Package: BAMBI (via r-universe)

October 26, 2024

Type Package

Title Bivariate Angular Mixture Models

Version 2.3.6

Date 2024-10-24

Maintainer Saptarshi Chakraborty <chakra.saptarshi@gmail.com>

Description Fit (using Bayesian methods) and simulate mixtures of univariate and bivariate angular distributions. Chakraborty and Wong (2021) <doi:10.18637/jss.v099.i11>.

License GPL-3

LazyData TRUE

RoxygenNote 7.3.2.9000

LinkingTo Rcpp, RcppArmadillo

Imports stats, stats4, graphics, lattice, grDevices, Rcpp, tcltk, qrng, mvtnorm, gtools, parallel, label.switching, methods, coda, future.apply, loo (>= 2.4.1), RColorBrewer, bridgesampling, scales, numDeriv

Suggests future, gridExtra

Depends R (>= 3.2.0)

URL https://doi.org/10.18637/jss.v099.i11

BugReports https://github.com/c7rishi/BAMBI/issues

Encoding UTF-8

NeedsCompilation yes

Author Saptarshi Chakraborty [aut, cre], Samuel W.K. Wong [aut]

Repository CRAN

Date/Publication 2024-10-25 13:30:02 UTC

Contents

`add_burnin_thin` *Add (extra) burnin and thin to angmcmc object after original run*

Description

Add (extra) burnin and thin to angmcmc object after original run

Usage

```
add_burnin_thin(object, burnin.prop = 0, thin = 1)
```

Arguments

`object`	angmcmc object
`burnin.prop`	proportion of iterations to used for burnin. Must be a be a number in [0, 1]. Default is 0.5.
`thin`	thining size to be used. Must be a positive integer. If `thin =` n, then every nth iteration is reatained in the final MCMC sample.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
lpdtrace(fit.vmsin.20)
# Now add extra burn-in
fit.vmsin.20.burn <- add_burnin_thin(fit.vmsin.20, 0.3)
lpdtrace(fit.vmsin.20.burn)
```

`as.mcmc.list.angmcmc` *Create an mcmc.list object from an angmcmc object*

Description

Create an mcmc.list object from an angmcmc object

Usage

```
## S3 method for class 'angmcmc'
as.mcmc.list(x, ...)
```

Arguments

x	angmcmc object
...	unused

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)

# now convert it to mcmc.list
library(coda)
fit.vmsin.20.mcmc <- as.mcmc.list(fit.vmsin.20)
```

BAMBI *BAMBI: An R package for Bivariate Angular Mixture Models*

Description

BAMBI is an R package that provides functions for fitting (using Bayesian methods) and simulating mixtures of univariate and bivariate angular distributions. Please see the reference for a detailed description of the functionalities of BAMBI.

References

Chakraborty, S., & Wong, S. W. (2021). BAMBI: An R package for fitting bivariate angular mixture models. *Journal of Statistical Software*, 99 (11), 1-69. doi:10.18637/jss.v099.i11

---

bestmodel *Convenience function for extracting angmcmc object, and the value of the model selection criterion corresponding to the best fitted model in stepwise fits*

Description

Convenience function for extracting angmcmc object, and the value of the model selection criterion corresponding to the best fitted model in stepwise fits

Usage

```
bestmodel(step_object)

bestcriterion(step_object)
```

Arguments

step_object	stepwise fitted object obtained from fit_incremental_angmix.

Details

These are convenience functions; the best fitted model and the corresponding value of model selection criterion can also be directly obtained by extracting the elements "fit.best" and "crit.best" from step_object respectively. Note that bestcriterion} returns: (a) a scalar number (class = numeric) if critused in originalfit_incremental_angmixcall is'AIC', 'BIC'or'DIC', (b) an element of class bridge "loo")ifcrit = 'WAIC', and (d) an element of class c("psis_loo", "loo")ifcrit = "LOOIC"'. See documentations of these model selection criteria for more details.

Value

bestmodel returns an angmcmc object, and bestcriterion returns the corresponding value of model selection criterion for the best fitted model in step_object.

Examples

```
# illustration only - more iterations needed for convergence
set.seed(1)
fit.vmsin.step.15 <- fit_incremental_angmix("vmsin", tim8, start_ncomp = 1,
                                            max_ncomp = 3, n.iter = 15,
                                            n.chains = 1,
                                            crit = "WAIC")
fit.vmsin.best.15 <- bestmodel(fit.vmsin.step.15)
fit.vmsin.best.15

crit.best <- bestcriterion(fit.vmsin.step.15)
crit.best
```

---

bridge_sampler.angmcmc *Log Marginal Likelihood via Bridge Sampling for angmcmc objects*

Description

Log Marginal Likelihood via Bridge Sampling for angmcmc objects

Usage

```
## S3 method for class 'angmcmc'
bridge_sampler(samples, ..., ave_over_chains = TRUE)
```

Arguments

samples	angmcmc object
...	additional argument passed to bridge_sampler. Note that default for the argument method is "warp3", (instead of "normal" as used in bridgesampling package) to account for multi-modality of the posterior density.
ave_over_chains	logical. Separately call bridge_sampler on each chain in the angmcmc object and then take the average? Defaults to TRUE. See details.

Details

Marginal likelihood is calculated by first converting the angmcmc object samples to an mcmc.list object, and then by passing the resulting mcmc.list object to bridge_sampler. If variablity across multiple chains (if any) are very different, then calling bridge_sampler separately for each chain usually provides more stable results; the final log ML is computed by averaging over chain specific MLs.

Examples

```
library(future)
library(parallel)
# plan(multisession, gc = TRUE) # parallelize chains

set.seed(100)
MC.fit <- fit_angmix("vmsin", tim8, ncomp=3, n.iter=5000,
                     n.chains = 3)


library(bridgesampling)
bridge_sampler(MC.fit)
```

---

circ_cor *Sample circular correlation coefficients*

Description

Sample circular correlation coefficients

Usage

```
circ_cor(
  x,
  type = "js",
  alternative = "two.sided",
  jackknife = FALSE,
```

```
  bootse = FALSE,
  n.boot = 100
)
```

Arguments

x	two column matrix. NA values are not allowed.
type	type of the circular correlation. Must be one of "fl", "js", "tau1" and "tau2". See details.
alternative	one of "two.sided", "less" or "greater" (defaults to "two.sided"). Hypothesis test is performed only when type is either "fl" or "js", in which case asymptotic standard error of the estimator is used to construct the test statistic.
jackknife	logical. Compute jackknifed estimate and standard error? Defaults to FALSE.
bootse	logical. Compute bootstrap standard error? Defaults to FALSE.
n.boot	number of bootstrapped samples to compute bootstrap standard error. Defaults to 100. Ignored if bootse if FALSE.

