinning` This is the amount of thinning requested by the user.

`W` This is a $N \times T$ matrix of normalized importance weights, where N is the number of un-thinned samples per mixture component and T is the number of iterations. Computationally, the algorithm uses the logarithm of the weights.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Cappe, O., Douc, R., Guillin, A., Marin, J.M., and Robert, C. (2008). "Adaptive Importance Sampling in General Mixture Classes". *Statistics and Computing*, 18, p. 587–600.

Cappe, O., Guillin, A., Marin, J.M., and Robert, C. (2004). "Population Monte Carlo". *Journal of Computational and Graphical Statistics*, 13, p. 907–929.

Gelman, A., Carlin, J., Stern, H., and Rubin, D. (2004). "Bayesian Data Analysis, Texts in Statistical Science, 2nd ed.". Chapman and Hall, London.

Wraith, D., Kilbinger, M., Benabed, K., Cappe, O., Cardoso, J.F., Fort, G., Prunet, S., and Robert, C.P. (2009). "Estimation of Cosmological Parameters Using Adaptive Importance Sampling". *Physical Review D*, 80(2), p. 023507.

See Also

BayesFactor, IterativeQuadrature, LaplaceApproximation, LML, PMC.RAM, Thin, and VariationalBayes.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#####################  Load the LaplacesDemon Library  #####################
library(LaplacesDemon)

#############################  Demon Data  #############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,c(1,4,10)]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

########################  Data List Preparation  ########################
mon.names <- "LP"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

########################  Model Specification  ########################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=LP,
```

```
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

set.seed(666)

############################  Initial Values  ############################
Initial.Values <- GIV(Model, MyData, PGF=TRUE)

########################  Population Monte Carlo  ########################
Fit <- PMC(Model, MyData, Initial.Values, Covar=NULL, Iterations=5,
     Thinning=1, alpha=NULL, M=1, N=100, CPUs=1)
Fit
print(Fit)
PosteriorChecks(Fit)
caterpillar.plot(Fit, Parms="beta")
plot(Fit, BurnIn=0, MyData, PDF=FALSE)
Pred <- predict(Fit, Model, MyData, CPUs=1)
summary(Pred, Discrep="Chi-Square")
plot(Pred, Style="Covariates", Data=MyData)
plot(Pred, Style="Density", Rows=1:9)
plot(Pred, Style="ECDF")
plot(Pred, Style="Fitted")
plot(Pred, Style="Jarque-Bera")
plot(Pred, Style="Predictive Quantiles")
plot(Pred, Style="Residual Density")
plot(Pred, Style="Residuals")
Levene.Test(Pred)
Importance(Fit, Model, MyData, Discrep="Chi-Square")

#End
```

PMC.RAM *PMC RAM Estimate*

Description

This function estimates the random-access memory (RAM) required to update a given model and data with PMC.

Warning: Unwise use of this function may crash a computer, so please read the details below.

Usage

```
PMC.RAM(Model, Data, Iterations, Thinning, M, N)
```

Arguments

Model	This is a model specification function. For more information, see PMC.
Data	This is a list of Data. For more information, see PMC.

Iterations	This is the number of iterations for which PMC would update. For more information, see PMC.
Thinning	This is the amount of thinning applied to the samples in PMC.For more information, see PMC.
M	This is the number of mixture components in PMC.
N	This is the number of samples in PMC.

Details

The PMC.RAM function uses the object.size function to estimate the size in MB of RAM required to update in PMC for a given model and data, and for a number of iterations and specified thinning. When RAM is exceeded, the computer will crash. This function can be useful when trying to estimate how many samples and iterations to update a model without crashing the computer. However, when estimating the required RAM, PMC.RAM actually creates several large objects, such as post (see below). If too many iterations are given as an argument to PMC.RAM, for example, then it will crash the computer while trying to estimate the required RAM.

The best way to use this function is as follows. First, prepare the model specification and list of data. Second, observe how much RAM the computer is using at the moment, as well as the maximum available RAM. The majority of the difference of these two is the amount of RAM the computer may dedicate to updating the model. Next, use this function with a small number of iterations. Note the estimated RAM. Increase the number of iterations, and again note the RAM. Continue to increase the number of iterations until, say, arbitrarily within 90% of the above-mentioned difference in RAM.

The computer operating system uses RAM, as does any other software running at the moment. R is currently using RAM, and other functions in the LaplacesDemon package, and any other package that is currently activated, are using RAM. There are numerous small objects that are not included in the returned list, that use RAM. For example, perplexity is a small vector, etc.

A potentially large objects that is not included is a matrix used for estimating LML.

Value

PMC.RAM returns a list with several components. Each component is an estimate in MB for an object. The list has the following components:

alpha	This is the estimated size in MB of RAM required for the matrix of mixture probabilities by iteration.
Covar	This is the estimated size in MB of RAM required for the covariance matrix or matrices.
Data	This is the estimated size in MB of RAM required for the list of data.
Deviance	This is the estimated size in MB of RAM required for the deviance vector before thinning.
Initial.Values	This is the estimated size in MB of RAM required for the matrix or vector of initial values.
LH	This is the estimated size in MB of RAM required for the $N \times T \times M$ array LH, where N is the number of samples, T is the number of iterations, and M is the number of mixture components. The LH array is not returned by PMC.

LP	This is the estimated size in MB of RAM required for the $N \times T \times M$ array LP, where N is the number of samples, T is the number of iterations, and M is the number of mixture components. The LP array is not returned by PMC.
Model	This is the estimated size in MB of RAM required for the model specification function.
Monitor	This is the estimated size in MB of RAM required for the $N \times J$ matrix Monitor, where N is the number of unthinned samples and J is the number of monitored variables. Although it is thinned later in the algorithm, the full matrix is created.
Posterior1	This is the estimated size in MB of RAM required for the $N \times J \times T \times M$ array Posterior1, where N is the number of samples, J is the number of parameters, T is the number of iterations, and M is the number of mixture components.
Posterior2	This is the estimated size in MB of RAM required for the $N \times J$ matrix Posterior2, where N is the number of samples and J is the number of initial values or parameters. Although this is thinned later, at one point it is un-thinned.
Summary	This is the estimated size in MB of RAM required for the summary table.
W	This is the estimated size in MB of RAM required for the matrix of importance weights.
Total	This is the estimated size in MB of RAM required in total to update with PMC for a given model and data, and for a number of iterations, specified thinning, mixture components, and number of samples.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

BigData, LML, object.size, and PMC.

PosteriorChecks *Posterior Checks*

Description

Not to be confused with posterior predictive checks, this function provides additional information about the marginal posterior distributions of continuous parameters, such as the probability that each posterior coefficient of the parameters (referred to generically as θ), is greater than zero [$p(\theta > 0)$], the estimated number of modes, the kurtosis and skewness of the posterior distributions, the burn-in of each chain (for MCMC only), integrated autocorrelation time, independent samples per minute, and acceptance rate. A posterior correlation matrix is provided only for objects of class demonoid or pmc.

For discrete parameters, see the Hangartner.Diagnostic.

Usage

```
PosteriorChecks(x, Parms)
```

Arguments

x	This required argument accepts an object of class demonoid, iterquad, laplace, pmc, or vb.
Parms	This argument accepts a vector of quoted strings to be matched for selecting parameters. This argument defaults to NULL and selects every parameter. Each quoted string is matched to one or more parameter names with the grep function. For example, if the user specifies Parms=c("eta", "tau"), and if the parameter names are beta[1], beta[2], eta[1], eta[2], and tau, then all parameters will be selected, because the string eta is within beta. Since grep is used, string matching uses regular expressions, so beware of meta-characters, though these are acceptable: ".", "[", and "]".

Details

PosteriorChecks is a supplemental function that returns a list with two components. Following is a summary of popular uses of the PosteriorChecks function.

First (and only for MCMC users), the user may be considering the current MCMC algorithm versus others. In this case, the PosteriorChecks function is often used to find the two MCMC chains with the highest IAT, and these chains are studied for non-randomness with a joint trace plot, via the joint.density.plot function. The best algorithm has the chains with the highest independent samples per minute (ISM).

Posterior correlation may be studied between model updates as well as after a model seems to have converged. While frequentists consider multicollinear predictor variables, Bayesians tend to consider posterior correlation of the parameters. Models with multicollinear parameters take more iterations to converge. Hierarchical models often have high posterior correlations. Posterior correlation often contributes to a lower effective sample size (ESS). Common remedies include transforming the predictors, re-parameterization to reduce posterior correlation, using WIPs (Weakly-Informative Priors), or selecting a different numerical approximation algorithm. An example of re-parameterization is to constrain related parameters to sum to zero. Another approach is to specify the parameters according to a multivariate distribution that is assisted by estimating a covariance matrix. Some algorithms are more robust to posterior correlation than others. For example, posterior correlation should generally be less problematic for twalk than AMWG in LaplacesDemon. Posterior correlation may be plotted with the plotMatrix function, and may be useful for blocking parameters. For more information on blockwise sampling, see the Blocks function.

After a user is convinced of the applicability of the current MCMC algorithm, and that the chains have converged, PosteriorChecks is often used to identify multimodal marginal posterior distributions for further study or model re-specification.

Although many marginal posterior distributions appear normally distributed, there is no such assumption. Nonetheless, a marginal posterior distribution tends to be distributed the same as its prior distribution. If a parameter has a prior specified with a Laplace distribution, then the marginal posterior distribution tends also to be Laplace-distributed. In the common case of normality, kurtosis and skewness may be used to identify discrepancies between the prior and posterior, and perhaps this should be called a 'prior-posterior check'.

Lastly, parameter importance may be considered, in which case it is recommended to be considered simultaneously with variable importance from the Importance function.

Value

`PosteriorChecks` returns an object of class `posteriorchecks` that is a list with the following components:

`Posterior.Correlation`
: This is a correlation matrix of the parameters selected with the `Parms` argument. This component is returned as `NA` for objects of classes `"laplace"` or `"vb"`.

`Posterior.Summary`
: This is a matrix in which each row is a parameter and there are eight columns: p(theta > 0), N.Modes, Kurtosis, Skewness, Burn-In, IAT, ISM, and AR. The first column, p(theta > 0), indicates parameter importance by reporting how much of the distribution is greater than zero. An important parameter distribution will have a result at least as extreme as 0.025 or 0.975, and an unimportant parameter distribution is centered at 0.5. This is not the importance of the associated variable relative to how well the model fits the data. For variable importance, see the `Importance` function. The second column, N.Modes, is the number of modes, estimated with the `Modes` function. Kurtosis and skewness are useful posterior checks that may suggest that a posterior distribution is non-normal or does not fit well with a distributional assumption, assuming a distributional assumption exists, which it may not. The burn-in is estimated for each chain (only for objects of class `demonoid` with the `burnin` function. The integrated autocorrelation time is estimated with `IAT`. The number of independent samples per minute (ISM) is calculated for objects of class `"demonoid"` as `ESS` divided by minutes. Lastly, the local acceptance rate of each MCMC chain is calculated with the `AcceptanceRate` function, and is set to 1 for objects of class `iterquad`, `laplace`, `pmc`, or `vb`.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

`AcceptanceRate`, `Blocks`, `burnin`, `ESS`, `Hangartner.Diagnostic`, `joint.density.plot`, `IAT`, `Importance`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `Modes`, `plotMatrix`, `PMC`, and `VariationalBayes`.

Examples

```
### See the LaplacesDemon function for an example.
```

`Precision` *Precision*

Description

Bayesians often use precision rather than variance. These are elementary utility functions to facilitate conversions between precision, standard deviation, and variance regarding scalars, vectors, and matrices, and these functions are designed for those who are new to Bayesian inference. The names of these functions consist of two different scale parameters, separated by a '2', and capital letters refer to matrices while lower case letters refer to scalars and vectors. For example, the `Prec2Cov` function converts a precision matrix to a covariance matrix, while the `prec2sd` function converts a scalar or vector of precision parameters to standard deviation parameters.

The modern Bayesian use of precision developed because it was more straightforward in a normal distribution to estimate precision τ with a gamma distribution as a conjugate prior, than to estimate σ^2 with an inverse-gamma distribution as a conjugate prior. Today, conjugacy is usually considered to be merely a convenience, and in this example, a non-conjugate half-Cauchy prior distribution is recommended as a weakly informative prior distribution for scale parameters.

Usage

```
Cov2Prec(Cov)
Prec2Cov(Prec)
prec2sd(prec=1)
prec2var(prec=1)
sd2prec(sd=1)
sd2var(sd=1)
var2prec(var=1)
var2sd(var=1)
```

Arguments

`Cov`	This is a covariance matrix, usually represented as Σ.
`Prec`	This is a precision matrix, usually represented as Ω.
`prec`	This is a precision scalar or vector, usually represented as τ.
`sd`	This is a standard deviation scalar or vector, usually represented as σ.
`var`	This is a variance scalar or vector, usually represented as σ^2.

Details

Bayesians often use precision rather than variance, where precision is the inverse of the variance. For example, a linear regression may be represented equivalently as $\mathbf{y} \sim \mathcal{N}(\mu, \sigma^2)$, or $\mathbf{y} \sim \mathcal{N}(\mu, \tau^{-1})$, where σ^2 is the variance, and τ is the precision, which is the inverse of the variance.

Value

`Cov2Prec`	This returns a precision matrix, Ω, from a covariance matrix, Σ, where $\Omega = \Sigma^{-1}$.
`Prec2Cov`	This returns a covariance matrix, Σ, from a precision matrix, Ω, where $\Sigma = \Omega^{-1}$.
`prec2sd`	This returns a standard deviation, σ, from a precision, τ, where $\sigma = \sqrt{\tau^{-1}}$.

prec2var	This returns a variance, σ^2, from a precision, τ, where $\sigma^2 = \tau^{-1}$.
sd2prec	This returns a precision, τ, from a standard deviation, σ, where $\tau = \sigma^{-2}$.
sd2var	This returns a variance, σ^2, from a standard deviation, σ, where $\sigma^2 = \sigma\sigma$.
var2prec	This returns a precision, τ, from a variance, σ^2, where $\tau = \frac{1}{\sigma^2}$.
var2sd	This returns a standard deviation, σ, from a variance, σ^2, where $\sigma = \sqrt{\sigma^2}$.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Cov2Cor

Examples

```
library(LaplacesDemon)
Cov2Prec(matrix(c(1,0.1,0.1,1),2,2))
Prec2Cov(matrix(c(1,0.1,0.1,1),2,2))
prec2sd(0.5)
prec2var(0.5)
sd2prec(1.4142)
sd2var(01.4142)
var2prec(2)
var2sd(2)
```

---

predict.demonoid *Posterior Predictive Checks*

Description

This may be used to predict either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$), and then perform posterior predictive checks. Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is predicted given an object of class demonoid, the model specification, and data.

Usage

```
## S3 method for class 'demonoid'
predict(object, Model, Data, CPUs=1, Type="PSOCK", ...)
```

Arguments

object	An object of class demonoid is required.
Model	The model specification function is required.
Data	A data set in a list is required. The dependent variable is required to be named either y or Y.

CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".
...	Additional arguments are unused.

Details

This function passes each iteration of marginal posterior samples along with data to Model, where the fourth component in the return list is labeled yhat, and is a vector of expectations of **y**, given the samples, model specification, and data. Stationary samples are used if detected, otherwise non-stationary samples will be used. To predict $\mathbf{y}^{rep}$, simply supply the data set used to estimate the model. To predict $\mathbf{y}^{new}$, supply a new data set instead (though for some model specifications, this cannot be done, and $\mathbf{y}_{new}$ must be specified in the Model function). If the new data set does not have **y**, then create y in the list and set it equal to something sensible, such as mean(y) from the original data set.

The variable y must be a vector. If instead it is matrix Y, then it will be converted to vector y. The vectorized length of y or Y must be equal to the vectorized length of yhat, the fourth component of the return list of the Model function.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

Value

This function returns an object of class demonoid.ppc (where ppc stands for posterior predictive checks). The returned object is a list with the following components:

y	This stores the vectorized form of **y**, the dependent variable.
yhat	This is a $N \times S$ matrix, where N is the number of records of **y** and S is the number of posterior samples.
Deviance	This is a vector of predictive deviance.

Author(s)

Statisticat, LLC.

See Also

LaplacesDemon

predict.iterquad *Posterior Predictive Checks*

Description

This may be used to predict either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$), and then perform posterior predictive checks. Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is predicted given an object of class `iterquad`, the model specification, and data. This function requires that posterior samples were produced with `IterativeQuadrature`.

Usage

```
## S3 method for class 'iterquad'
predict(object, Model, Data, CPUs=1, Type="PSOCK", ...)
```

Arguments

`object`	An object of class `iterquad` is required.
`Model`	The model specification function is required.
`Data`	A data set in a list is required. The dependent variable is required to be named either `y` or `Y`.
`CPUs`	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to `CPUs=1`, in which parallel processing does not occur.
`Type`	This argument specifies the type of parallel processing to perform, accepting either `Type="PSOCK"` or `Type="MPI"`.
`...`	Additional arguments are unused.

Details

Since iterative quadrature characterizes marginal posterior distributions with means and variances, and posterior predictive checks involve samples, the `predict.iterquad` function requires the use of independent samples of the marginal posterior distributions, provided by `IterativeQuadrature` when `sir=TRUE`.

The samples of the marginal posterior distributions of the target distributions (the parameters) are passed along with the data to the `Model` specification and used to draw samples from the deviance and monitored variables. At the same time, the fourth component in the returned list, which is labeled `yhat`, is a vector of expectations of $\mathbf{y}$, given the samples, model specification, and data. To predict $\mathbf{y}^{rep}$, simply supply the data set used to estimate the model. To predict $\mathbf{y}^{new}$, supply a new data set instead (though for some model specifications, this cannot be done, and $\mathbf{y}_{new}$ must be specified in the `Model` function). If the new data set does not have $\mathbf{y}$, then create `y` in the list and set it equal to something sensible, such as `mean(y)` from the original data set.

The variable `y` must be a vector. If instead it is matrix `Y`, then it will be converted to vector `y`. The vectorized length of `y` or `Y` must be equal to the vectorized length of `yhat`, the fourth component of the returned list of the `Model` function.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

Value

This function returns an object of class iterquad.ppc (where "ppc" stands for posterior predictive checks). The returned object is a list with the following components:

y	This stores **y**, the dependent variable.
yhat	This is a $N \times S$ matrix, where N is the number of records of **y** and S is the number of posterior samples.
Deviance	This is a vector of length S, where S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, SIR.
monitor	This is a $N \times S$ matrix, where N is the number of monitored variables and S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, SIR.

Author(s)

Statisticat, LLC.

See Also

IterativeQuadrature and SIR.

---

predict.laplace *Posterior Predictive Checks*

Description

This may be used to predict either new, unobserved instances of **y** (called $\mathbf{y}^{new}$) or replicates of **y** (called $\mathbf{y}^{rep}$), and then perform posterior predictive checks. Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is predicted given an object of class laplace, the model specification, and data. This function requires that posterior samples were produced with LaplaceApproximation.

Usage

```
## S3 method for class 'laplace'
predict(object, Model, Data, CPUs=1, Type="PSOCK", ...)
```

Arguments

object	An object of class laplace is required.
Model	The model specification function is required.
Data	A data set in a list is required. The dependent variable is required to be named either y or Y.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".
...	Additional arguments are unused.

Details

Since Laplace Approximation characterizes marginal posterior distributions with modes and variances, and posterior predictive checks involve samples, the predict.laplace function requires the use of independent samples of the marginal posterior distributions, provided by LaplaceApproximation when sir=TRUE.

The samples of the marginal posterior distributions of the target distributions (the parameters) are passed along with the data to the Model specification and used to draw samples from the deviance and monitored variables. At the same time, the fourth component in the returned list, which is labeled yhat, is a vector of expectations of $\mathbf{y}$, given the samples, model specification, and data. To predict $\mathbf{y}^{rep}$, simply supply the data set used to estimate the model. To predict $\mathbf{y}^{new}$, supply a new data set instead (though for some model specifications, this cannot be done, and $\mathbf{y}_{new}$ must be specified in the Model function). If the new data set does not have $\mathbf{y}$, then create y in the list and set it equal to something sensible, such as mean(y) from the original data set.

The variable y must be a vector. If instead it is matrix Y, then it will be converted to vector y. The vectorized length of y or Y must be equal to the vectorized length of yhat, the fourth component of the returned list of the Model function.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

Value

This function returns an object of class laplace.ppc (where "ppc" stands for posterior predictive checks). The returned object is a list with the following components:

y	This stores $\mathbf{y}$, the dependent variable.
yhat	This is a $N \times S$ matrix, where N is the number of records of $\mathbf{y}$ and S is the number of posterior samples.

`Deviance` This is a vector of length S, where S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, `SIR`.

`monitor` This is a $N \times S$ matrix, where N is the number of monitored variables and S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, `SIR`.

Author(s)

Statisticat, LLC.

See Also

`LaplaceApproximation` and `SIR`.

`predict.pmc` *Posterior Predictive Checks*

Description

This may be used to predict either new, unobserved instances of **y** (called $\mathbf{y}^{new}$) or replicates of **y** (called $\mathbf{y}^{rep}$), and then perform posterior predictive checks. Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is predicted given an object of class `demonoid`, the model specification, and data.

Usage

```
## S3 method for class 'pmc'
predict(object, Model, Data, CPUs=1, Type="PSOCK", ...)
```

Arguments

`object` An object of class `pmc` is required.

`Model` The model specification function is required.

`Data` A data set in a list is required. The dependent variable is required to be named either `y` or `Y`.

`CPUs` This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to `CPUs=1`, in which parallel processing does not occur.

`Type` This argument specifies the type of parallel processing to perform, accepting either `Type="PSOCK"` or `Type="MPI"`.

`...` Additional arguments are unused.

Details

This function passes each iteration of marginal posterior samples along with data to `Model`, where the fourth component in the return list is labeled `yhat`, and is a vector of expectations of $\mathbf{y}$, given the samples, model specification, and data. Stationary samples are used if detected, otherwise non-stationary samples will be used. To predict $\mathbf{y}^{rep}$, simply supply the data set used to estimate the model. To predict $\mathbf{y}^{new}$, supply a new data set instead (though for some model specifications, this cannot be done, and $\mathbf{y}_{new}$ must be specified in the `Model` function). If the new data set does not have $\mathbf{y}$, then create `y` in the list and set it equal to something sensible, such as `mean(y)` from the original data set.

The variable `y` must be a vector. If instead it is matrix `Y`, then it will be converted to vector `y`. The vectorized length of `y` or `Y` must be equal to the vectorized length of `yhat`, the fourth component of the return list of the `Model` function.

Parallel processing may be performed when the user specifies `CPUs` to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

Value

This function returns an object of class `pmc.ppc` (where ppc stands for posterior predictive checks). The returned object is a list with the following components:

`y`	This stores the vectorized form of $\mathbf{y}$, the dependent variable.
`yhat`	This is a $N \times S$ matrix, where N is the number of records of $\mathbf{y}$ and S is the number of posterior samples.
`Deviance`	This is a vector of predictive deviance.

Author(s)

Statisticat, LLC.

See Also

PMC

predict.vb *Posterior Predictive Checks*

Description

This may be used to predict either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$), and then perform posterior predictive checks. Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is predicted given an object of class vb, the model specification, and data. This function requires that posterior samples were produced with VariationalBayes.

Usage

```
## S3 method for class 'vb'
predict(object, Model, Data, CPUs=1, Type="PSOCK", ...)
```

Arguments

object	An object of class vb is required.
Model	The model specification function is required.
Data	A data set in a list is required. The dependent variable is required to be named either y or Y.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".
...	Additional arguments are unused.

Details

Since Variational Bayes characterizes marginal posterior distributions with modes and variances, and posterior predictive checks involve samples, the predict.vb function requires the use of independent samples of the marginal posterior distributions, provided by VariationalBayes when sir=TRUE.

The samples of the marginal posterior distributions of the target distributions (the parameters) are passed along with the data to the Model specification and used to draw samples from the deviance and monitored variables. At the same time, the fourth component in the returned list, which is labeled yhat, is a vector of expectations of $\mathbf{y}$, given the samples, model specification, and data. To predict $\mathbf{y}^{rep}$, simply supply the data set used to estimate the model. To predict $\mathbf{y}^{new}$, supply a new data set instead (though for some model specifications, this cannot be done, and $\mathbf{y}_{new}$ must be specified in the Model function). If the new data set does not have $\mathbf{y}$, then create y in the list and set it equal to something sensible, such as mean(y) from the original data set.

The variable y must be a vector. If instead it is matrix Y, then it will be converted to vector y. The vectorized length of y or Y must be equal to the vectorized length of yhat, the fourth component of the returned list of the Model function.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface is used (MPI). With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

Value

This function returns an object of class vb.ppc (where "ppc" stands for posterior predictive checks). The returned object is a list with the following components:

y	This stores **y**, the dependent variable.
yhat	This is a $N \times S$ matrix, where N is the number of records of **y** and S is the number of posterior samples.
Deviance	This is a vector of length S, where S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, SIR.
monitor	This is a $N \times S$ matrix, where N is the number of monitored variables and S is the number of independent posterior samples. Samples are obtained with the sampling importance resampling algorithm, SIR.

Author(s)

Statisticat, LLC.

See Also

SIR and VariationalBayes.

---

print.demonoid *Print an object of class* demonoid *to the screen.*

Description

This may be used to print the contents of an object of class demonoid to the screen.

Usage

```
## S3 method for class 'demonoid'
print(x, ...)
```

Arguments

x	An object of class demonoid is required.
...	Additional arguments are unused.

Details

If the user has an object of class demonoid.hpc, then the print function may still be used by specifying the chain as a component in a list, such as printing the second chain with print(Fit[[2]]) when the demonoid.hpc object is named Fit, for example.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Consort, LaplacesDemon, and LaplacesDemon.hpc.

Examples

```
### See the LaplacesDemon function for an example.
```

---

print.heidelberger *Print an object of class* heidelberger *to the screen.*

Description

This may be used to print the contents of an object of class heidelberger to the screen.

Usage

```
## S3 method for class 'heidelberger'
print(x, digits=3, ...)
```

Arguments

x	An object of class heidelberger is required.
digits	This is the number of digits to print.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Heidelberger.Diagnostic.

Examples

```
### See the Heidelberger.Diagnostic function for an example.
```

print.iterquad *Print an object of class* iterquad *to the screen.*

Description

This may be used to print the contents of an object of class iterquad to the screen.

Usage

```
## S3 method for class 'iterquad'
print(x, ...)
```

Arguments

x	An object of class iterquad is required.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

IterativeQuadrature

Examples

```
### See the IterativeQuadrature function for an example.
```

print.laplace *Print an object of class* laplace *to the screen.*

Description

This may be used to print the contents of an object of class laplace to the screen.

Usage

```
## S3 method for class 'laplace'
print(x, ...)
```

Arguments

x	An object of class laplace is required.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

LaplaceApproximation

Examples

```
### See the LaplaceApproximation function for an example.
```

---

print.miss *Print an object of class* miss *to the screen.*

Description

This may be used to print the contents of an object of class miss to the screen.

Usage

```
## S3 method for class 'miss'
print(x, ...)
```

Arguments

x	An object of class miss is required.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

MISS.

Examples

```
### See the MISS function for an example.
```

print.pmc *Print an object of class* pmc *to the screen.*

Description

This may be used to print the contents of an object of class pmc to the screen.

Usage

```
## S3 method for class 'pmc'
print(x, ...)
```

Arguments

x	An object of class pmc is required.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

PMC.

Examples

```
### See the PMC function for an example.
```

print.raftery *Print an object of class* raftery *to the screen.*

Description

This may be used to print the contents of an object of class raftery to the screen.

Usage

```
## S3 method for class 'raftery'
print(x, digits=3, ...)
```

Arguments

x	An object of class raftery is required.
digits	This is the number of digits to print.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

Raftery.Diagnostic.

Examples

```
### See the Raftery.Diagnostic function for an example.
```

---

print.vb *Print an object of class* vb *to the screen.*

Description

This may be used to print the contents of an object of class vb to the screen.

Usage

```
## S3 method for class 'vb'
print(x, ...)
```

Arguments

x	An object of class vb is required.
...	Additional arguments are unused.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

VariationalBayes

Examples

```
### See the VariationalBayes function for an example.
```

`Raftery.Diagnostic` *Raftery and Lewis's diagnostic*

Description

Raftery and Lewis (1992) introduced an MCMC diagnostic that estimates the number of iterations needed for a given level of precision in posterior samples, as well as estimating burn-in, when quantiles are the posterior summaries of interest.

Usage

```
Raftery.Diagnostic(x, q=0.025, r=0.005, s=0.95, eps=0.001)
```

Arguments

`x`	This required argument accepts an object of class `demonoid`. It attempts to use `Posterior2`, but when this is missing it uses `Posterior1`.
`q`	This is the quantile to be estimated.
`r`	This is the desired margin of error of the estimate, also called the accuracy.
`s`	This is the probability of obtaining an estimate in the interval (q-r, q+r).
`eps`	This is the precision required for the estimate of time to convergence.

Details

In this MCMC diagnostic, a posterior quantile q of interest is specified. Next, an acceptable tolerance r is specified for q, which means that it is desired to measure q with an accuracy of +/- r. Finally, the user selects a probability s, which is the probability of being within the interval $(q-r, q+r)$. The `Raftery.Diagnostic` then estimates the number N of iterations and the number M of burn-in iterations that are necessary to satisfy the specified conditions regarding quantile q.

The diagnostic was designed to test a short, initial update, in which the chains were called pilot chains, and the application was later suggested for iterative use after any update as a general method for pursuing convergence (Raftery and Lewis, 1996).

Results of the `Raftery.Diagnostic` differ depending on the chosen quantile q. Estimates are conservative, so more iterations are suggested than necessary.

Value

The `Raftery.Diagnostic` function returns an object of class `raftery` that is list. A print method is available for objects of this class. The list has the following components:

`tspar`	These are the time-series parameters of the posterior samples in `x`.
`params`	This is a vector containing the parameters `q`, `r`, and `s`.
`Niters`	This is the number of iterations in the posterior samples in `x`.

`resmatrix` This is a 3-dimensional array containing the results: M is the suggested burn-in, N is the suggested number of iterations, $Nmin$ is the suggested number of iterations based on zero autocorrelation, and $I = (M + N)/Nmin$ is the "dependence factor". The dependence factor is interpreted as the proportional increase in the number of iterations attributable to autocorrelation. Highly autocorrelated chains (> 5) are worrisome, and may be due to influential initial values, parameter correlations, or poor mixing.

Note

The `Raftery.Diagnostic` function was adapted from the `raftery.diag` function in the coda package, which was adapted from the FORTRAN program 'gibbsit', written by Steven Lewis.

References

Raftery, A.E. and Lewis, S.M. (1992). "How Many Iterations in the Gibbs Sampler?" In *Bayesian Statistics*, 4 (J.M. Bernardo, J.O. Berger, A.P. Dawid and A.F.M. Smith, eds.). Oxford, U.K.: Oxford University Press, p. 763–773.

Raftery, A.E. and Lewis, S.M. (1992). "One Long Run with Diagnostics: Implementation Strategies for Markov chain Monte Carlo". *Statistical Science*, 7, p. 493–497.

Raftery, A.E. and Lewis, S.M. (1996). "Implementing MCMC". *In* Practical Markov Chain Monte Carlo (W.R. Gilks, D.J. Spiegelhalter and S. Richardson, eds.). Chapman and Hall: Baton Rouge, FL.

See Also

`burnin`, `LaplacesDemon`, `print.raftery`, and `Thin`.

Examples