Details

circ_cor calculates the (sample) circular correlation between the columns of x. Two parametric (the Jammalamadaka-Sarma (1988, equation 2.6) form "js", and the Fisher-Lee (1983, Section 3) form "fl") and two non-parametric (two versions of Kendall's tau) correlation coefficients are considered. The first version of Kendall's tau ("tau1") is based on equation 2.1 in Fisher and Lee (1982), whereas the second version ("tau2") is computed using equations 6.7-6.8 in Zhan et al (2017).

The cost-complexity for "js", "fl", "tau2" and "tau1" are $O(n), O(n^2), O(n^2)$ and $O(n^3)$ respectively, where n denotes the number of rows in x. As such, for large n evaluation of "tau1" will be slow.

References

Fisher, N. I. and Lee, A. J. (1982). Nonparametric measures of angular-angular association. Biometrika, 69(2), 315-321.

Fisher, N. I. and Lee, A. J. (1983). A correlation coefficient for circular data. Biometrika, 70(2):327-332.

Jammalamadaka, S. R. and Sarma, Y. (1988). A correlation coefficient for angular variables. Statistical theory and data analysis II, pages 349-364.

Zhan, X., Ma, T., Liu, S., & Shimizu, K. (2017). On circular correlation for data on the torus. Statistical Papers, 1-21.

Examples

```
# generate data from vmsin model
set.seed(1)
dat <- rvmsin(100, 2,3,-0.8,0,0)

# now calculate circular correlation(s) between the 2 columns of dat
```

```
circ_cor(dat, type="js")
circ_cor(dat, type="fl")
circ_cor(dat, type="tau1")
circ_cor(dat, type="tau2")
```

`circ_varcor_model` *Analytic circular variances and correlations for bivariate angular models*

Description

Analytic circular variances and correlations for bivariate angular models

Usage

```
circ_varcor_model(
  model = "vmsin",
  kappa1 = 1,
  kappa2 = 1,
  kappa3 = 0,
  mu1 = 0,
  mu2 = 0,
  nsim = 10000,
  ...
)
```

Arguments

`model`	bivariate angular model. Must be one of `"vmsin"`, `"vmcos"`, or `"wnorm2"`.
`kappa1, kappa2, kappa3`	concentration and covariance parameters. Recycled to the same size. kappa3^2 must be < kappa1*kappa2 in the wnorm2 model (see rwnorm2 for a detailed parameterization of `wnorm2`).
`mu1, mu2`	mean parameters. Ignored as they do not play any role in the analytical formulas.
`nsim`	Monte Carlo sample size. Ignored if all of `kappa1`, `kappa2` and `abs(kappa3)` are < 150 or if model = `"wnorm2"`.
`...`	additional model specific argment

Details

The function computes the analytic circular variances and correlations (both Jammalamadaka-Sarma (JS) and Fisher-Lee (FL) forms) for von Mises sine, von Mises cosine and bivariate wrapped normal distributions.

For wnorm2, expressions for the circular variances, JS and FL correlation coefficients can be found in Mardia and Jupp (2009), Jammalamadaka and Sarma (1988) and Fisher and Lee (1983) respectively. For vmsin and vmcos these expressions are provided in Chakraborty and Wong (2018).

Because the analytic expressions in vmsin and vmcos models involve infinite sums of product of Bessel functions, if any of kappa1, kappa2 and abs(kappa3) is larger than or equal to 150, IID Monte Carlo with sample size nsim is used to approximate rho_js for numerical stability. From rho_js, rho_fl is computed using Corollary 2.2 in Chakraborty and Wong (2018), which makes cost-complexity for the rho_fl evaluation to be of order O(nsim) for vmsin and vmcos models. (In general, rho_fl evaluation is of order O(nsim^2)).

In addition, for the vmcos model, when -150 < kappa3 < -1 or 50 < max(kappa1, kappa2, abs(kappa3)) <= 150, the analytic formulas in Chakraborty and Wong (2018) are used; however, the reciprocal of the normalizing constant and its partial derivatives are all calculated numerically via (quasi) Monte carlo method for numerical stability. These (quasi) random numbers can be provided through the argument qrnd, which must be a two column matrix, with each element being a (quasi) random number between 0 and 1. Alternatively, if n_qrnd is provided (and qrnd is missing), a two dimensional sobol sequence of size n_qrnd is generated via the function sobol from the R package qrng. If none of qrnd or n_qrnd is available, a two dimensional sobol sequence of size 1e4 is used.

Value

Returns a list with elements var1, var2 (circular variances for the first and second coordinates), rho_fl and rho_js (circular correlations). See details.

References

Fisher, N. I. and Lee, A. (1983). A correlation coefficient for circular data. Biometrika, 70(2):327-332.

Jammalamadaka, S. R. and Sarma, Y. (1988). A correlation coefficient for angular variables. Statistical theory and data analysis II, pages 349-364.

Mardia, K. and Jupp, P. (2009). Directional Statistics. Wiley Series in Probability and Statistics. Wiley.

Chakraborty, S. and Wong, S, W.K. (2018). On the circular correlation coefficients for bivariate von Mises distributions on a torus. arXiv e-print.