```
#library(LaplacesDemon)
###After updating with LaplacesDemon, do:
#rd <- Raftery.Diagnostic(Fit)
#print(rd)
```

RejectionSampling *Rejection Sampling*

Description

The `RejectionSampling` function implements rejection sampling of a target density given a proposal density.

Usage

```
RejectionSampling(Model, Data, mu, S, df=Inf, logc, n=1000, CPUs=1, Type="PSOCK")
```

Arguments

Model	This is a model specification function. For more information, see LaplaceApproximation.
Data	This is a list of data. For more information, see LaplaceApproximation.
mu	This is a mean vector μ for the multivariate normal or multivariate t proposal density.
S	This is a convariance matrix Σ for the multivariate normal or multivariate t proposal density.
df	This is a scalar degrees of freedom parameter ν. It defaults to infinity, in which case the multivariate normal density is used.
logc	This is the logarithm of the rejection sampling constant.
n	This is the number of independent draws to be simulated from the proposal density.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".

Details

Rejection sampling (von Neumann, 1951) is a Monte Carlo method for drawing independent samples from a distribution that is proportional to the target distribution, $f(x)$, given a sampling distribution, $g(x)$, from which samples can readily be drawn, and for which there is a finite constant c.

Here, the target distribution, $f(x)$, is the result of the Model function. The sampling distribution, $g(x)$, is either a multivariate normal or multivariate t-distribution. The parameters of $g(x)$ (mu, S, and df) are used to create random draws, θ, of the sampling distribution, $g(x)$. These draws, θ, are used to evaluate the target distribution, $f(x)$, via the Model specification function. The evaluations of the target distribution, sampling distribution, and the constant are used to create a probability of acceptance for each draw, by comparing to a vector of n uniform draws, u. Each draw, θ is accepted if

$$u \leq \frac{f(\theta|\mathbf{y})}{cg(\theta)}$$

Before beginning rejection sampling, a goal of the user is to find the bounding constant, c, such that $f(\theta|\mathbf{y}) \leq cg(\theta)$ for all θ. These are all expressed in logarithms, so the goal is to find $\log f(\theta|\mathbf{y}) - \log g(\theta) \leq \log(c)$ for all θ. This is done by maximizing $\log f(\theta|\mathbf{y}) - \log g(\theta)$ over all θ. By using, say, LaplaceApproximation to find the modes of the parameters of interest, and using the resultant LP, the mode of the logarithm of the joint posterior distribution, as $\log(c)$.

The RejectionSampling function performs one iteration of rejection sampling. Rejection sampling is often iterated, then called the rejection sampling algorithm, until a sufficient number or proportion of θ is accepted. An efficient rejection sampling algorithm has a high acceptance rate. However, rejection sampling becomes less efficient as the model dimension (the number of parameters) increases.

Extensions of rejection sampling include Adaptive Rejection Sampling (ARS) (either derivative-based or derivative-free) and Adaptive Rejection Metropolis Sampling (ARMS), as in Gilks et al.

(1995). The random-walk Metropolis algorithm (Metropolis et al., 1953) combined the rejection sampling (a method of Monte Carlo simulation) of von Neumann (1951) with Markov chains.

Parallel processing may be performed when the user specifies `CPUs` to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface (MPI) is used. With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

This function is similar to the `rejectsampling` function in the `LearnBayes` package.

Value

The `RejectionSampling` function returns an object of class `rejection`, which is a matrix of accepted, independent, simulated draws from the target distribution.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

References

Gilks, W.R., Best, N.G., Tan, K.K.C. (1995). "Adaptive Rejection Metropolis Sampling within Gibbs Sampling". Journal of the Royal Statistical Society. Series C (Applied Statistics), Vol. 44, No. 4, p. 455–472.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., and Teller, E. (1953). "Equation of State Calculations by Fast Computing Machines". Journal of Chemical Physics, 21, p. 1087-1092.

von Neumann, J. (1951). "Various Techniques Used in Connection with Random Digits. Monte Carlo Methods". National Bureau Standards, 12, p. 36–38.

See Also

`dmvn`, `dmvt`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, and `VariationalBayes`.

Examples

```
library(LaplacesDemon)
### Suppose an output object of class laplace is called Fit:
#rs <- RejectionSampling(Model, MyData, mu=Fit$Summary1[,1],
#     S=Fit$Covar, df=Inf, logc=Fit$LP.Final, n=1000)
#rs
```

`SensitivityAnalysis` *Sensitivity Analysis*

Description

This function performs an elementary sensitivity analysis for two models regarding marginal posterior distributions and posterior inferences.

Usage

```
SensitivityAnalysis(Fit1, Fit2, Pred1, Pred2)
```

Arguments

`Fit1`	This argument accepts an object of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb`.
`Fit2`	This argument accepts an object of class `demonoid`, `iterquad`, `laplace`, `pmc`, or `vb`.
`Pred1`	This argument accepts an object of class `demonoid.ppc`, `iterquad.ppc`, `laplace.ppc`, `pmc.ppc`, or `vb.ppc`.
`Pred2`	This argument accepts an object of class `demonoid.ppc`, `iterquad.ppc`, `laplace.ppc`, `pmc.ppc`, or `vb.ppc`.

Details

Sensitivity analysis is concerned with the influence from changes to the inputs of a model on the output. Comparing differences resulting from different prior distributions is the most common application of sensitivity analysis, though results from different likelihoods may be compared as well. The outputs of interest are the marginal posterior distributions and posterior inferences.

There are many more methods of conducting a sensitivity analysis than exist in the `SensitivityAnalysis` function. For more information, see Oakley and O'Hagan (2004). The `SIR` function is useful for approximating changes in the posterior due to small changes in prior distributions.

The `SensitivityAnalysis` function compares marginal posterior distributions and posterior predictive distributions. Specifically, it calculates the probability that each distribution in `Fit1` and `Pred1` is greater than the associated distribution in `Fit2` and `Pred2`, and returns a variance ratio of each pair of distributions. If the probability is 0.5 that a distribution is greater than another, or if the variance ratio is 1, then no difference is found due to the inputs.

Additional comparisons and methods are currently outside the scope of the `SensitivityAnalysis` function. The `BayesFactor` function may also be considered, as well as comparing posterior predictive checks resulting from `summary.demonoid.ppc`, `summary.iterquad.ppc`, `summary.laplace.ppc`, `summary.pmc.ppc`, or `summary.vb.ppc`.

Regarding marginal posterior distributions, the `SensitivityAnalysis` function compares only distributions with identical parameter names. For example, suppose a statistician conducts a sensitivity analysis to study differences resulting from two prior distributions: a normal distribution and a Student t distribution. These distributions have two and three parameters, respectively. The statistician

has named the parameters `beta` and `sigma` for the normal distribution, while for the Student t distribution, the parameters are named `beta`, `sigma`, and `nu`. In this case, the `SensitivityAnalysis` function compares the marginal posterior distributions for `beta` and `sigma`, though `nu` is ignored because it is not in both models. If the statistician does not want certain parameters compared, then differing parameter names should be assigned.

Robust Bayesian analysis is a very similar topic, and often called simply Bayesian sensitivity analysis. In robust Bayesian analysis, the robustness of answers from a Bayesian analysis to uncertainty about the precise details of the analysis is studied. An answer is considered robust if it does not depend sensitively on the assumptions and inputs on which it is based. Robust Bayes methods acknowledge that it is sometimes very difficult to come up with precise distributions to be used as priors. Likewise the appropriate likelihood function that should be used for a particular problem may also be in doubt. In a robust Bayesian analysis, a standard Bayesian analysis is applied to all possible combinations of prior distributions and likelihood functions selected from classes of priors and likelihoods considered empirically plausible by the statistician.

Value

This function returns a list with the following components:

`Posterior`	This is a $J \times 2$ matrix of J marginal posterior distributions. Column names are "p(Fit1 > Fit2)" and "var(Fit1) / var(Fit2)".
`Post.Pred.Dist`	This is a $N \times 2$ matrix of N posterior predictive distributions. Column names are "p(Pred1 > Pred2)" and "var(Pred1) / var(Pred2)".

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Berger, J.O. (1984). "The Robust Bayesian Viewpoint (with discussion)". In J. B. Kadane, editor, Robustness of Bayesian Analyses, p. 63–144. North-Holland, Amsterdam.

Berger, J.O. (1985). "Statistical Decision Theory and Bayesian Analysis". Springer-Verlag, New York.

Berger, J.O. (1994). "An Overview of Robust Bayesian Analysis (with discussion)". Test, 3, p. 5–124.

Oakley, J. and O'Hagan, A. (2004). "Probabilistic Sensitivity Analysis of Complex Models: a Bayesian Approach". *Journal of the Royal Statistical Society, Series B*, 66, p. 751–769.

Weiss, R. (1995). "An Approach to Bayesian Sensitivity Analysis". *Journal of the Royal Statistical Society, Series B*, 58, p. 739–750.

See Also

`BayesFactor`, `IterativeQuadrature`, `LaplaceApproximation`, `LaplacesDemon`, `PMC`, `predict.demonoid`, `predict.iterquad`, `predict.laplace`, `predict.pmc`, `SIR`, `summary.demonoid.ppc`, `summary.iterquad.ppc`, `summary.laplace.ppc`, `summary.pmc.ppc`, and `VariationalBayes`.

Examples

```
#sa <- SensitivityAnalysis(Fit1, Fit2, Pred1, Pred2)
#sa
```

SIR *Sampling Importance Resampling*

Description

The SIR function performs Sampling Importance Resampling, also called Sequential Importance Resampling, and uses a multivariate normal proposal density.

Usage

```
SIR(Model, Data, mu, Sigma, n=1000, CPUs=1, Type="PSOCK")
```

Arguments

Model	This is a model specification function. For more information, see LaplaceApproximation.
Data	This is a list of data. For more information, see LaplaceApproximation.
mu	This is a mean vector, μ, for a multivariate normal distribution, and is usually the posterior means from an object of class iterquad (from IterativeQuadrature) or class vb (from VariationalBayes), or the posterior modes from an object of class laplace (from LaplaceApproximation).
Sigma	This is a covariance matrix, Σ, for a multivariate normal distribution, and is usually the Covar component of an object of class iterquad, laplace, or vb.
n	This is the number of samples to be drawn from the posterior distribution.
CPUs	This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to CPUs=1, in which parallel processing does not occur.
Type	This argument specifies the type of parallel processing to perform, accepting either Type="PSOCK" or Type="MPI".

Details

Sampling Importance Resampling (SIR) was introduced in Gordon, et al. (1993), and is the original particle filtering algorithm (and this family of algorithms is also known as Sequential Monte Carlo). A distribution is approximated with importance weights, which are approximations to the relative posterior densities of the particles, and the sum of the weights is one. In this terminology, each sample in the distribution is a "particle". SIR is a sequential or recursive form of importance sampling. As in importance sampling, the expectation of a function can be approximated as a weighted average. The optimal proposal distribution is the target distribution.

In the LaplacesDemon package, the main use of the SIR function is to produce posterior samples for iterative quadrature, Laplace Approximation, or Variational Bayes, and SIR is called behind-the-scenes by the IterativeQuadrature, LaplaceApproximation, or VariationalBayes function.

Iterative quadrature estimates the posterior mean and the associated covariance matrix. Assuming normality, this output characterizes the marginal posterior distributions. However, it is often useful to have posterior samples, in which case the SIR function is used to draw samples. The number of samples, n, should increase with the number and intercorrelations of the parameters. Otherwise, multimodal posterior distributions may occur.

Laplace Approximation estimates the posterior mode and the associated covariance matrix. Assuming normality, this output characterizes the marginal posterior distributions. However, it is often useful to have posterior samples, in which case the SIR function is used to draw samples. The number of samples, n, should increase with the number and intercorrelations of the parameters. Otherwise, multimodal posterior distributions may occur.

Variational Bayes estimates both the posterior mean and variance. Assuming normality, this output characterizes the marginal posterior distributions. However, it is often useful to have posterior samples, in which case the SIR function is used to draw samples. The number of samples, n, should increase with the number of intercorrelations of the parameters. Otherwise, multimodal posterior distributions may occur.

SIR is also commonly used when considering a mild change in a prior distribution. For example, suppose a model was updated in LaplacesDemon, and it had a least-informative prior distribution, but the statistician would like to estimate the impact of changing to a weakly-informative prior distribution. The change is made in the model specification function, and the posterior means and covariance are supplied to the SIR function. The returned samples are estimates of the posterior, given the different prior distribution. This is akin to sensitivity analysis (see the SensitivityAnalysis function).

In other contexts (for which this function is not designed), SIR is used with dynamic linear models (DLMs) and state-space models (SSMs) for state filtering.

Parallel processing may be performed when the user specifies CPUs to be greater than one, implying that the specified number of CPUs exists and is available. Parallelization may be performed on a multicore computer or a computer cluster. Either a Simple Network of Workstations (SNOW) or Message Passing Interface (MPI) is used. With small data sets and few samples, parallel processing may be slower, due to computer network communication. With larger data sets and more samples, the user should experience a faster run-time.

This function was adapted from the sir function in the LearnBayes package.

Value

The SIR function returns a matrix of samples drawn from the posterior distribution.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Gordon, N.J., Salmond, D.J., and Smith, A.F.M. (1993). "Novel Approach to Nonlinear/Non-Gaussian Bayesian State Estimation". *IEEE Proceedings F on Radar and Signal Processing*, 140(2), p. 107–113.

See Also

dmvn, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, SensitivityAnalysis, and VariationalBayes.

`Stick` *Truncated Stick-Breaking*

Description

The `Stick` function provides the utility of truncated stick-breaking regarding the vector θ. Stick-breaking is commonly referred to as a stick-breaking process, and is used often in a Dirichlet process (Sethuraman, 1994). It is commonly associated with infinite-dimensional mixtures, but in practice, the 'infinite' number is truncated to a finite number, since it is impossible to estimate an infinite number of parameters (Ishwaran and James, 2001).

Usage

```
Stick(theta)
```

Arguments

`theta`	This required argument, θ is a vector of length $(M - 1)$ regarding M mixture components.

Details

The Dirichlet process (DP) is a stochastic process used in Bayesian nonparametric modeling, most commonly in DP mixture models, otherwise known as infinite mixture models. A DP is a distribution over distributions. Each draw from a DP is itself a discrete distribution. A DP is an infinite-dimensional generalization of Dirichlet distributions. It is called a DP because it has Dirichlet-distributed, finite-dimensional, marginal distributions, just as the Gaussian process has Gaussian-distributed, finite-dimensional, marginal distributions. Distributions drawn from a DP cannot be described using a finite number of parameters, thus the classification as a nonparametric model. The truncated stick-breaking (TSB) process is associated with a truncated Dirichlet process (TDP).

An example of a TSB process is cluster analysis, where the number of clusters is unknown and treated as mixture components. In such a model, the TSB process calculates probability vector π from θ, given a user-specified maximum number of clusters to explore as C, where C is the length of $\theta + 1$. Vector π is assigned a TSB prior distribution (for more information, see `dStick`).

Elsewhere, each element of θ is constrained to the interval (0,1), and the original TSB form is beta-distributed with the α parameter of the beta distribution constrained to 1 (Ishwaran and James, 2001). The β hyperparameter in the beta distribution is usually gamma-distributed.

A larger value for a given θ_m is associated with a higher probability of the associated mixture component, however, the proportion changes according to the position of the element in the θ vector.

A variety of stick-breaking processes exist. For example, rather than each θ being beta-distributed, there have been other forms introduced such as logistic and probit, among others.

Value

The Stick function returns a probability vector wherein each element relates to a mixture component.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Ishwaran, H. and James, L. (2001). "Gibbs Sampling Methods for Stick Breaking Priors". *Journal of the American Statistical Association*, 96(453), p. 161–173.

Sethuraman, J. (1994). "A Constructive Definition of Dirichlet Priors". *Statistica Sinica*, 4, p. 639–650.

See Also

ddirichlet, dmvpolya, and dStick.

---

summary.demonoid.ppc *Posterior Predictive Check Summary*

Description

This may be used to summarize either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$). Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is summarized, depending on predict.demonoid.

Usage

```
## S3 method for class 'demonoid.ppc'
summary(object, Categorical, Rows,
     Discrep, d, Quiet, ...)
```

Arguments

object	An object of class demonoid.ppc is required.
Categorical	Logical. If TRUE, then y and yhat are considered to be categorical (such as y=0 or y=1), rather than continuous.
Rows	An optional vector of row numbers, for example c(1:10). All rows will be estimated, but only these rows will appear in the summary.
Discrep	A character string indicating a discrepancy test. Discrep defaults to NULL. Valid character strings when y is continuous are: "Chi-Square", "Chi-Square2", "Kurtosis", "L-criterion", "MASE", "MSE", "PPL", "Quadratic Loss", "Quadratic Utility", "RMSE", "Skewness", "max(yhat[i,]) > max(y)", "mean(yhat[i,]) > mean(y)", "mean(yhat[i,] > d)", "mean(yhat[i,] > mean(y))", "min(yhat[i,]) < min(y)", "round(yhat[i,]) = d", and "sd(yhat[i,]) > sd(y)". Valid character strings when y is categorical are: "p(yhat[i,] != y[i])". Kurtosis and skewness are not discrepancies, but are included here for convenience.

d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0.
Quiet	This logical argument defaults to FALSE and will print results to the console. When TRUE, results are not printed.
...	Additional arguments are unused.

Details

This function summarizes an object of class demonoid.ppc, which consists of posterior predictive checks on either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$, depending respectively on whether unobserved instances of $\mathbf{y}$ or the model sample of $\mathbf{y}$ was used in the predict.demonoid function.

The purpose of a posterior predictive check is to assess how well (or poorly) the model fits the data, or to assess discrepancies between the model and the data. For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

When $\mathbf{y}$ is continuous and known, this function estimates the predictive concordance between $\mathbf{y}$ and $\mathbf{y}^{rep}$ as per Gelfand (1996), and the predictive quantile (PQ), which is for record-level outlier detection used to calculate Gelfand's predictive concordance.

When $\mathbf{y}$ is categorical and known, this function estimates the record-level lift, which is p(yhat[i,] = y[i]) / [p(y = j) / n], or the number of correctly predicted samples over the rate of that category of $\mathbf{y}$ in vector $\mathbf{y}$.

A discrepancy measure is an approach to studying discrepancies between the model and data (Gelman et al., 1996). Below is a list of discrepancy measures, followed by a brief introduction to discrepancy analysis:

- The "Chi-Square" discrepancy measure is the chi-square goodness-of-fit test that is recommended by Gelman. For each record i=1:N, this returns (y[i] - E(y[i]))^2 / var(yhat[i,]).
- The "Chi-Square2" discrepancy measure returns the following for each record: Pr(chisq.rep[i,] > chisq.obs[i,]), where chisq.obs[i,] <- (y[i] - E(y[i]))^2 / E(y[i]), and chisq.rep[i,] <- (yhat[i,] - E(yhat[i,]))^2 / E(yhat[i,]), and the overall discrepancy is the percent of records that were outside of the 95% quantile-based probability interval (see p.interval).
- The "Kurtosis" discrepancy measure returns the kurtosis of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding kurtotic replicate distributions.
- The "L-criterion" discrepancy measure of Laud and Ibrahim (1995) provides the record-level combination of two components (see below), and the discrepancy statistic is the sum, L, as well as a calibration number, S.L. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".
- The "MASE" (Mean Absolute Scaled Error) is a discrepancy measure for the accuracy of time-series forecasts, estimated as (|y - yhat|) / mean(abs(diff(y))). The discrepancy statistic is the mean of the record-level values.
- The "MSE" (Mean Squared Error) discrepancy measure provides the MSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level MSEs. MSE and quadratic loss are identical.

- The "PPL" (Posterior Predictive Loss) discrepancy measure of Gelfand and Ghosh (1998) provides the record-level combination of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The d=0 argument applies the following weight to the accuracy component, which is then added to the variance component: $d/(d+1)$. For $\mathbf{y}^{new}$, use $d = 0$. For $\mathbf{y}^{rep}$ and model comparison, d is commonly set to 1, 10, or 100000. Larger values of d put more stress on fit and downgrade the precision of the estimates.
- The "Quadratic Loss" discrepancy measure provides the mean quadratic loss for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic losses. Quadratic loss and MSE are identical, and quadratic loss is the negative of quadratic utility.
- The "Quadratic Utility" discrepancy measure provides the mean quadratic utility for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic utilities. Quadratic utility is the negative of quadratic loss.
- The "RMSE" (Root Mean Squared Error) discrepancy measure provides the RMSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level RMSEs.
- The "Skewness" discrepancy measure returns the skewness of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding skewed replicate distributions.
- The "max(yhat[i,]) > max(y)" discrepancy measure returns a record-level indicator when a record's maximum $\mathbf{y}_i^{rep}$ exceeds the maximum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications that exceed the maximum of $\mathbf{y}$.
- The "mean(yhat[i,]) > mean(y)" discrepancy measure returns a record-level indicator when the mean of a record's $\mathbf{y}_i^{rep}$ is greater than the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with mean replications that exceed the mean of $\mathbf{y}$.
- The "mean(yhat[i,] > d)" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds a specified value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "mean(yhat[i,] > mean(y))" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level proportions.
- The "min(yhat[i,]) < min(y)" discrepancy measure returns a record-level indicator when a record's minimum $\mathbf{y}_i^{rep}$ is less than the minimum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications less than the minimum of $\mathbf{y}$.
- The "round(yhat[i,]) = d" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that, when rounded, is equal to a specified discrete value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "sd(yhat[i,]) > sd(y)" discrepancy measure returns a record-level indicator when the standard deviation of replicates is larger than the standard deviation of all of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with larger standard deviations than $\mathbf{y}$.

- The "p(yhat[i,] != y[i])" discrepancy measure returns the record-level probability that $\mathbf{y}_i^{rep}$ is not equal to **y**. This is valid when **y** is categorical and yhat is the predicted category. The probability is the proportion of replicates.

After observing a discrepancy statistic, the user attempts to improve the model by revising the model to account for discrepancies between data and the current model. This approach to model revision relies on an analysis of the discrepancy statistic. Given a discrepancy measure that is based on model fit, such as the L-criterion, the user may correlate the record-level discrepancy statistics with the dependent variable, independent variables, and interactions of independent variables. The discrepancy statistic should not correlate with the dependent and independent variables. Interaction variables may be useful for exploring new relationships that are not in the current model. Alternatively, a decision tree may be applied to the record-level discrepancy statistics, given the independent variables, in an effort to find relationships in the data that may be helpful in the model. Model revision may involve the addition of a finite mixture component to account for outliers in discrepancy, or specifying the model with a distribution that is more robust to outliers. There are too many suggestions to include here, and discrepancy analysis varies by model.

Value

This function returns a list with the following components:

`BPIC` The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$. The goal is to minimize BPIC.

`Concordance` This is the percentage of the records of y that are within the 95% quantile-based probability interval (see `p.interval`) of $\mathbf{y}^{rep}$. Gelfand's suggested goal is to achieve 95% predictive concordance. Lower percentages indicate too many outliers and a poor fit of the model to the data, and higher percentages may suggest overfitting. Concordance occurs only when **y** is continuous.

`Mean Lift` This is the mean of the record-level lifts, and occurs only when **y** is specified as categorical with `Categorical=TRUE`.

`Discrepancy.Statistic`
This is only reported if the `Discrep` argument receives a valid discrepancy measure as listed above. The `Discrep` applies to each record of **y**, and the `Discrepancy.Statistic` reports the results of the discrepancy measure on the entire data set. For example, if `Discrep="min(yhat[i,]) < min(y)"`, then the overall result is the proportion of records in which the minimum sample of yhat was less than the overall minimum **y**. This is `Pr(min(yhat[i,]) < min(y) | y, Theta)`, where `Theta` is the parameter set.

`L-criterion` The L-criterion (Laud and Ibrahim, 1995) was developed for model and variable selection. It is a sum of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The L-criterion measures model performance with a combination of how close its predictions are to the observed data and variability of the predictions. Better models have smaller values of `L`. `L` is measured in the same units as the response variable, and measures how close the data vector **y** is to the predictive

distribution. In addition to the value of L, there is a value for S.L, which is the calibration number of L, and is useful in determining how much of a decrease is necessary between models to be noteworthy.

`Summary` When **y** is continuous, this is a $N \times 8$ matrix, where N is the number of records of **y** and there are 8 columns, as follows: y, Mean, SD, LB (the 2.5% quantile), Median, UB (the 97.5% quantile), PQ (the predictive quantile, which is $Pr(\mathbf{y}^{rep} \geq \mathbf{y})$), and Test, which shows the record-level result of a test, if specified. When **y** is categorical, this matrix has a number of columns equal to the number of categories of **y** plus 3, also including y, Lift, and Discrep.

Author(s)

Statisticat, LLC.

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelfand, A. and Ghosh, S. (1998). "Model Choice: A Minimum Posterior Predictive Loss Approach". *Biometrika*, 85, p. 1–11.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

LaplacesDemon, predict.demonoid, and p.interval.

Examples

```
### See the LaplacesDemon function for an example.
```

summary.iterquad.ppc *Posterior Predictive Check Summary*

Description

This may be used to summarize either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$). Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is summarized, depending on predict.iterquad.

Usage