Examples

```
circ_varcor_model("vmsin", kappa1= 1, kappa2 = 2, kappa3 = 3)

# Monte Carlo approximation
set.seed(1)
dat <- rvmsin(1000, 1, 2, 3)
# sample circular variance
circ_var <- function(x)
  1 - mean(cos(x - atan2(mean(sin(x)), mean(cos(x))) ))
circ_var(dat[, 1])
circ_var(dat[, 2])
circ_cor(dat, "fl")
circ_cor(dat, "js")
```

`contour.angmcmc` *Contour plot for angmcmc objects with bivariate data*

Description

Contour plot for angmcmc objects with bivariate data

Usage

```
## S3 method for class 'angmcmc'
contour(
  x,
  fn = "MAP",
  type = "point-est",
  show.data = TRUE,
  xpoints = seq(0, 2 * pi, length.out = 100),
  ypoints = seq(0, 2 * pi, length.out = 100),
  levels,
  nlevels = 20,
  cex = 1,
  col = "red",
  alpha = 0.4,
  pch = 19,
  ...
)
```

Arguments

`x`	angular MCMC object (with bivariate data).
`fn`	function, or a single character string specifying its name, to evaluate on MCMC samples to estimate parameters. Defaults to `mean`, which computes the estimated posterior mean. Note that if `fn = "MODE"` (warning: not `"mode"`) or `fn = "MAP"`, then the maximum aposteriori estimate (MAP) is calculated.
`type`	Passed to d_fitted. Possible choices are "point-est" and "post-pred".
`show.data`	logical. Should the data points be added to the contour plot? Ignored if `object` is NOT supplied.
`xpoints`	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
`ypoints`	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
`levels`	numeric vector of levels at which to draw contour lines; passed to the contour function in graphics.
`nlevels`	number of contour levels desired **if** levels is not supplied; passed to the contour function in graphics.

cex, col, pch	graphical parameters passed to points from graphics for plotting the data points. Ignored if show.data == FALSE.
alpha	color transparency for the data points, implemented via alpha from package scales. Ignored if show.data == FALSE.
...	additional arguments to be passed to the function contour.

Details

contour.angmcmc is an S3 function for angmcmc objects that calls contour from graphics.

To estimate the mixture density required to construct the contour plot, first the parameter vector η is estimated by applying fn on the MCMC samples, yielding the (consistent) Bayes estimate $\hat{\eta}$. Then the mixture density $f(x|\eta)$ at any point x is (consistently) estimated by $f(x|\hat{\eta})$.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# now create a contour plot
contour(fit.vmsin.20)
```

---

contour_model *Contourplot for bivariate angular mixture model densities*

Description

Contourplot for bivariate angular mixture model densities

Usage

```
contour_model(
  model = "vmsin",
  kappa1,
  kappa2,
  kappa3,
  mu1,
  mu2,
  pmix = rep(1/length(kappa1), length(kappa1)),
  xpoints = seq(0, 2 * pi, length.out = 100),
  ypoints = seq(0, 2 * pi, length.out = 100),
  levels,
  nlevels = 20,
  xlab = "x",
  ylab = "y",
  col = "black",
```

```
  lty = 1,
  main,
  ...
)
```

Arguments

model	bivariate angular model whose mixture is of interest. Must be one of "vmsin", "vmcos" and "wnorm2".
kappa1, kappa2, kappa3, mu1, mu2, pmix	model parameters and mixing proportions. See the respective mixture model densities (dvmsinmix, dvmcosmix, dwnorm2mix) for more details.
xpoints	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
ypoints	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
levels	numeric vector of levels at which to draw contour lines; passed to the contour function in graphics.
nlevels	number of contour levels desired **if** levels is not supplied; passed to the contour function in graphics.
xlab, ylab, col, lty, main	graphical parameters passed to contour.
...	additional model specific argment

Examples

```
contour_model('vmsin', 1, 1, 1.5, pi, pi)
contour_model('vmcos', 1, 1, 1.5, pi, pi)
```

---

densityplot.angmcmc *Density plots for angmcmc objects*

Description

Plot fitted angular mixture model density surfaces or curves.

Usage

```
## S3 method for class 'angmcmc'
densityplot(
  x,
  data = NULL,
  fn = mean,
  type = "point-est",
```

```
  log.density = FALSE,
  xpoints = seq(0, 2 * pi, length.out = 35),
  ypoints = seq(0, 2 * pi, length.out = 35),
  plot = TRUE,
  show.hist = ifelse(log.density, FALSE, TRUE),
  xlab,
  ylab,
  zlab = ifelse(log.density, "Log Density", "Density"),
  main,
  ...
)
```

Arguments

x	angmcmc object.
data	unused. The parameter is already filled with results from fitted angular model. It is kept to ensure compatibility with the lattice S3 generic densityplot.
fn	function, or a single character string specifying its name, to evaluate on MCMC samples to estimate parameters. Defaults to mean, which computes the estimated posterior mean. Note that if fn = "MODE" (warning: not "mode") or fn = "MAP", then the maximum aposteriori estimate (MAP) is calculated.
type	Passed to d_fitted. Possible choices are "point-est" and "post-pred".
log.density	logical. Should log density be used for the plot?
xpoints, ypoints	Points on the x and y coordinates (if bivariate) or only x coordinate (if univariate) where the density is to be evaluated. Each defaults to seq(0, 2*pi, length.out=100).
plot	logical. Should the density surface (if the fitted data is bivariate) or the density curve (if univariate) be plotted?
show.hist	logical. Should a histogram for the data points be added to the plot, if the fitted data is univariate? Ignored if data is bivariate.
xlab, ylab, zlab, main	graphical parameters passed to lattice::wireframe (if bivariate) or plot (if univariate). If the data is univariate, zlab and ylab can be used interchangeably (both correspond to the density).
...	additional arguments passed to lattice::wireframe if fitted data is bivariate, or to hist (if (show.hist == TRUE)), if the fitted data is univariate

Details

When plot==TRUE, densityplot.angmcmc calls lattice::wireframe or plot from graphics to draw the surface or curve.

To estimate the mixture density, first the parameter vector η is estimated by applying fn on the MCMC samples, yielding the (consistent) Bayes estimate $\hat{\eta}$. Then the mixture density $f(x|\eta)$ at any point x is (consistently) estimated by $f(x|\hat{\eta})$.

Note that densityplot.angmcmc **does not** plot the kernel densitie estimates of the MCMC parameters. (These plots can be obtained by first converting an angmcmc object to an mcmc object via as.mcmc.list, and then by using densplot from package coda on the resulting mcmc.list object. Instead, densityplot.angmcmc returns the surface (if 2-D) or the curve (if 1-D) of the fitted model density evaluated at the estimated parameter vector (obtain through pointest).

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# now create density surface with the default first 1/3 as burn-in and thin = 1
library(lattice)
densityplot(fit.vmsin.20)
# the viewing angles can be changed through the argument 'screen'
# (passed to lattice::wireframe)
densityplot(fit.vmsin.20, screen = list(z=-30, x=-60))
densityplot(fit.vmsin.20, screen = list(z=30, x=-60))
# the colors can be changed through 'col.regions'
cols <- grDevices::colorRampPalette(c("blue", "green",
                                      "yellow", "orange", "red"))(100)
densityplot(fit.vmsin.20, col.regions = cols)

# Now fit a vm mixture model
# illustration only - more iterations needed for convergence
fit.vm.20 <- fit_vmmix(wind$angle, ncomp = 3, n.iter =  20,
                             n.chains = 1)
densityplot(fit.vm.20)
```