```
## S3 method for class 'iterquad.ppc'
summary(object, Categorical, Rows,
     Discrep, d, Quiet, ...)
```

Arguments

object	An object of class iterquad.ppc is required.
Categorical	Logical. If TRUE, then y and yhat are considered to be categorical (such as y=0 or y=1), rather than continuous.
Rows	An optional vector of row numbers, for example c(1:10). All rows will be estimated, but only these rows will appear in the summary.
Discrep	A character string indicating a discrepancy test. Discrep defaults to NULL. Valid character strings when y is continuous are: "Chi-Square", "Chi-Square2", "Kurtosis", "L-criterion", "MASE", "MSE", "PPL", "Quadratic Loss", "Quadratic Utility", "RMSE", "Skewness", "max(yhat[i,]) > max(y)", "mean(yhat[i,]) > mean(y)", "mean(yhat[i,] > d)", "mean(yhat[i,] > mean(y))", "min(yhat[i,]) < min(y)", "round(yhat[i,]) = d", and "sd(yhat[i,]) > sd(y)". Valid character strings when y is categorical are: "p(yhat[i,] != y[i])". Kurtosis and skewness are not discrepancies, but are included here for convenience.
d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0.
Quiet	This logical argument defaults to FALSE and will print results to the console. When TRUE, results are not printed.
...	Additional arguments are unused.

Details

This function summarizes an object of class iterquad.ppc, which consists of posterior predictive checks on either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$, depending respectively on whether unobserved instances of $\mathbf{y}$ or the model sample of $\mathbf{y}$ was used in the predict.iterquad function. The deviance and monitored variables are also summarized.

The purpose of a posterior predictive check is to assess how well (or poorly) the model fits the data, or to assess discrepancies between the model and the data. For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

When **y** is continuous and known, this function estimates the predictive concordance between **y** and $\mathbf{y}^{rep}$ as per Gelfand (1996), and the predictive quantile (PQ), which is for record-level outlier detection used to calculate Gelfand's predictive concordance.

When **y** is categorical and known, this function estimates the record-level lift, which is `p(yhat[i,] = y[i]) / [p(y = j) / n]`, or the number of correctly predicted samples over the rate of that category of **y** in vector **y**.

A discrepancy measure is an approach to studying discrepancies between the model and data (Gelman et al., 1996). Below is a list of discrepancy measures, followed by a brief introduction to discrepancy analysis:

- The `"Chi-Square"` discrepancy measure is the chi-square goodness-of-fit test that is recommended by Gelman. For each record i=1:N, this returns (y[i] - E(y[i]))^2 / var(yhat[i,]).
- The `"Chi-Square2"` discrepancy measure returns the following for each record: Pr(chisq.rep[i,] > chisq.obs[i,]), where chisq.obs[i,] <- (y[i] - E(y[i]))^2 / E(y[i]), and chisq.rep[i,] <- (yhat[i,] - E(yhat[i,]))^2 / E(yhat[i,]), and the overall discrepancy is the percent of records that were outside of the 95% quantile-based probability interval (see `p.interval`).
- The `"Kurtosis"` discrepancy measure returns the kurtosis of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding kurtotic replicate distributions.
- The `"L-criterion"` discrepancy measure of Laud and Ibrahim (1995) provides the record-level combination of two components (see below), and the discrepancy statistic is the sum, `L`, as well as a calibration number, `S.L`. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".
- The `"MASE"` (Mean Absolute Scaled Error) is a discrepancy measure for the accuracy of time-series forecasts, estimated as `(|y - yhat|) / mean(abs(diff(y)))`. The discrepancy statistic is the mean of the record-level values.
- The `"MSE"` (Mean Squared Error) discrepancy measure provides the MSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level MSEs. MSE and quadratic loss are identical.
- The `"PPL"` (Posterior Predictive Loss) discrepancy measure of Gelfand and Ghosh (1998) provides the record-level combination of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The `d=0` argument applies the following weight to the accuracy component, which is then added to the variance component: $d/(d+1)$. For $\mathbf{y}^{new}$, use $d = 0$. For $\mathbf{y}^{rep}$ and model comparison, d is commonly set to 1, 10, or 100000. Larger values of d put more stress on fit and downgrade the precision of the estimates.
- The `"Quadratic Loss"` discrepancy measure provides the mean quadratic loss for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic losses. Quadratic loss and MSE are identical, and quadratic loss is the negative of quadratic utility.
- The `"Quadratic Utility"` discrepancy measure provides the mean quadratic utility for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic utilities. Quadratic utility is the negative of quadratic loss.
- The `"RMSE"` (Root Mean Squared Error) discrepancy measure provides the RMSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level RMSEs.

- The "Skewness" discrepancy measure returns the skewness of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding skewed replicate distributions.
- The "max(yhat[i,]) > max(y)" discrepancy measure returns a record-level indicator when a record's maximum $\mathbf{y}_i^{rep}$ exceeds the maximum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications that exceed the maximum of $\mathbf{y}$.
- The "mean(yhat[i,]) > mean(y)" discrepancy measure returns a record-level indicator when the mean of a record's $\mathbf{y}_i^{rep}$ is greater than the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with mean replications that exceed the mean of $\mathbf{y}$.
- The "mean(yhat[i,] > d)" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds a specified value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "mean(yhat[i,] > mean(y))" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level proportions.
- The "min(yhat[i,]) < min(y)" discrepancy measure returns a record-level indicator when a record's minimum $\mathbf{y}_i^{rep}$ is less than the minimum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications less than the minimum of $\mathbf{y}$.
- The "round(yhat[i,]) = d" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that, when rounded, is equal to a specified discrete value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "sd(yhat[i,]) > sd(y)" discrepancy measure returns a record-level indicator when the standard deviation of replicates is larger than the standard deviation of all of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with larger standard deviations than $\mathbf{y}$.
- The "p(yhat[i,] != y[i])" discrepancy measure returns the record-level probability that $\mathbf{y}_i^{rep}$ is not equal to $\mathbf{y}$. This is valid when $\mathbf{y}$ is categorical and yhat is the predicted category. The probability is the proportion of replicates.

After observing a discrepancy statistic, the user attempts to improve the model by revising the model to account for discrepancies between data and the current model. This approach to model revision relies on an analysis of the discrepancy statistic. Given a discrepancy measure that is based on model fit, such as the L-criterion, the user may correlate the record-level discrepancy statistics with the dependent variable, independent variables, and interactions of independent variables. The discrepancy statistic should not correlate with the dependent and independent variables. Interaction variables may be useful for exploring new relationships that are not in the current model. Alternatively, a decision tree may be applied to the record-level discrepancy statistics, given the independent variables, in an effort to find relationships in the data that may be helpful in the model. Model revision may involve the addition of a finite mixture component to account for outliers in discrepancy, or specifying the model with a distribution that is more robust to outliers. There are too many suggestions to include here, and discrepancy analysis varies by model.

Value

This function returns a list with the following components:

`BPIC` The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$. The goal is to minimize BPIC.

`Concordance` This is the percentage of the records of y that are within the 95% quantile-based probability interval (see `p.interval`) of $\mathbf{y}^{rep}$. Gelfand's suggested goal is to achieve 95% predictive concordance. Lower percentages indicate too many outliers and a poor fit of the model to the data, and higher percentages may suggest overfitting. Concordance occurs only when **y** is continuous.

`Mean Lift` This is the mean of the record-level lifts, and occurs only when **y** is specified as categorical with `Categorical=TRUE`.

`Discrepancy.Statistic`
This is only reported if the `Discrep` argument receives a valid discrepancy measure as listed above. The `Discrep` applies to each record of **y**, and the `Discrepancy.Statistic` reports the results of the discrepancy measure on the entire data set. For example, if `Discrep="min(yhat[i,]) < min(y)"`, then the overall result is the proportion of records in which the minimum sample of yhat was less than the overall minimum **y**. This is `Pr(min(yhat[i,]) < min(y) | y, Theta)`, where `Theta` is the parameter set.

`L-criterion` The L-criterion (Laud and Ibrahim, 1995) was developed for model and variable selection. It is a sum of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The L-criterion measures model performance with a combination of how close its predictions are to the observed data and variability of the predictions. Better models have smaller values of `L`. `L` is measured in the same units as the response variable, and measures how close the data vector **y** is to the predictive distribution. In addition to the value of `L`, there is a value for `S.L`, which is the calibration number of `L`, and is useful in determining how much of a decrease is necessary between models to be noteworthy.

`Monitor` This is a $N \times 5$ matrix, where N is the number of monitored variables and there are 5 columns, as follows: Mean, SD, LB (the 2.5% quantile), Median, and UB (the 97.5% quantile).

`Summary` When **y** is continuous, this is a $N \times 8$ matrix, where N is the number of records of **y** and there are 8 columns, as follows: y, Mean, SD, LB (the 2.5% quantile), Median, UB (the 97.5% quantile), PQ (the predictive quantile, which is $Pr(\mathbf{y}^{rep} \geq \mathbf{y})$), and Test, which shows the record-level result of a test, if specified. When **y** is categorical, this matrix has a number of columns equal to the number of categories of **y** plus 3, also including `y`, `Lift`, and `Discrep`.

Author(s)

Statisticat, LLC.

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelfand, A. and Ghosh, S. (1998). "Model Choice: A Minimum Posterior Predictive Loss Approach". *Biometrika*, 85, p. 1–11.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

`IterativeQuadrature`, `predict.iterquad`, and `p.interval`.

Examples

```
### See the IterativeQuadrature function for an example.
```

`summary.laplace.ppc` *Posterior Predictive Check Summary*

Description

This may be used to summarize either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$). Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is summarized, depending on `predict.laplace`.

Usage

```
## S3 method for class 'laplace.ppc'
summary(object, Categorical, Rows, Discrep,
     d, Quiet, ...)
```

Arguments

`object`	An object of class `laplace.ppc` is required.
`Categorical`	Logical. If `TRUE`, then `y` and `yhat` are considered to be categorical (such as y=0 or y=1), rather than continuous.
`Rows`	An optional vector of row numbers, for example `c(1:10)`. All rows will be estimated, but only these rows will appear in the summary.

Discrep	A character string indicating a discrepancy test. Discrep defaults to NULL. Valid character strings when y is continuous are: "Chi-Square", "Chi-Square2", "Kurtosis", "L-criterion", "MASE", "MSE", "PPL", "Quadratic Loss", "Quadratic Utility", "RMSE", "Skewness", "max(yhat[i,]) > max(y)", "mean(yhat[i,]) > mean(y)", "mean(yhat[i,] > d)", "mean(yhat[i,] > mean(y))", "min(yhat[i,]) < min(y)", "round(yhat[i,]) = d", and "sd(yhat[i,]) > sd(y)". Valid character strings when y is categorical are: "p(yhat[i,] != y[i])". Kurtosis and skewness are not discrepancies, but are included here for convenience.
d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0.
Quiet	This logical argument defaults to FALSE and will print results to the console. When TRUE, results are not printed.
...	Additional arguments are unused.

Details

This function summarizes an object of class laplace.ppc, which consists of posterior predictive checks on either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$, depending respectively on whether unobserved instances of $\mathbf{y}$ or the model sample of $\mathbf{y}$ was used in the predict.laplace function. The deviance and monitored variables are also summarized.

The purpose of a posterior predictive check is to assess how well (or poorly) the model fits the data, or to assess discrepancies between the model and the data. For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

When $\mathbf{y}$ is continuous and known, this function estimates the predictive concordance between $\mathbf{y}$ and $\mathbf{y}^{rep}$ as per Gelfand (1996), and the predictive quantile (PQ), which is for record-level outlier detection used to calculate Gelfand's predictive concordance.

When $\mathbf{y}$ is categorical and known, this function estimates the record-level lift, which is p(yhat[i,] = y[i]) / [p(y = j) / n], or the number of correctly predicted samples over the rate of that category of $\mathbf{y}$ in vector $\mathbf{y}$.

A discrepancy measure is an approach to studying discrepancies between the model and data (Gelman et al., 1996). Below is a list of discrepancy measures, followed by a brief introduction to discrepancy analysis:

- The "Chi-Square" discrepancy measure is the chi-square goodness-of-fit test that is recommended by Gelman. For each record i=1:N, this returns (y[i] - E(y[i]))^2 / var(yhat[i,]).
- The "Chi-Square2" discrepancy measure returns the following for each record: Pr(chisq.rep[i,] > chisq.obs[i,]), where chisq.obs[i,] <- (y[i] - E(y[i]))^2 / E(y[i]), and chisq.rep[i,] <- (yhat[i,] - E(yhat[i,]))^2 / E(yhat[i,]), and the overall discrepancy is the percent of records that were outside of the 95% quantile-based probability interval (see p.interval).
- The "Kurtosis" discrepancy measure returns the kurtosis of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding kurtotic replicate distributions.
- The "L-criterion" discrepancy measure of Laud and Ibrahim (1995) provides the record-level combination of two components (see below), and the discrepancy statistic is the sum, L, as well as a calibration number, S.L. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".

- The "MASE" (Mean Absolute Scaled Error) is a discrepancy measure for the accuracy of time-series forecasts, estimated as (|y - yhat|) / mean(abs(diff(y))). The discrepancy statistic is the mean of the record-level values.
- The "MSE" (Mean Squared Error) discrepancy measure provides the MSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level MSEs. MSE and quadratic loss are identical.
- The "PPL" (Posterior Predictive Loss) discrepancy measure of Gelfand and Ghosh (1998) provides the record-level combination of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The d=0 argument applies the following weight to the accuracy component, which is then added to the variance component: $d/(d+1)$. For $\mathbf{y}^{new}$, use $d = 0$. For $\mathbf{y}^{rep}$ and model comparison, d is commonly set to 1, 10, or 100000. Larger values of d put more stress on fit and downgrade the precision of the estimates.
- The "Quadratic Loss" discrepancy measure provides the mean quadratic loss for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic losses. Quadratic loss and MSE are identical, and quadratic loss is the negative of quadratic utility.
- The "Quadratic Utility" discrepancy measure provides the mean quadratic utility for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic utilities. Quadratic utility is the negative of quadratic loss.
- The "RMSE" (Root Mean Squared Error) discrepancy measure provides the RMSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level RMSEs.
- The "Skewness" discrepancy measure returns the skewness of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding skewed replicate distributions.
- The "max(yhat[i,]) > max(y)" discrepancy measure returns a record-level indicator when a record's maximum $\mathbf{y}_i^{rep}$ exceeds the maximum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications that exceed the maximum of $\mathbf{y}$.
- The "mean(yhat[i,]) > mean(y)" discrepancy measure returns a record-level indicator when the mean of a record's $\mathbf{y}_i^{rep}$ is greater than the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with mean replications that exceed the mean of $\mathbf{y}$.
- The "mean(yhat[i,] > d)" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds a specified value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "mean(yhat[i,] > mean(y))" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level proportions.
- The "min(yhat[i,]) < min(y)" discrepancy measure returns a record-level indicator when a record's minimum $\mathbf{y}_i^{rep}$ is less than the minimum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications less than the minimum of $\mathbf{y}$.

- The "round(yhat[i,]) = d" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that, when rounded, is equal to a specified discrete value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "sd(yhat[i,]) > sd(y)" discrepancy measure returns a record-level indicator when the standard deviation of replicates is larger than the standard deviation of all of **y**. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with larger standard deviations than **y**.
- The "p(yhat[i,] != y[i])" discrepancy measure returns the record-level probability that $\mathbf{y}_i^{rep}$ is not equal to **y**. This is valid when **y** is categorical and yhat is the predicted category. The probability is the proportion of replicates.

After observing a discrepancy statistic, the user attempts to improve the model by revising the model to account for discrepancies between data and the current model. This approach to model revision relies on an analysis of the discrepancy statistic. Given a discrepancy measure that is based on model fit, such as the L-criterion, the user may correlate the record-level discrepancy statistics with the dependent variable, independent variables, and interactions of independent variables. The discrepancy statistic should not correlate with the dependent and independent variables. Interaction variables may be useful for exploring new relationships that are not in the current model. Alternatively, a decision tree may be applied to the record-level discrepancy statistics, given the independent variables, in an effort to find relationships in the data that may be helpful in the model. Model revision may involve the addition of a finite mixture component to account for outliers in discrepancy, or specifying the model with a distribution that is more robust to outliers. There are too many suggestions to include here, and discrepancy analysis varies by model.

Value

This function returns a list with the following components:

BPIC — The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$. The goal is to minimize BPIC.

Concordance — This is the percentage of the records of y that are within the 95% quantile-based probability interval (see p.interval) of $\mathbf{y}^{rep}$. Gelfand's suggested goal is to achieve 95% predictive concordance. Lower percentages indicate too many outliers and a poor fit of the model to the data, and higher percentages may suggest overfitting. Concordance occurs only when **y** is continuous.

Mean Lift — This is the mean of the record-level lifts, and occurs only when **y** is specified as categorical with Categorical=TRUE.

Discrepancy.Statistic — This is only reported if the Discrep argument receives a valid discrepancy measure as listed above. The Discrep applies to each record of **y**, and the Discrepancy.Statistic reports the results of the discrepancy measure on the entire data set. For example, if Discrep="min(yhat[i,]) < min(y)", then the overall result is the proportion of records in which the minimum sample of yhat was less than the overall minimum **y**. This is Pr(min(yhat[i,]) < min(y) | y, Theta), where Theta is the parameter set.

`L-criterion` The L-criterion (Laud and Ibrahim, 1995) was developed for model and variable selection. It is a sum of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The L-criterion measures model performance with a combination of how close its predictions are to the observed data and variability of the predictions. Better models have smaller values of `L`. `L` is measured in the same units as the response variable, and measures how close the data vector **y** is to the predictive distribution. In addition to the value of `L`, there is a value for `S.L`, which is the calibration number of `L`, and is useful in determining how much of a decrease is necessary between models to be noteworthy.

`Monitor` This is a $N \times 5$ matrix, where N is the number of monitored variables and there are 5 columns, as follows: Mean, SD, LB (the 2.5% quantile), Median, and UB (the 97.5% quantile).

`Summary` When **y** is continuous, this is a $N \times 8$ matrix, where N is the number of records of **y** and there are 8 columns, as follows: y, Mean, SD, LB (the 2.5% quantile), Median, UB (the 97.5% quantile), PQ (the predictive quantile, which is $Pr(\mathbf{y}^{rep} \geq \mathbf{y})$), and Test, which shows the record-level result of a test, if specified. When **y** is categorical, this matrix has a number of columns equal to the number of categories of **y** plus 3, also including `y`, `Lift`, and `Discrep`.

Author(s)

Statisticat, LLC.

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelfand, A. and Ghosh, S. (1998). "Model Choice: A Minimum Posterior Predictive Loss Approach". *Biometrika*, 85, p. 1–11.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

`LaplaceApproximation`, `predict.laplace`, and `p.interval`.

Examples

```
### See the LaplaceApproximation function for an example.
```

`summary.miss` *MISS Summary*

Description

This function summarizes posterior predictive distributions from an object of class `miss`.

Usage

```
## S3 method for class 'miss'
summary(object, ...)
```

Arguments

`object`	An object of class `miss` is required.
`...`	Additional arguments are unused.

Details

This function summarizes the posterior predictive distributions from an object of class `miss`.

Value

This function returns a $M \times 7$ matrix, in which each row is the posterior predictive distribution of one of M missing values. Columns are Mean, SD, MCSE, ESS, LB, Median, and UB.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

See Also

`MISS`.

Examples

```
### See the MISS function for an example.
```

summary.pmc.ppc *Posterior Predictive Check Summary*

Description

This may be used to summarize either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$). Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is summarized, depending on predict.pmc.

Usage

```
## S3 method for class 'pmc.ppc'
summary(object, Categorical, Rows,
     Discrep, d, Quiet, ...)
```

Arguments

object	An object of class pmc.ppc is required.
Categorical	Logical. If TRUE, then y and yhat are considered to be categorical (such as y=0 or y=1), rather than continuous.
Rows	An optional vector of row numbers, for example c(1:10). All rows will be estimated, but only these rows will appear in the summary.
Discrep	A character string indicating a discrepancy test. Discrep defaults to NULL. Valid character strings when y is continuous are: "Chi-Square", "Chi-Square2", "Kurtosis", "L-criterion", "MASE", "MSE", "PPL", "Quadratic Loss", "Quadratic Utility", "RMSE", "Skewness", "max(yhat[i,]) > max(y)", "mean(yhat[i,]) > mean(y)", "mean(yhat[i,] > d)", "mean(yhat[i,] > mean(y))", "min(yhat[i,]) < min(y)", "round(yhat[i,]) = d", and "sd(yhat[i,]) > sd(y)". Valid character strings when y is categorical are: "p(yhat[i,] != y[i])". Kurtosis and skewness are not discrepancies, but are included here for convenience.
d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0.
Quiet	This logical argument defaults to FALSE and will print results to the console. When TRUE, results are not printed.
...	Additional arguments are unused.

Details

This function summarizes an object of class pmc.ppc, which consists of posterior predictive checks on either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$, depending respectively on whether unobserved instances of $\mathbf{y}$ or the model sample of $\mathbf{y}$ was used in the predict.demonoid function.

The purpose of a posterior predictive check is to assess how well (or poorly) the model fits the data, or to assess discrepancies between the model and the data. For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

When **y** is continuous and known, this function estimates the predictive concordance between **y** and $\mathbf{y}^{rep}$ as per Gelfand (1996), and the predictive quantile (PQ), which is for record-level outlier detection used to calculate Gelfand's predictive concordance.

When **y** is categorical and known, this function estimates the record-level lift, which is `p(yhat[i,] = y[i]) / [p(y = j) / n]`, or the number of correctly predicted samples over the rate of that category of **y** in vector **y**.

A discrepancy measure is an approach to studying discrepancies between the model and data (Gelman et al., 1996). Below is a list of discrepancy measures, followed by a brief introduction to discrepancy analysis:

- The `"Chi-Square"` discrepancy measure is the chi-square goodness-of-fit test that is recommended by Gelman. For each record i=1:N, this returns (y[i] - E(y[i]))^2 / var(yhat[i,]).
- The `"Chi-Square2"` discrepancy measure returns the following for each record: Pr(chisq.rep[i,] > chisq.obs[i,]), where chisq.obs[i,] <- (y[i] - E(y[i]))^2 / E(y[i]), and chisq.rep[i,] <- (yhat[i,] - E(yhat[i,]))^2 / E(yhat[i,]), and the overall discrepancy is the percent of records that were outside of the 95% quantile-based probability interval (see `p.interval`).
- The `"Kurtosis"` discrepancy measure returns the kurtosis of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding kurtotic replicate distributions.
- The `"L-criterion"` discrepancy measure of Laud and Ibrahim (1995) provides the record-level combination of two components (see below), and the discrepancy statistic is the sum, `L`, as well as a calibration number, `S.L`. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".
- The `"MASE"` (Mean Absolute Scaled Error) is a discrepancy measure for the accuracy of time-series forecasts, estimated as `(|y - yhat|) / mean(abs(diff(y)))`. The discrepancy statistic is the mean of the record-level values.
- The `"MSE"` (Mean Squared Error) discrepancy measure provides the MSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level MSEs. MSE and quadratic loss are identical.
- The `"PPL"` (Posterior Predictive Loss) discrepancy measure of Gelfand and Ghosh (1998) provides the record-level combination of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The `d=0` argument applies the following weight to the accuracy component, which is then added to the variance component: $d/(d+1)$. For $\mathbf{y}^{new}$, use $d = 0$. For $\mathbf{y}^{rep}$ and model comparison, d is commonly set to 1, 10, or 100000. Larger values of d put more stress on fit and downgrade the precision of the estimates.
- The `"Quadratic Loss"` discrepancy measure provides the mean quadratic loss for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic losses. Quadratic loss and MSE are identical, and quadratic loss is the negative of quadratic utility.
- The `"Quadratic Utility"` discrepancy measure provides the mean quadratic utility for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic utilities. Quadratic utility is the negative of quadratic loss.
- The `"RMSE"` (Root Mean Squared Error) discrepancy measure provides the RMSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level RMSEs.

- The "Skewness" discrepancy measure returns the skewness of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding skewed replicate distributions.
- The "max(yhat[i,]) > max(y)" discrepancy measure returns a record-level indicator when a record's maximum $\mathbf{y}_i^{rep}$ exceeds the maximum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications that exceed the maximum of $\mathbf{y}$.
- The "mean(yhat[i,]) > mean(y)" discrepancy measure returns a record-level indicator when the mean of a record's $\mathbf{y}_i^{rep}$ is greater than the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with mean replications that exceed the mean of $\mathbf{y}$.
- The "mean(yhat[i,] > d)" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds a specified value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "mean(yhat[i,] > mean(y))" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level proportions.
- The "min(yhat[i,]) < min(y)" discrepancy measure returns a record-level indicator when a record's minimum $\mathbf{y}_i^{rep}$ is less than the minimum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications less than the minimum of $\mathbf{y}$.
- The "round(yhat[i,]) = d" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that, when rounded, is equal to a specified discrete value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "sd(yhat[i,]) > sd(y)" discrepancy measure returns a record-level indicator when the standard deviation of replicates is larger than the standard deviation of all of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with larger standard deviations than $\mathbf{y}$.
- The "p(yhat[i,] != y[i])" discrepancy measure returns the record-level probability that $\mathbf{y}_i^{rep}$ is not equal to $\mathbf{y}$. This is valid when $\mathbf{y}$ is categorical and yhat is the predicted category. The probability is the proportion of replicates.

After observing a discrepancy statistic, the user attempts to improve the model by revising the model to account for discrepancies between data and the current model. This approach to model revision relies on an analysis of the discrepancy statistic. Given a discrepancy measure that is based on model fit, such as the L-criterion, the user may correlate the record-level discrepancy statistics with the dependent variable, independent variables, and interactions of independent variables. The discrepancy statistic should not correlate with the dependent and independent variables. Interaction variables may be useful for exploring new relationships that are not in the current model. Alternatively, a decision tree may be applied to the record-level discrepancy statistics, given the independent variables, in an effort to find relationships in the data that may be helpful in the model. Model revision may involve the addition of a finite mixture component to account for outliers in discrepancy, or specifying the model with a distribution that is more robust to outliers. There are too many suggestions to include here, and discrepancy analysis varies by model.

Value

This function returns a list with the following components:

`BPIC` The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$. The goal is to minimize BPIC.

`Concordance` This is the percentage of the records of y that are within the 95% quantile-based probability interval (see `p.interval`) of $\mathbf{y}^{rep}$. Gelfand's suggested goal is to achieve 95% predictive concordance. Lower percentages indicate too many outliers and a poor fit of the model to the data, and higher percentages may suggest overfitting. Concordance occurs only when **y** is continuous.

`Mean Lift` This is the mean of the record-level lifts, and occurs only when **y** is specified as categorical with `Categorical=TRUE`.

`Discrepancy.Statistic`
This is only reported if the `Discrep` argument receives a valid discrepancy measure as listed above. The `Discrep` applies to each record of **y**, and the `Discrepancy.Statistic` reports the results of the discrepancy measure on the entire data set. For example, if `Discrep="min(yhat[i,]) < min(y)"`, then the overall result is the proportion of records in which the minimum sample of yhat was less than the overall minimum **y**. This is `Pr(min(yhat[i,]) < min(y) | y, Theta)`, where `Theta` is the parameter set.

`L-criterion` The L-criterion (Laud and Ibrahim, 1995) was developed for model and variable selection. It is a sum of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The L-criterion measures model performance with a combination of how close its predictions are to the observed data and variability of the predictions. Better models have smaller values of `L`. `L` is measured in the same units as the response variable, and measures how close the data vector **y** is to the predictive distribution. In addition to the value of `L`, there is a value for `S.L`, which is the calibration number of `L`, and is useful in determining how much of a decrease is necessary between models to be noteworthy.

`Summary` When **y** is continuous, this is a $N \times 8$ matrix, where N is the number of records of **y** and there are 8 columns, as follows: y, Mean, SD, LB (the 2.5% quantile), Median, UB (the 97.5% quantile), PQ (the predictive quantile, which is $Pr(\mathbf{y}^{rep} \geq \mathbf{y})$), and Test, which shows the record-level result of a test, if specified. When **y** is categorical, this matrix has a number of columns equal to the number of categories of **y** plus 3, also including `y`, `Lift`, and `Discrep`.

Author(s)

Statisticat, LLC.

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelfand, A. and Ghosh, S. (1998). "Model Choice: A Minimum Posterior Predictive Loss Approach". *Biometrika*, 85, p. 1–11.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

PMC, predict.pmc, and p.interval.

Examples

```
### See the PMC function for an example.
```

---

summary.vb.ppc *Posterior Predictive Check Summary*

Description

This may be used to summarize either new, unobserved instances of $\mathbf{y}$ (called $\mathbf{y}^{new}$) or replicates of $\mathbf{y}$ (called $\mathbf{y}^{rep}$). Either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$ is summarized, depending on predict.vb.

Usage

```
## S3 method for class 'vb.ppc'
summary(object, Categorical, Rows, Discrep,
     d, Quiet, ...)
```

Arguments

object	An object of class vb.ppc is required.
Categorical	Logical. If TRUE, then y and yhat are considered to be categorical (such as y=0 or y=1), rather than continuous.
Rows	An optional vector of row numbers, for example c(1:10). All rows will be estimated, but only these rows will appear in the summary.

Discrep	A character string indicating a discrepancy test. Discrep defaults to NULL. Valid character strings when y is continuous are: "Chi-Square", "Chi-Square2", "Kurtosis", "L-criterion", "MASE", "MSE", "PPL", "Quadratic Loss", "Quadratic Utility", "RMSE", "Skewness", "max(yhat[i,]) > max(y)", "mean(yhat[i,]) > mean(y)", "mean(yhat[i,] > d)", "mean(yhat[i,] > mean(y))", "min(yhat[i,]) < min(y)", "round(yhat[i,]) = d", and "sd(yhat[i,]) > sd(y)". Valid character strings when y is categorical are: "p(yhat[i,] != y[i])". Kurtosis and skewness are not discrepancies, but are included here for convenience.
d	This is an optional integer to be used with the Discrep argument above, and it defaults to d=0.
Quiet	This logical argument defaults to FALSE and will print results to the console. When TRUE, results are not printed.
...	Additional arguments are unused.

Details

This function summarizes an object of class vb.ppc, which consists of posterior predictive checks on either $\mathbf{y}^{new}$ or $\mathbf{y}^{rep}$, depending respectively on whether unobserved instances of $\mathbf{y}$ or the model sample of $\mathbf{y}$ was used in the predict.vb function. The deviance and monitored variables are also summarized.

The purpose of a posterior predictive check is to assess how well (or poorly) the model fits the data, or to assess discrepancies between the model and the data. For more information on posterior predictive checks, see https://web.archive.org/web/20150215050702/http://www.bayesian-inference.com/posteriorpredictivechecks.

When $\mathbf{y}$ is continuous and known, this function estimates the predictive concordance between $\mathbf{y}$ and $\mathbf{y}^{rep}$ as per Gelfand (1996), and the predictive quantile (PQ), which is for record-level outlier detection used to calculate Gelfand's predictive concordance.

When $\mathbf{y}$ is categorical and known, this function estimates the record-level lift, which is p(yhat[i,] = y[i]) / [p(y = j) / n], or the number of correctly predicted samples over the rate of that category of $\mathbf{y}$ in vector $\mathbf{y}$.

A discrepancy measure is an approach to studying discrepancies between the model and data (Gelman et al., 1996). Below is a list of discrepancy measures, followed by a brief introduction to discrepancy analysis:

- The "Chi-Square" discrepancy measure is the chi-square goodness-of-fit test that is recommended by Gelman. For each record i=1:N, this returns (y[i] - E(y[i]))^2 / var(yhat[i,]).
- The "Chi-Square2" discrepancy measure returns the following for each record: Pr(chisq.rep[i,] > chisq.obs[i,]), where chisq.obs[i,] <- (y[i] - E(y[i]))^2 / E(y[i]), and chisq.rep[i,] <- (yhat[i,] - E(yhat[i,]))^2 / E(yhat[i,]), and the overall discrepancy is the percent of records that were outside of the 95% quantile-based probability interval (see p.interval).
- The "Kurtosis" discrepancy measure returns the kurtosis of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding kurtotic replicate distributions.
- The "L-criterion" discrepancy measure of Laud and Ibrahim (1995) provides the record-level combination of two components (see below), and the discrepancy statistic is the sum, L, as well as a calibration number, S.L. For more information on the L-criterion, see the accompanying vignette entitled "Bayesian Inference".

- The "MASE" (Mean Absolute Scaled Error) is a discrepancy measure for the accuracy of time-series forecasts, estimated as (|y - yhat|) / mean(abs(diff(y))). The discrepancy statistic is the mean of the record-level values.
- The "MSE" (Mean Squared Error) discrepancy measure provides the MSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level MSEs. MSE and quadratic loss are identical.
- The "PPL" (Posterior Predictive Loss) discrepancy measure of Gelfand and Ghosh (1998) provides the record-level combination of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The d=0 argument applies the following weight to the accuracy component, which is then added to the variance component: $d/(d+1)$. For $\mathbf{y}^{new}$, use $d = 0$. For $\mathbf{y}^{rep}$ and model comparison, d is commonly set to 1, 10, or 100000. Larger values of d put more stress on fit and downgrade the precision of the estimates.
- The "Quadratic Loss" discrepancy measure provides the mean quadratic loss for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic losses. Quadratic loss and MSE are identical, and quadratic loss is the negative of quadratic utility.
- The "Quadratic Utility" discrepancy measure provides the mean quadratic utility for each record across all replicates, and the discrepancy statistic is the mean of the record-level mean quadratic utilities. Quadratic utility is the negative of quadratic loss.
- The "RMSE" (Root Mean Squared Error) discrepancy measure provides the RMSE for each record across all replicates, and the discrepancy statistic is the mean of the record-level RMSEs.
- The "Skewness" discrepancy measure returns the skewness of $\mathbf{y}^{rep}$ for each record, and the discrepancy statistic is the mean for all records. This does not measure discrepancies between the model and data, and is useful for finding skewed replicate distributions.
- The "max(yhat[i,]) > max(y)" discrepancy measure returns a record-level indicator when a record's maximum $\mathbf{y}_i^{rep}$ exceeds the maximum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications that exceed the maximum of $\mathbf{y}$.
- The "mean(yhat[i,]) > mean(y)" discrepancy measure returns a record-level indicator when the mean of a record's $\mathbf{y}_i^{rep}$ is greater than the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with mean replications that exceed the mean of $\mathbf{y}$.
- The "mean(yhat[i,] > d)" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds a specified value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "mean(yhat[i,] > mean(y))" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that exceeds the mean of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level proportions.
- The "min(yhat[i,]) < min(y)" discrepancy measure returns a record-level indicator when a record's minimum $\mathbf{y}_i^{rep}$ is less than the minimum of $\mathbf{y}$. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with replications less than the minimum of $\mathbf{y}$.

- The "round(yhat[i,]) = d" discrepancy measure returns a record-level proportion of $\mathbf{y}_i^{rep}$ that, when rounded, is equal to a specified discrete value, d. The discrepancy statistic is the mean of the record-level proportions.
- The "sd(yhat[i,]) > sd(y)" discrepancy measure returns a record-level indicator when the standard deviation of replicates is larger than the standard deviation of all of **y**. The discrepancy statistic is the mean of the record-level indicators, reporting the proportion of records with larger standard deviations than **y**.
- The "p(yhat[i,] != y[i])" discrepancy measure returns the record-level probability that $\mathbf{y}_i^{rep}$ is not equal to **y**. This is valid when **y** is categorical and yhat is the predicted category. The probability is the proportion of replicates.

After observing a discrepancy statistic, the user attempts to improve the model by revising the model to account for discrepancies between data and the current model. This approach to model revision relies on an analysis of the discrepancy statistic. Given a discrepancy measure that is based on model fit, such as the L-criterion, the user may correlate the record-level discrepancy statistics with the dependent variable, independent variables, and interactions of independent variables. The discrepancy statistic should not correlate with the dependent and independent variables. Interaction variables may be useful for exploring new relationships that are not in the current model. Alternatively, a decision tree may be applied to the record-level discrepancy statistics, given the independent variables, in an effort to find relationships in the data that may be helpful in the model. Model revision may involve the addition of a finite mixture component to account for outliers in discrepancy, or specifying the model with a distribution that is more robust to outliers. There are too many suggestions to include here, and discrepancy analysis varies by model.

Value

This function returns a list with the following components:

BPIC — The Bayesian Predictive Information Criterion (BPIC) was introduced by Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". $BPIC = Dbar + 2pD$. The goal is to minimize BPIC.

Concordance — This is the percentage of the records of y that are within the 95% quantile-based probability interval (see p.interval) of $\mathbf{y}^{rep}$. Gelfand's suggested goal is to achieve 95% predictive concordance. Lower percentages indicate too many outliers and a poor fit of the model to the data, and higher percentages may suggest overfitting. Concordance occurs only when **y** is continuous.

Mean Lift — This is the mean of the record-level lifts, and occurs only when **y** is specified as categorical with Categorical=TRUE.

Discrepancy.Statistic — This is only reported if the Discrep argument receives a valid discrepancy measure as listed above. The Discrep applies to each record of **y**, and the Discrepancy.Statistic reports the results of the discrepancy measure on the entire data set. For example, if Discrep="min(yhat[i,]) < min(y)", then the overall result is the proportion of records in which the minimum sample of yhat was less than the overall minimum **y**. This is Pr(min(yhat[i,]) < min(y) | y, Theta), where Theta is the parameter set.

`L-criterion` The L-criterion (Laud and Ibrahim, 1995) was developed for model and variable selection. It is a sum of two components: one involves the predictive variance and the other includes the accuracy of the means of the predictive distribution. The L-criterion measures model performance with a combination of how close its predictions are to the observed data and variability of the predictions. Better models have smaller values of `L`. `L` is measured in the same units as the response variable, and measures how close the data vector **y** is to the predictive distribution. In addition to the value of `L`, there is a value for `S.L`, which is the calibration number of `L`, and is useful in determining how much of a decrease is necessary between models to be noteworthy.

`Monitor` This is a $N \times 5$ matrix, where N is the number of monitored variables and there are 5 columns, as follows: Mean, SD, LB (the 2.5% quantile), Median, and UB (the 97.5% quantile).

`Summary` When **y** is continuous, this is a $N \times 8$ matrix, where N is the number of records of **y** and there are 8 columns, as follows: y, Mean, SD, LB (the 2.5% quantile), Median, UB (the 97.5% quantile), PQ (the predictive quantile, which is $Pr(\mathbf{y}^{rep} \geq \mathbf{y})$), and Test, which shows the record-level result of a test, if specified. When **y** is categorical, this matrix has a number of columns equal to the number of categories of **y** plus 3, also including `y`, `Lift`, and `Discrep`.

Author(s)

Statisticat, LLC.

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Gelfand, A. (1996). "Model Determination Using Sampling Based Methods". In Gilks, W., Richardson, S., Spiegehalter, D., Chapter 9 in Markov Chain Monte Carlo in Practice. Chapman and Hall: Boca Raton, FL.

Gelfand, A. and Ghosh, S. (1998). "Model Choice: A Minimum Posterior Predictive Loss Approach". *Biometrika*, 85, p. 1–11.

Gelman, A., Meng, X.L., and Stern H. (1996). "Posterior Predictive Assessment of Model Fitness via Realized Discrepancies". *Statistica Sinica*, 6, p. 733–807.

Laud, P.W. and Ibrahim, J.G. (1995). "Predictive Model Selection". *Journal of the Royal Statistical Society*, B 57, p. 247–262.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

`predict.vb`, `p.interval`, and `VariationalBayes`.

Examples