DIC *Deviance Information Criterion (DIC) for angmcmc objects*

Description

Deviance Information Criterion (DIC) for angmcmc objects

Usage

```
DIC(object, form = 2, ...)
```

Arguments

object	angular MCMC object.
form	form of DIC to use. Available choices are 1 and 2 (default). See details.
...	additional model specific arguments to be passed to DIC. For example, int.displ specifies integer dispacement in wnorm and wnorm2 models. See fit_wnormmix and fit_wnorm2mix for more details.

Details

Given a deviance function $D(\theta) = -2log(p(y|\theta))$, and an estimate $\theta* = (\sum \theta_i)/N$ of the posterior mean $E(\theta|y)$, where y denote the data, θ are the unknown parameters of the model, $\theta_1, ..., \theta_N$ are MCMC samples from the posterior distribution of θ given y and $p(y|\theta)$ is the likelihood function, the (form 1 of) Deviance Infomation Criterion (DIC) is defined as

$$DIC = 2((\sum_{s=1}^{N} D(\theta_s))/N - D(\theta*))$$

The second form for DIC is given by

$$DIC = D(\theta*) - 4\hat{var} \log p(y|\theta_s)$$

where for $i = 1, ..., n$, $\hat{var} \log p(y|\theta)$ denotes the estimated variance of the log likelihood based on the realizations $\theta_1, ..., \theta_N$.

Like AIC and BIC, DIC is an asymptotic approximation for large samples, and is only valid when the posterior distribution is approximately normal.

Value

Computes the DIC for a given angmcmc object

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
DIC(fit.vmsin.20)
```

`d_fitted` *Density and random deviates from an angmcmc object*

Description

Density and random deviates from an angmcmc object

Usage

```
d_fitted(x, object, type = "point-est", fn = mean, log = FALSE, chain.no, ...)

r_fitted(n = 1, object, type = "point-est", fn = mean, chain.no, ...)
```

Arguments

x	vector, if univariate or a two column matrix, if bivariate, with each row a 2-D vector, (can also be a data frame of similar dimensions) of points where the densities are to be computed.
object	angular MCMC object. The dimension of the model must match with x.
type	Method of estimating density/generating random deviates. Possible choices are "post-pred" and "point-est". See details. Defaults to "point-est".
fn	function, or a single character string specifying its name, to evaluate on MCMC samples to estimate parameters. Defaults to mean, which computes the estimated posterior mean. Note that if fn = "MODE" (warning: not "mode") or fn = "MAP", then the maximum aposteriori estimate (MAP) is calculated.
log	logical. Should the log density be returned instead?
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
...	additional arguments to be passed to the function.
n	number of observations to be generated.

Details

If type = 'point-est', density is evaluated/random samples are generated at a point estimate of the parameter values. To estimate the mixture density, first the parameter vector η is estimated by applying fn on the MCMC samples (using the function pointest), yielding the (consistent) Bayes estimate $\hat{\eta}$. Then the mixture density $f(x|\eta)$ at any point x is (consistently) estimated by $f(x|\hat{\eta})$. The random deviates are generated from the estimated mixture density $f(x|\hat{\eta})$.

If type == 'post-pred', posterior predictive samples and densities are returned. That is, the average density $S^{-1}\sum_{s=1}^{S} f(x|\eta_s)$ is returned in d_fitted, where $\eta_1, \ldots, \eta_S$ is the set posterior MCMC samples obtained from object. In r_fitted, first a random sub-sample $\eta_{(1)}, \ldots, \eta_{(n)}$ of size n from the set of posterior samples $\eta_1, \ldots, \eta_S$ is drawn (with replacement if n > S). Then the i-th posterior predictive data point is generated from the mixture density $f(x|\eta_{(i)})$ for i = 1,..., n.

Value

d_fitted gives a vector the densities computed at the given points and r_fitted creates a vector (if univariate) or a matrix (if bivariate) with each row being a 2-D point, of random deviates.

Examples

```
set.seed(1)
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
d_fitted(c(0,0), fit.vmsin.20, type = "post-pred")
d_fitted(c(0,0), fit.vmsin.20, type = "point-est")

r_fitted(10, fit.vmsin.20, type = "post-pred")
r_fitted(10, fit.vmsin.20, type = "point-est")
```

`extractsamples` *Extract MCMC samples for parameters from an angmcmc object*

Description

Extract MCMC samples for parameters from an angmcmc object

Usage

```
extractsamples(object, par.name, comp.label, chain.no, drop = TRUE, ...)
```

Arguments

`object`	angular MCMC object
`par.name`	vector of names of parameters for which point estimates are to be computed. If `NULL`, results for all parameters are provided.
`comp.label`	vector of component labels (positive integers, e.g., `1, 2, ...`) for which point estimates are to be computed. If `NULL`, results for all components are provided.
`chain.no`	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
`drop`	logical. Should the dimension of the output be dropped, if `par.name`, `comp.label` or `chain.no` has a single level?
`...`	additional arguments to be passed to the function.

Details

The default for both `par.name` and `comp.label` are the all possible choices available in `object`.

Value

Returns a four dimensional array with

dimension 1 - model parameters and mixing proportions dimention 2 - components dimension 3 - MCMC iterations dimension 4 - chain number

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# extract Markov chain realizations for kappa1 from component 1
extr_kappa1_1 <- extractsamples(fit.vmsin.20, "kappa1", 1)
# for kappa1 from component from all components
extr_kappa1 <- extractsamples(fit.vmsin.20, "kappa1")
# for all parameters in component 1
extr_1 <- extractsamples(fit.vmsin.20, comp.label = 1)
```