```
### See the VariationalBayes function for an example.
```

`Thin` *Thin*

Description

This function reduces the number of posterior samples by retaining every kth sample.

Usage

```
Thin(x, By=1)
```

Arguments

`x`	This is a vector or matrix of posterior samples to be thinned.
`By`	This argument specifies that every kth posterior sample will be retained, and `By` defaults to 1, meaning that thinning will not occur, because every sample will be retained.

Details

A thinned matrix of posterior samples is a matrix in which only every kth posterior sample (or row) in the original matrix is retained. The act of thinning posterior samples has been criticized as throwing away information, which is correct. However, it is common practice to thin posterior samples, usually associated with MCMC such as `LaplacesDemon`, for two reasons. First, Each chain (column vector) in a matrix of posterior samples probably has higher autocorrelation than desired, which reduces the effective sample size (see `ESS` for more information). Therefore, a thinned matrix usually contains posterior samples that are closer to independent than an un-thinned matrix. The other reason for the popularity of thinning is that it a user may not have the random-access memory (RAM) to store large, un-thinned matrices of posterior samples.

`LaplacesDemon` and `PMC` automatically thin posterior samples, deviance samples, and samples of monitored variables, according to its own user-specified argument. The `Thin` function is made available here, should it be necessary to thin posterior samples outside of objects of class `demonoid` or `pmc`.

Value

The `Thin` argument returns a thinned matrix. When `x` is a vector, the returned object is a matrix with 1 column.

Author(s)

Statisticat, LLC. `<software@bayesian-inference.com>`

See Also

ESS, LaplacesDemon, and PMC.

Examples

```
library(LaplacesDemon)
x <- matrix(runif(100), 10, 10)
Thin(x, By=2)
```

Validate *Holdout Validation*

Description

This function performs holdout validation on an object of class demonoid or pmc, given both a modeled and validation data set.

Usage

```
Validate(object, Model, Data, plot=FALSE, PDF=FALSE)
```

Arguments

object	This is an object of class demonoid or pmc.
Model	This is a model specification function for LaplacesDemon or PMC.
Data	This is a list that contains two lists of data, as specified for LaplacesDemon. The first component in the list is the list of modeled data, and the second component in the list is the list of validation data.
plot	Logical. When plot=TRUE, two plots are displayed. The upper plot shows the density of the modeled deviance in black and the density of the validation deviance in red. The lower plot shows the density of the change in deviance in gray. The plot argument defaults to FALSE.
PDF	Logical. When PDF=TRUE (and plot=TRUE), the plot is saved as a .pdf file. The PDF argument defaults to FALSE.

Details

There are numerous ways to validate a model. In this context, validation means to assess the predictive performance of a model on out-of-sample data. If reasonable, leave-one-out cross-validation (LOOCV) via the conditional predictive ordinate (CPO) should be considered when using LaplacesDemon or PMC. For more information on CPO, see the accompanying vignettes entitled "Bayesian Inference" and "Examples". CPO is unavailable when using LaplaceApproximation or VariationalBayes.

For LaplaceApproximation or VariationalBayes, it is recommended that the user perform holdout validation by comparing posterior predictive checks, comparing the differences in the specified discrepancy measure.

When LOOCV is unreasonable, popular alternatives include k-fold cross-validation and holdout validation. Although k-fold cross-validation is not performed explicitly here, the user may accomplish it with some effort. Of these methods, holdout validation includes the most bias, but is the most common in applied use, since only one model is fitted, rather than $k - 1$ models in k-fold cross-validation. The `Validate` function performs holdout validation.

For holdout validation, the observed data is sampled randomly into two data sets of approximately equal size, or three data sets that consists of two data sets of approximately equal size and a remainder data set. Of the two data sets approximately equal in size, one is called the modeled (or training) data set, and the other is called the validation (or test) data set. The modeled data set is used when updating the model. After the model is updated, both data sets are predicted in the `Validate` function, given the model. Predictive loss is estimated for the validation data set, relative to the modeled data set.

Predictive loss is associated with overfitting, differences between the model and validation data set, or model misspecification. Bayesian inference is reputed to be much more robust to overfitting than frequentist inference.

There are many ways to measure predictive loss, and within each approach, there are usually numerous possible loss functions. The log-likelihood of the model is a popular approximate utility function, and consequently, the deviance of the model is a popular loss function.

A vector of model-level (rather than record-level) deviance samples is returned with each object of class `demonoid` or `pmc`. The `Validate` function obtains this vector for each data set, and then calculates the Bayesian Predictive Information Criterion (BPIC), as per Ando (2007). BPIC is a variation of the Deviance Information Criterion (DIC) that has been modified for predictive distributions. For more information on DIC (Spiegelhalter et al., 2002), see the accompanying vignette entitled "Bayesian Inference". The goal is to minimize BPIC.

When DIC is applied after the model, such as with a predictive distribution, it is positively biased, or too small. The bias is due to the same data $\mathbf{y}$ being used both to construct the posterior distributions and to evaluate pD, the penalty term for model complexity. For example, for validation data set $\mathbf{y}_{new}$, BPIC is:

$$BPIC = -2\log[p(\mathbf{y}_{new}|\mathbf{y}, \Theta)] + 2pD$$

When `plot=TRUE`, the distributions of the modeled and validation deviances are plotted above, and the lower plot is the modeled deviance subtracted from the validation deviance. When positive, this distribution of the change in deviance is the loss in predictive deviance associated with moving from the modeled data set to the validation data set.

After using the `Validate` function, the user is encouraged to perform posterior predictive checks on each data set via the `summary.demonoid.ppc` or `summary.pmc.ppc` function.

Value

This function returns a list with three components. The first two components are also lists. Each list consists of `y`, `yhat`, and `Deviance`. The third component is a matrix that reports the expected deviance, pD, and BPIC. The object is of class `demonoid.val` for `LaplacesDemon`, or `pmc.val` when associated with `PMC`.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Ando, T. (2007). "Bayesian Predictive Information Criterion for the Evaluation of Hierarchical Bayesian and Empirical Bayes Models". *Biometrika*, 94(2), p. 443–458.

Spiegelhalter, D.J., Best, N.G., Carlin, B.P., and van der Linde, A. (2002). "Bayesian Measures of Model Complexity and Fit (with Discussion)". *Journal of the Royal Statistical Society*, B 64, p. 583–639.

See Also

LaplaceApproximation, LaplacesDemon, PMC, and VariationalBayes.

Examples

```
library(LaplacesDemon)
#Given an object called Fit of class demonoid, a Model specification,
#and a modeled data set (MyData.M) and validation data set (MyData.V):
#Validate(Fit, Model, Data=list(MyData.M=MyData.M, MyData.V=MyData.V))
```

VariationalBayes *Variational Bayes*

Description

The VariationalBayes function is a numerical approximation method for deterministically estimating the marginal posterior distributions, target distributions, in a Bayesian model with approximated distributions by minimizing the Kullback-Leibler Divergence (KLD) between the target and its approximation.

Usage