fit_angmix *Fitting Bivariate and univariate angular mixture models*

Description

Fitting Bivariate and univariate angular mixture models

Usage

```
fit_angmix(
  model = "vmsin",
  data,
  ncomp,
  cov.restrict = "NONE",
  unimodal.component = FALSE,
  start_par = NULL,
  rand_start = rep(FALSE, n.chains),
  method = "hmc",
  perm_sampling = FALSE,
  n.chains = 3,
  chains_parallel = TRUE,
  return_llik_contri = FALSE,
  int.displ = 3,
  epsilon = 0.1,
  L = 10,
  epsilon.random = TRUE,
  L.random = FALSE,
  burnin.prop = 0.5,
  tune.prop = 1,
  thin = 1,
  propscale = 0.05,
  n.iter = 500,
  pmix.alpha = NULL,
  norm.var = 1000,
  autotune = TRUE,
  show.progress = TRUE,
  accpt.prob.upper,
  accpt.prob.lower,
  epsilon.incr = 0.05,
  L.incr = 0.075,
  tune.incr = 0.05,
  tune_ave_size = 100,
  kappa_upper = 150,
  kappa_lower = 1e-04,
  return_tune_param = FALSE,
  qrnd = NULL,
  n_qrnd = NULL,
```

```
  ...
)
```

Arguments

model	angular model whose mixtures are to be fitted. Available choices are "vmsin", "vmcos" and "wnorm2" for bivariate data, and "vm" and "wnorm" for univariate data.
data	data matrix (if bivarate, in which case it must have two columns) or vector. If outside, the values are transformed into the scale $[0, 2\pi)$. *Note:* BAMBI cannot handle missing data. Missing values must either be removed or properly imputed.
ncomp	number of components in the mixture model. Must be a positive integer. vector values are not allowed. If comp == 1, a single component model is fitted.
cov.restrict	Should there be any restriction on the covariance parameter for a bivariate model. Available choices are "POSITIVE", "NEGATIVE", "ZERO" and "NONE". Note that "ZERO" fits a mixture with product components. Defaults to "NONE".
unimodal.component	logical. Should each component in the mixture model be unimodal? Only used if model is either "vmsin" or "vmcos". Defaults to FALSE.
start_par	list with elements pmix (ignored if comp == 1), together with kappa1, kappa2, mu1 and mu2, for bivariate models, and kappa and mu for univariate models, all being vectors of length same as ncomp. These provides the starting values for the Markov chain; with j-th component of each vector corresponding to the j-th component of the mixture distribution. If missing, the data is first clustered into ncomp groups either via k-means (after projecting onto a unit sphere), or randomly, depending on rand_start, and then moment estimators for components are used as the starting points. Note that a very wrong starting point can potentially lead the chain to get stuck at a wrong solution for thousands of iterations. As such, we recommend using the default option, which is k-means followed by moment estimation.
rand_start	logical. Should a random starting clustering be used? Must be either a scalar, or a vector of length ncomp, one for each chain. Ignored if start_par is supplied. See start_par for more details. Defaults to FALSE.
method	MCMC strategy to be used for the model paramters: "hmc" or "rwmh".
perm_sampling	logical. Should the permutation sampling algorithm of Fruhwirth-Schnatter (2001) be used? If TRUE, at every iteration after burnin, once model parameters and mixing proportions are sampled, a random permutation of 1, ..., ncomp is considered, and components are relabelled according to this random permutation. This forced random label switchings may imporve the mixing rate of the chage. However, (automated) tuning is very difficult with such a scheme, as there is no simple way of keeping track of the "original" component labels. This creates problem with computing standard deviations of the generated model parameters, thus making the scaling step used in tuning for epsilon or paramscale problematic as well. As such, perm_sampling is always turned off during burn-in (even if autotune = FALSE), and turned on thereafter, if TRUE. Defaults to and is set to FALSE.

`n.chains` number of chains to run. Must be a positive integer.

`chains_parallel`
logical. Should the chains be run in parallel? Defaluts to TRUE, and ignored if `n.chains = 1`. Note that parallelization is implemented via future_lapply from package `future.apply` which uses futures for this purpose, and thus provides a convenient way of parallelization across various OSs and computing environments. However, a proper plan must be set for the parallization before running the chain. Otherwise the chains will run sequentially.

`return_llik_contri`
logical. Should the log likelihood contribution of each data point for each MCMC iteration in each chain be returned? This makes computation of waic.angmcmc and loo.angmcmc much faster. *Warning*: Depending on the length of data and `n.iter`, this can be very memory intensive. We suggest setting `return_llik_contri = TRUE` only if waic.angmcmc and loo.angmcmc are aimed for. Defaults to `FALSE`.

`int.displ` absolute integer displacement for each coordinate for `wnorm` and `wnorm2` models (ignored otherwise). Default is 3. Allowed minimum and maximum are 1 and 5 respectively.

`epsilon, L` tuning parameters for HMC; ignored if `method = "rwmh"`. `epsilon` (step-size) is a single number, or a vector of size `2*ncomp` for univariate models and `5*ncomp` for bivariate models. Note that the "mass matrix" in HMC is assumed to be identity. As such, `epsilon`'s corresponding to different model parameters need to be in proper scale for optimal acceptance rate. Can be autotuned during burnin. See `autotune`. `L` (leapfrog steps) is a positive integer or a vector of positive integers of length `n.chains`. If multiple chains are used, we suggest same `L` values acorss different chains to make the chains as homogenous as possible.

`epsilon.random` logical. Should `epsilon*delta`, where `delta` is a random number between `(1-epsilon.incr, 1+epsilon.incr)` be used instead of `epsilon` at each iteration? Ignored if `method = "rwmh"`.

`L.random` logical. Should a random integer between `L.orig/exp(L.incr)` and `L.orig*exp(L.incr)`be used instead as `L` at each iteration? Ignored if `method = "rwmh"`. Defaults to `TRUE`.

`burnin.prop` proportion of iterations to used for burnin. Must be a be a number in [0, 1]. Default is 0.5.

`tune.prop` proportion of *`burnin`* used to tune the parameters (`epsilon` in HMC and `propscale` in RWMH). Must be a number between 0 and 1; defaults to 1. Ignored if `autotune == FALSE`.

`thin` thining size to be used. Must be a positive integer. If `thin =` n, then every nth iteration is reatained in the final MCMC sample.

`propscale` tuning parameters for RWMH; a vector of size 5 (for bivariate models) or 2 (for univariate models) representing the variances for the proposal normal densities for the model parameters. Ignored if `method = "hmc"`. Can be autotuned during burnin. See `autotune`.

`n.iter` number of iterations for the Markov Chain.

`pmix.alpha` concentration parameter(s) for the Dirichlet prior for `pmix`. Must either be a positive real number, or a vector with positive entries and of length `ncomp`. The default is $(r+r(r+1)/2)/2+3$, where r is 1 or 2 according as whether the model is univariate or bivariate. Note that it is recommended to use larger `alpha` values to ensure the a good posterior behavior, especially when fit_incremental_angmix is used for model selection, which handles overfitting in "let two component-specific parameters be size, and then penalizes for model complexity. See Fruhwirth-Schnatter (2011) for more details on this.

`norm.var` variance (hyper-) parameters in the normal prior for `log(kappa)`, `log(kappa1)`, `log(kappa2)` and `kappa3`. (Prior mean is zero). Can be a vector. Default is 1000 that makes the prior non-informative.

`autotune` logical. Should the Markov chain auto-tune the parameter `epsilon` (in HMC) or `propscale` (in RWMH) during burn-in? Set to `TRUE` by default. An adaptive tuning strategy is implemented. Here, at every 10th iteration during in burn-in, the acceptance ratio in the last `tune_ave_size` iterations is calculated. Then the tuning parameter is decreased (increased) by a factor of `1-tune.incr` (`1+tune.incr`) if the calculated acceptance rate falls below (above) `accpt.prob.lower` (`accpt.prob.upper`). In addditon, when `iter` is a multiple of `tune_ave_size`, `epsilon` for each model parameter is rescaled via the standard deviation of the corresponding parameter over the past `tune_ave_size` iterations.

`show.progress` logical. Should a progress bar be displayed?

`accpt.prob.lower`, `accpt.prob.upper`
lower and upper limits of acceptance ratio to be maintained while tuning during burn-in. Must be numbers between 0 and 1, which `accpt.prob.lower` < `accpt.prob.upper`. See `autotune`. Default to (0.6, 0,9) for HMC and (0.3, 0.5) for RWMH. Ignored if `autotune = FALSE`.

`epsilon.incr` amount of randomness incorporated in `epsilon` if `epsilon.random = TRUE`.

`L.incr` amount of randomness incorporated in L if `L.random = TRUE`.

`tune.incr` how much should the tuning parameter be increased or decreased at each step while tuning during burn-in? Must be a number between 0 and 1. See `autotune`. Defaults to 0.05. Ignored if `autotune = FALSE`.

`tune_ave_size` number previous iterations used to compute the acceptance rate while tuning in burn-in. Must be a positive integer. Defaults to 100.

`kappa_upper`, `kappa_lower`
upper and lower bounds for the concentration and (absolute) association parameters. Must be a positive integers. Defaults to 150 and 1e-4, and parameter with value above or below these limits rarely make sense in practice. Warning: values much larger or smaller than the default are not recommended as they can cause numerical instability.

`return_tune_param`
logical. Should the values of the tuning parameters used at each iteration in each chain be returned? Defaults to `FALSE`.

`qrnd`, `n_qrnd` Used only if `method="vmcos"`. See dvmcos for details.

`...` Unused.

Note

Sampling is done in log scale for the concentration parameters (kappa, kappa1 and kappa2).

Parallelization is done by default when more than one chain is used, but the chains can be run sequentially as well by setting `chains_parallel = FALSE`. To retain reproducibility while running multiple chains in parallel, the same RNG state is passed at the beginning of each chain. This is done by specifying `future.seed = TRUE` in `future.apply::future_lapply` call. Then at the beginning of the i-th chain, before drawing any parameters, i-many Uniform(0, 1) random numbers are generated using `runif(i)` (and then thrown away). This ensures that the RNG states across chains prior to random generation of the parameters are different, and hence, no two chains can become identical, even if they have the same starting and tuning parameters. This, however creates a difference between a `fit_angmix` call with multiple chains which is run sequentially by setting `chains_parallel = FALSE`, and another which is run sequentially because of a sequential `plan()` (or no `plan()`), with `chains_parallel = TRUE`. In the former, different RNG states are passed at the initiation of each chain.

References

Fruhwirth-Schnatter, S. (2011). Label switching under model uncertainty. Mixtures: Estimation and Application, 213-239.

Fruhwirth-Schnatter, S. (2001). Markov chain Monte Carlo estimation of classical and dynamic switching and mixture models. Journal of the American Statistical Association, 96(453), 194-209.

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_angmix("vmsin", tim8,
  ncomp = 3, n.iter = 20,
  n.chains = 1
)
fit.vmsin.20


# Parallelization is implemented via future_lapply from the
# package future.apply. To parallelize, first provide a parallel
# plan(); otherwise the chains will run sequentially.
# Note that not all plan() might work on every OS, as they execute
# functions defined internally in fit_mixmodel. We suggest
# plan(multisession) which works on every OS.

library(future)
library(parallel)
# plan(multisession, gc = TRUE) # parallelize chains

set.seed(1)
MC.fit <- fit_angmix("vmsin", tim8,
  ncomp = 3, n.iter = 5000,
  n.chains = 3
)
```

```
pointest(MC.fit)

MC.fix <- fix_label(MC.fit)

contour(MC.fit)
contour(MC.fix)
lpdtrace(MC.fit)
```

`fit_incremental_angmix` *Stepwise fitting of angular mixture models with incremental component sizes and optimum model selection*

Description

Stepwise fitting of angular mixture models with incremental component sizes and optimum model selection

Usage