```
VariationalBayes(Model, parm, Data, Covar=NULL, Interval=1.0E-6,
     Iterations=1000, Method="Salimans2", Samples=1000, sir=TRUE,
     Stop.Tolerance=1.0E-5, CPUs=1, Type="PSOCK")
```

Arguments

Model	This required argument receives the model from a user-defined function. The user-defined function is where the model is specified. VariationalBayes passes two arguments to the model function, parms and Data. For more information, see the LaplacesDemon function and "LaplacesDemon Tutorial" vignette.
parm	This argument requires a vector of initial values equal in length to the number of parameters. VariationalBayes will attempt to optimize these initial values for the parameters, where the optimized values are the posterior means, for later use with the IterativeQuadrature, LaplacesDemon, or PMC function. The GIV function may be used to randomly generate initial values. Parameters must be continuous.

`Data` This required argument accepts a list of data. The list of data must include `mon.names` which contains monitored variable names, and `parm.names` which contains parameter names. `VariationalBayes` must be able to determine the sample size of the data, and will look for a scalar sample size variable `n` or `N`. If not found, it will look for variable `y` or `Y`, and attempt to take its number of rows as sample size. `VariationalBayes` needs to determine sample size due to the asymptotic nature of this method. Sample size should be at least $\sqrt{J}$ with J exchangeable parameters.

`Covar` This argument defaults to `NULL`, but may otherwise accept a $K \times K$ covariance matrix (where K is the number of dimensions or parameters) of the parameters. When the model is updated for the first time and prior variance or covariance is unknown, then `Covar=NULL` should be used. Once `VariationalBayes` has finished updating, it may be desired to continue updating where it left off, in which case the covariance matrix from the last run can be input into the next run.

`Interval` This argument receives an interval for estimating approximate gradients. The logarithm of the unnormalized joint posterior density of the Bayesian model is evaluated at the current parameter value, and again at the current parameter value plus this interval.

`Iterations` This argument accepts an integer that determines the number of iterations that `VariationalBayes` will attempt to maximize the logarithm of the unnormalized joint posterior density. `Iterations` defaults to 1000. `VariationalBayes` will stop before this number of iterations if the tolerance is less than or equal to the `Stop.Tolerance` criterion. The required amount of computer memory increases with `Iterations`. If computer memory is exceeded, then all will be lost.

`Method` This optional argument currently accepts only `Salimans2`, which is the second algorithm in Salimans and Knowles (2013).

`Samples` This argument indicates the number of posterior samples to be taken with sampling importance resampling via the `SIR` function, which occurs only when `sir=TRUE`. Note that the number of samples should increase with the number and intercorrelations of the parameters.

`sir` This logical argument indicates whether or not Sampling Importance Resampling (SIR) is conducted via the `SIR` function to draw independent posterior samples. This argument defaults to `TRUE`. Even when `TRUE`, posterior samples are drawn only when `VariationalBayes` has converged. Posterior samples are required for many other functions, including `plot.vb` and `predict.vb`. The only time that it is advantageous for `sir=FALSE` is when `VariationalBayes` is used to help the initial values for `IterativeQuadrature`, `LaplacesDemon`, or `PMC`, and it is unnecessary for time to be spent on sampling. Less time can be spent on sampling by increasing `CPUs`, which parallelizes the sampling.

`Stop.Tolerance` This argument accepts any positive number and defaults to 1.0E-3. Tolerance is calculated each iteration, and the criteria varies by algorithm. The algorithm is considered to have converged to the user-specified `Stop.Tolerance` when the tolerance is less than or equal to the value of `Stop.Tolerance`, and the algorithm terminates at the end of the current iteration. Often, multiple criteria are used, in which case the maximum of all criteria becomes the tolerance. For example, when partial derivatives are taken, it is commonly required that the

Euclidean norm of the partial derivatives is a criterion, and another common criterion is the Euclidean norm of the differences between the current and previous parameter values. Several algorithms have other, specific tolerances.

`CPUs` This argument accepts an integer that specifies the number of central processing units (CPUs) of the multicore computer or computer cluster. This argument defaults to `CPUs=1`, in which parallel processing does not occur. Parallelization occurs only for sampling with `SIR` when `sir=TRUE`.

`Type` This argument specifies the type of parallel processing to perform, accepting either `Type="PSOCK"` or `Type="MPI"`.

Details

Variational Bayes (VB) is a family of numerical approximation algorithms that is a subset of variational inference algorithms, or variational methods. Some examples of variational methods include the mean-field approximation, loopy belief propagation, tree-reweighted belief propagation, and expectation propagation (EP).

Variational inference for probabilistic models was introduced in the field of machine learning, influenced by statistical physics literature (Saul et al., 1996; Saul and Jordan, 1996; Jaakkola, 1997). The mean-field methods in Neal and Hinton (1999) led to variational algorithms.

Variational inference algorithms were later generalized for conjugate exponential-family models (Attias, 1999, 2000; Wiegerinck, 2000; Ghahramani and Beal, 2001; Xing et al., 2003). These algorithms still require different designs for different model forms. Salimans and Knowles (2013) introduced general-purpose VB algorithms for Gaussian posteriors.

A VB algorithm deterministically estimates the marginal posterior distributions (target distributions) in a Bayesian model with approximated distributions by minimizing the Kullback-Leibler Divergence (`KLD`) between the target and its approximation. The complicated posterior distribution is approximated with a simpler distribution. The simpler, approximated distribution is called the variational approximation, or approximation distribution, of the posterior. The term variational is derived from the calculus of variations, and regards optimization algorithms that select the best function (which is a distribution in VB), rather than merely selecting the best parameters.

VB algorithms often use Gaussian distributions as approximating distributions. In this case, both the mean and variance of the parameters are estimated.

Usually, a VB algorithm is slower to convergence than a Laplace Approximation algorithm, and faster to convergence than a Monte Carlo algorithm such as Markov chain Monte Carlo (MCMC). VB often provides solutions with comparable accuracy to MCMC in less time. Though Monte Carlo algorithms provide a numerical approximation to the exact posterior using a set of samples, VB provides a locally-optimal, exact analytical solution to an approximation of the posterior. VB is often more applicable than MCMC to big data or large-dimensional models.

Since VB is deterministic, it is asymptotic and subject to the same limitations with respect to sample size as Laplace Approximation. However, VB estimates more parameters than Laplace Approximation, such as when Laplace Approximation optimizes the posterior mode of a Gaussian distribution, while VB optimizes both the Gaussian mean and variance.

Traditionally, VB algorithms required customized equations. The `VariationalBayes` function uses general-purpose algorithms. A general-purpose VB algorithm is less efficient than an algorithm custom designed for the model form. However, a general-purpose algorithm is applied consistently and easily to numerous model forms.

When Method="Salimans2", the second algorithm of Salimans and Knowles (2013) is used. This requires the gradient and Hessian, which is more efficient with a small number of parameters as long as the posterior is twice differentiable. The step size is constant. This algorithm is suitable for marginal posterior distributions that are Gaussian and unimodal. A stochastic approximation algorithm is used in the context of fixed-form VB, inspired by considering fixed-form VB to be equivalent to performing a linear regression with the sufficient statistics of the approximation as independent variables and the unnormalized logarithm of the joint posterior density as the dependent variable. The number of requested iterations should be large, since the step-size decreases for larger requested iterations, and a small step-size will eventually converge. A large number of requested iterations results in a smaller step-size and better convergence properties, so hope for early convergence. However convergence is checked only in the last half of the iterations after the algorithm begins to average the mean and variance from the samples of the stochastic approximation. The history of stochastic samples is returned.

Value

VariationalBayes returns an object of class vb that is a list with the following components:

Call	This is the matched call of VariationalBayes.
Converged	This is a logical indicator of whether or not VariationalBayes converged within the specified Iterations according to the supplied Stop.Tolerance criterion. Convergence does not indicate that the global maximum has been found, but only that the tolerance was less than or equal to the Stop.Tolerance criterion.
Covar	This is the estimated covariance matrix. The Covar matrix may be scaled and input into the Covar argument of the LaplacesDemon or PMC function for further estimation, or the diagonal of this matrix may be used to represent the posterior variance of the parameters, provided the algorithm converged and matrix inversion was successful. To scale this matrix for use with Laplace's Demon or PMC, multiply it by $2.38^2/d$, where d is the number of initial values.
Deviance	This is a vector of the iterative history of the deviance in the VariationalBayes function, as it sought convergence.
History	This is an array of the iterative history of the parameters in the VariationalBayes function, as it sought convergence. The first matrix is for means and the second matrix is for variances.
Initial.Values	This is the vector of initial values that was originally given to VariationalBayes in the parm argument.
LML	This is an approximation of the logarithm of the marginal likelihood of the data (see the LML function for more information). When the model has converged and sir=TRUE, the NSIS method is used. When the model has converged and sir=FALSE, the LME method is used. This is the logarithmic form of equation 4 in Lewis and Raftery (1997). As a rough estimate of Kass and Raftery (1995), the LME-based LML is worrisome when the sample size of the data is less than five times the number of parameters, and LML should be adequate in most problems when the sample size of the data exceeds twenty times the number of parameters (p. 778). The LME is inappropriate with hierarchical models. However LML is estimated, it is useful for comparing multiple models with the BayesFactor function.

`LP.Final` This reports the final scalar value for the logarithm of the unnormalized joint posterior density.

`LP.Initial` This reports the initial scalar value for the logarithm of the unnormalized joint posterior density.

`Minutes` This is the number of minutes that `VariationalBayes` was running, and this includes the initial checks as well as drawing posterior samples and creating summaries.

`Monitor` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the monitored variables.

`Posterior` When `sir=TRUE`, a number of independent posterior samples equal to `Samples` is taken, and the draws are stored here as a matrix. The rows of the matrix are the samples, and the columns are the parameters.

`Step.Size.Final`
This is the final, scalar `Step.Size` value at the end of the `VariationalBayes` algorithm.

`Step.Size.Initial`
This is the initial, scalar `Step.Size`.

`Summary1` This is a summary matrix that summarizes the point-estimated posterior means and variances. Uncertainty around the posterior means is estimated from the estimated covariance matrix. Rows are parameters. The following columns are included: Mean, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Summary2` This is a summary matrix that summarizes the posterior samples drawn with sampling importance resampling (`SIR`) when `sir=TRUE`, given the point-estimated posterior means and covariance matrix. Rows are parameters. The following columns are included: Mean, SD (Standard Deviation), LB (Lower Bound), and UB (Upper Bound). The bounds constitute a 95% probability interval.

`Tolerance.Final`
This is the last `Tolerance` of the `VariationalBayes` algorithm.

`Tolerance.Stop` This is the `Stop.Tolerance` criterion.

Author(s)

Statisticat, LLC <software@bayesian-inference.com>

References

Attias, H. (1999). "Inferring Parameters and Structure of Latent Variable Models by Variational Bayes". In *Uncertainty in Artificial Intelligence*.

Attias, H. (2000). "A Variational Bayesian Framework for Graphical Models". In *Neural Information Processing Systems*.

Ghahramani, Z. and Beal, M. (2001). "Propagation Algorithms for Variational Bayesian Learning". In *Neural Information Processing Systems*, p. 507–513.

Jaakkola, T. (1997). "Variational Methods for Inference and Estimation in Graphical Models". PhD thesis, Massachusetts Institute of Technology.

Salimans, T. and Knowles, D.A. (2013). "Fixed-Form Variational Posterior Approximation through Stochastic Linear Regression". *Bayesian Analysis*, 8(4), p. 837–882.

Neal, R. and Hinton, G. (1999). "A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants". In Learning in Graphical Models, p. 355–368. MIT Press, 1999.

Saul, L. and Jordan, M. (1996). "Exploiting Tractable Substructures in Intractable Networks". *Neural Information Processing Systems*.

Saul, L., Jaakkola, T., and Jordan, M. (1996). "Mean Field Theory for Sigmoid Belief Networks". *Journal of Artificial Intelligence Research*, 4, p. 61–76.

Wiegerinck, W. (2000). "Variational Approximations Between Mean Field Theory and the Junction Tree Algorithm". In *Uncertainty in Artificial Intelligence*.

Xing, E., Jordan, M., and Russell, S. (2003). "A Generalized Mean Field Algorithm for Variational Inference in Exponential Families". In *Uncertainty in Artificial Intelligence*.

See Also

BayesFactor, IterativeQuadrature, LaplaceApproximation, LaplacesDemon, GIV, LML, PMC, and SIR.

Examples

```
# The accompanying Examples vignette is a compendium of examples.
#####################  Load the LaplacesDemon Library  #####################
library(LaplacesDemon)

#############################  Demon Data  ##############################
data(demonsnacks)
y <- log(demonsnacks$Calories)
X <- cbind(1, as.matrix(log(demonsnacks[,10]+1)))
J <- ncol(X)
for (j in 2:J) X[,j] <- CenterScale(X[,j])

########################  Data List Preparation  ########################
mon.names <- "mu[1]"
parm.names <- as.parm.names(list(beta=rep(0,J), sigma=0))
pos.beta <- grep("beta", parm.names)
pos.sigma <- grep("sigma", parm.names)
PGF <- function(Data) {
     beta <- rnorm(Data$J)
     sigma <- runif(1)
     return(c(beta, sigma))
     }
MyData <- list(J=J, PGF=PGF, X=X, mon.names=mon.names,
     parm.names=parm.names, pos.beta=pos.beta, pos.sigma=pos.sigma, y=y)

########################  Model Specification  ########################
Model <- function(parm, Data)
     {
     ### Parameters
     beta <- parm[Data$pos.beta]
     sigma <- interval(parm[Data$pos.sigma], 1e-100, Inf)
```

```
     parm[Data$pos.sigma] <- sigma
     ### Log-Prior
     beta.prior <- sum(dnormv(beta, 0, 1000, log=TRUE))
     sigma.prior <- dhalfcauchy(sigma, 25, log=TRUE)
     ### Log-Likelihood
     mu <- tcrossprod(Data$X, t(beta))
     LL <- sum(dnorm(Data$y, mu, sigma, log=TRUE))
     ### Log-Posterior
     LP <- LL + beta.prior + sigma.prior
     Modelout <- list(LP=LP, Dev=-2*LL, Monitor=mu[1],
          yhat=rnorm(length(mu), mu, sigma), parm=parm)
     return(Modelout)
     }

############################  Initial Values  ############################
#Initial.Values <- GIV(Model, MyData, PGF=TRUE)
Initial.Values <- rep(0,J+1)

#Fit <- VariationalBayes(Model, Initial.Values, Data=MyData, Covar=NULL,
#     Iterations=1000, Method="Salimans2", Stop.Tolerance=1e-3, CPUs=1)
#Fit
#print(Fit)
#PosteriorChecks(Fit)
#caterpillar.plot(Fit, Parms="beta")
#plot(Fit, MyData, PDF=FALSE)
#Pred <- predict(Fit, Model, MyData, CPUs=1)
#summary(Pred, Discrep="Chi-Square")
#plot(Pred, Style="Covariates", Data=MyData)
#plot(Pred, Style="Density", Rows=1:9)
#plot(Pred, Style="Fitted")
#plot(Pred, Style="Jarque-Bera")
#plot(Pred, Style="Predictive Quantiles")
#plot(Pred, Style="Residual Density")
#plot(Pred, Style="Residuals")
#Levene.Test(Pred)
#Importance(Fit, Model, MyData, Discrep="Chi-Square")

#Fit$Covar is scaled (2.38^2/d) and submitted to LaplacesDemon as Covar.
#Fit$Summary[,1] is submitted to LaplacesDemon as Initial.Values.
#End
```

WAIC *Widely Applicable Information Criterion*

Description

This function calculates the Widely Applicable Information Criterion (WAIC), also known as the Widely Available Information Criterion or the Watanable-Akaike, of Watanabe (2010).

Usage

```
WAIC(x)
```

Arguments

x	This required argument accepts a $N \times S$ matrix of log-likelihood (LL) for N records and S samples.

Details

WAIC is an extension of the Akaike Information Criterion (AIC) that is more fully Bayesian than the Deviance Information Criterion (DIC).

Like DIC, WAIC estimates the effective number of parameters to adjust for overfitting. Two adjustments have been proposed. pWAIC1 is similar to pD in the original DIC. In contrast, pWAIC2 is calculated with variance more similarly to pV, which Gelman proposed for DIC. Gelman et al. (2014, p.174) recommends pWAIC2 because its results are closer in practice to the results of leave-one-out cross-validation (LOO-CV). pWAIC is considered an approximation to the number of unconstrained and uninformed parameters, where a parameter counts as 1 when estimated without contraint or any prior information, 0 if fully constrained or all information comes from the prior distribution, or an intermediate number if both the data and prior are informative, which is usually the case.

Gelman et al. (2014, p. 174) scale the WAIC of Watanabe (2010) by a factor of 2 so that it is comparable to AIC and DIC. WAIC is then reported as $-2(lppd - pWAIC)$. Gelman et al. (2014) prefer WAIC to AIC or DIC when feasible, which is less often than AIC or DIC. The LaplacesDemon function requires the model specification function to return the model-level deviance, which is $-2(LL)$, where LL is the sum of the record-level log-likelihood. Therefore, if the user desires to calculate WAIC, then the record-level log-likelihood must be monitored.

Value

The WAIC argument returns a list with four components:

WAIC	This is the Widely Applicable Information Criterion (WAIC), which is $-2(lppd - pWAIC)$.
lppd	This is the log pointwise predictive density. For more information, see Gelman et al. (2014, p. 168).
pWAIC	This is the effective number of parameters preferred by Gelman et al. (2014).
pWAIC1	This is the effective number of parameters, is calculated with an alternate method, and is included here for completeness. It is not used to calculate WAIC in the WAIC function.

Author(s)

Statisticat, LLC. <software@bayesian-inference.com>

References

Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A., and Rubin, D.B. (2014). "Bayesian Data Analysis, 3rd ed.". CRC Press: Boca Raton, FL.

Watanabe, S. (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory". *Journal of Machine Learning Research*, 11, p. 3571–3594.

See Also

LaplacesDemon

Examples

```
#library(LaplacesDemon)
#N <- 10 #Number of records
#S <- 1000 #Number of samples
#LL <- t(rmvn(S, -70+rnorm(N),
#     as.positive.definite(matrix(rnorm(N*N),N,N))))
#WAIC(LL)
### Compare with DIC:
#Dev <- -2*colSums(LL)
#DIC <- list(DIC=mean(Dev) + var(Dev)/2, Dbar=mean(Dev), pV=var(Dev)/2)
#DIC
```

Index