```
fit_incremental_angmix(
  model,
  data,
  crit = "LOOIC",
  start_ncomp = 1,
  max_ncomp = 10,
  L = NULL,
  fn = mean,
  fix_label = NULL,
  form = 2,
  start_par = NULL,
  prev_par = TRUE,
  logml_maxiter = 10000,
  return_all = FALSE,
  save_fits = FALSE,
  save_file = NULL,
  save_dir = "",
  silent = FALSE,
  return_llik_contri = (crit %in% c("LOOIC", "WAIC")),
  use_best_chain = TRUE,
  alpha = 0.05,
  bonferroni_alpha = TRUE,
  bonferroni_adj_type = "decreasing",
  ...
)
```

Arguments

model	angular model whose mixtures are to be fitted. Available choices are "vmsin", "vmcos" and "wnorm2" for bivariate data, and "vm" and "wnorm" for univariate data.
data	data matrix (if bivarate, in which case it must have two columns) or vector. If outside, the values are transformed into the scale $[0, 2\pi)$. *Note:* BAMBI cannot handle missing data. Missing values must either be removed or properly imputed.
crit	model selection criteria, one of "LOOIC", "WAIC", "AIC", "BIC", "DIC" or "LOGML". Default is "LOOIC".
start_ncomp	starting component size. A single component model is fitted if start_ncomp is equal to one.
max_ncomp	maximum number of components allowed in the mixture model.
L	HMC tuning parameter (trajectory length) passed to fit_angmix. Can be a numeric vetor (or scalar), in which case the same L is passed to all fit_angmix calls, or can be a list of length max_ncomp-start_ncomp+1, so that L_list[[i]] is passed as the argument L to fit_angmix call with ncomp = max_ncomp+i-1. See fit_angmix for more details on L including its default values. Ignored if method = "rwmh".
fn	function to evaluate on MCMC samples to estimate parameters. Defaults to mean, which computes the estimated posterior means. If fn = max, then MAP estimate is calculated from the MCMC run. Used only if crit = "DIC", and ignored otherwise.
fix_label	logical. Should the label switchings on the current fit (only the corresponding "best chain" if use_best_chain = TRUE) be fixed before computing parameter estimates and model selection criterion? Defaults to TRUE if perm_sampling is true in the fit_angmix call, or if crit = "DIC" and form = 1.
form	form of crit to be used. Available choices are 1 and 2. Used only if crit is "DIC" and ignored otherwise.
start_par	list with elements pmix (ignored if comp == 1), together with kappa1, kappa2, mu1 and mu2, for bivariate models, and kappa and mu for univariate models, all being vectors of length same as ncomp. These provides the starting values for the Markov chain; with j-th component of each vector corresponding to the j-th component of the mixture distribution. If missing, the data is first clustered into ncomp groups either via k-means (after projecting onto a unit sphere), or randomly, depending on rand_start, and then moment estimators for components are used as the starting points. Note that a very wrong starting point can potentially lead the chain to get stuck at a wrong solution for thousands of iterations. As such, we recommend using the default option, which is k-means followed by moment estimation.
prev_par	logical. Should the MAP estimated parameters from the model with ncomp = K be used in the model with ncomp = K+1 as the starting parameters, with the component with largest mixing proportion appearing twice in the bigger model?
logml_maxiter	maximum number of iterations (maxiter) passed to bridge_sampler for calculating LOGML. Ignored if crit is not LOGML.

`return_all` logical. Should all angmcmc objects obtained during step-wise run be returned? *Warning*: depending on the sizes of `n.iter`, `start_ncomp`, `max_ncomp` and `n.chains`, this can be very memory intesive. In such cases, it is recommended that `return_all` be set to `FALSE`, and, if required, the intermediate fitted objects be saved to file by setting `save_fits = TRUE`.

`save_fits` logical. Should the intermediate angmcmc objects obtained during step-wise run be saved to file using save? Defaults to TRUE. See `save_file` and `save_dir`.

`save_file, save_dir`
`save_file` is a list of size `max_ncomp-start_ncomp+1`, with k-th entry providing the `file` argument used to save the intermediate angmcmc object with `ncomp = k` (titled `"fit_angmcmc"`). If not provided, then k-th element of `save_file[[k]]` is taken to be `paste(save_dir, "comp_k", sep="/")`. Both are ignored if `save_fits = FALSE`.

`silent` logical. Should the current status (such as what is the current component labels, which job is being done etc.) be printed? Defaults to `TRUE`.

`return_llik_contri`
passed to fit_angmix. By default, set to `TRUE` if `crit` is either `"LOOIC"` or `"WAIC"`, and to `FALSE` otherwise.

`use_best_chain` logical. Should only the "best" chain obtained during each intermediate fit be used during computation of model selection criterion? Here "best" means the chain with largest (mean over iterations) log-posterior density. This can be helpful if one of the chains gets stuck at local optima. Defaults to TRUE.

`alpha` significance level used in the test H_{0K}: expected log predictive density (elpd) for the fitted model with K components >= elpd for the fitted model with K + 1 components if `crit` is `"LOOIC"` or `"WAIC"`. Must be a scalar between 0 and 1. Defaults to 0.05. See Details. Ignored for any other `crit`.

`bonferroni_alpha`
logical. Should a Bonferroni correction be made on the test size `alpha` to adjust for multiplicity due to (`max_ncomp - start_ncomp`) possible hypothesis tests? Defaults to TRUE. Relevant only if `crit` is in `c("LOOIC", "WAIC")`, and ignored otherwise. See Details.

`bonferroni_adj_type`
character string. Denoting type of Bonferroni adjustment to make. Possible choices are `"decreasing"` (default) and `"equal"`. Ignored if either `bonferroni_alpha` is `FALSE`, or `crit` is outside `c("LOOIC", "WAIC")`. See Details.

`...` additional arguments passed to fit_angmix.

Details

The goal is to fit an angular mixture model with an optimally chosen component size K. To obtain an optimum K, mixture models with incremental component sizes between `start_ncomp` and `max_ncomp` are fitted incrementally using fit_angmix, starting from K = 1. If the model selection criterion `crit` is `"LOOIC"` or `"WAIC"`, then a test of hypothesis H_{0K}: expected log predictive density (elpd) for the fitted model with K components >= elpd for the fitted model with K + 1 components, is performed at every K >= 1. The test-statistic used for the test is an approximate z-score based on the normalized estimated elpd difference between the two models obtained from compare, which provides estimated elpd difference along with its standard error estimate. Because the computed

standard error of elpd difference can be overly optimistic when the elpd difference is small (in particular < 4), a conservative worst-case estimate (equal to twice of the computed standard error) is used in such cases. To account for multiplicity among the M = (max_ncomp - start_ncomp) possible sequential tests performed, by default a Bonferroni adjustment to the test level alpha is made. Set bonferroni_alpha = FALSE} to remove the adjustment. To encourage parsimony in the final model, by defaul = "decreasing") a decreasing sequence of adjusted alphas of the form alpha * (0.5)^(1:M) / sum((0.5)^(1:M))is used. Setbonferroni_adj_type = "equal"to use equal sequence of adjusted alphas (i.e.,alpha instead.

The incremental fitting stops if H_{0K} cannot be rejected (at level alpha) for some K >= 1; this K is then regarded as the optimum number of components. If crit is not "LOOIC" or "WAIC" then mixture model with the first minimum value of the model selection criterion crit is taken as the best model.

Note that in each intermediate fitted model, the total number of components (instead of the number of "non-empty components") in the model is used to estimate of the true component size, and then the fitted model is penalized for model complexity (via the model selection criterion used). This approach of selecting an optimal K follows the perspective "let two component specific parameters be identical" for overfitting mixtures, and as such the Dirichlet prior hyper-parameters pmix.alpha (passed to fit_angmix) should be large. See Fruhwirth-Schnatter (2011) for more deltails.

Note that the stability of bridge_sampler used in marginal likelihood estimation heavily depends on stationarity of the chains. As such, while using this criterion, we recommending running the chain long enough, and setting fix_label = TRUE for optimal performance.

Value

Returns a named list (with class = stepfit) with the following seven elements:

fit.all (if return_all = TRUE) - a list all angmcmc objects created at each component size;

fit.best - angmcmc object corresponding to the optimum component size;

ncomp.best - optimum component size (integer);

crit - which model comparison criterion used (one of "LOOIC", "WAIC", "AIC", "BIC", "DIC" or "LOGML");

crit.all - all crit values calculated (for all component sizes);

crit.best - crit value for the optimum component size; and

maxllik.all - maximum (obtained from MCMC iterations) log likelihood for all fitted models

maxllik.best - maximum log likelihodd for the optimal model; and

check_min - logical; is the optimum component size less than max_ncomp?

References

Fruhwirth-Schnatter, S.: Label switching under model uncertainty. In: Mengerson, K., Robert, C., Titterington, D. (eds.) Mixtures: Estimation and Application, pp. 213-239. Wiley, New York (2011).

Examples

```
# illustration only - more iterations needed for convergence
set.seed(1)
fit.vmsin.step.15 <- fit_incremental_angmix("vmsin", tim8, "BIC", start_ncomp = 1,
                                            max_ncomp = 3, n.iter = 15,
                                            n.chains = 1, save_fits=FALSE)
(fit.vmsin.best.15 <- bestmodel(fit.vmsin.step.15))
lattice::densityplot(fit.vmsin.best.15)
```

`fit_vmcosmix` *Fitting bivariate von Mises cosine model mixtures using MCMC*

Description

Fitting bivariate von Mises cosine model mixtures using MCMC

Usage

```
fit_vmcosmix(...)
```

Arguments

`...`	arguments (other than `model`) passed to fit_angmix

Details

Wrapper for fit_angmix with `model = "vmcos"`.

Examples

```
# illustration only - more iterations needed for convergence
fit.vmcos.10 <- fit_vmcosmix(tim8, ncomp = 3, n.iter =  10,
                             n.chains = 1)
fit.vmcos.10
```

fit_vmmix *Fitting univariate von Mises mixtures using MCMC*

Description

Fitting univariate von Mises mixtures using MCMC

Usage

```
fit_vmmix(...)
```

Arguments

...	arguments (other than model) passed to fit_angmix

Details

Wrapper for fit_angmix with model = "vm".

Examples

```
# illustration only - more iterations needed for convergence
fit.vm.20 <- fit_vmmix(wind$angle, ncomp = 3, n.iter =  20,
                       n.chains = 1)
fit.vm.20
```

fit_vmsinmix *Fitting bivariate von Mises sine model mixtures using MCMC*

Description

Fitting bivariate von Mises sine model mixtures using MCMC

Usage

```
fit_vmsinmix(...)
```

Arguments

...	arguments (other than model) passed to fit_angmix

Details

Wrapper for fit_angmix with model = "vmsin"

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
fit.vmsin.20
```

`fit_wnorm2mix` *Fitting bivariate wrapped normal model mixtures using MCMC*

Description

Fitting bivariate wrapped normal model mixtures using MCMC

Usage

```
fit_wnorm2mix(...)
```

Arguments

`...`	arguments (other than `model`) passed to fit_angmix

Details

Wrapper for fit_angmix with `model = "wnorm2"`.

Examples

```
# illustration only - more iterations needed for convergence
fit.wnorm2.10 <- fit_wnorm2mix(tim8, ncomp = 3, n.iter =  10,
                             n.chains = 1)
fit.wnorm2.10
```

`fit_wnormmix` *Fitting univariate wrapped normal mixtures using MCMC*

Description

Fitting univariate wrapped normal mixtures using MCMC

Usage

```
fit_wnormmix(...)
```

Arguments

...	arguments (other than model) passed to fit_angmix

Details

Wrapper for fit_angmix with model = "wnorm".

Examples

```
# illustration only - more iterations needed for convergence
fit.wnorm.20 <- fit_wnormmix(wind$angle, ncomp = 3, n.iter =  20,
                             n.chains = 1)
fit.wnorm.20
```

---

fix_label *Fix label switching in angmcmc objects*

Description

Fix label switching in angmcmc objects

Usage

```
fix_label(object, ...)
```

Arguments

object	angular MCMC object.
...	arguments other than z, K, complete, mcmc, p and data passed to label.switching. See details.

Details

fix_label is a wrapper for label.switching from package label.switching for angmcmc objects. The arguments z, K, complete, mcmc, p and data are appropriately filled in from object. The label.switching argument method can be a scalar or vector; for this wrapper it defaults to "STEPHENS" if the angmcmc was created with permutation sampling (by setting perm_sampling = TRUE in fit_angmix), and to "DATA-BASED" otherwise.

Value

Returns a single angmcmc object or a list of angmcmc objects (according as whether the argument method is a scalar or vector) with label switchings corrected (after burn-in and thin) according to the resulting permutation from label.switching.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# now apply fix_label
fit.vmsin.20.fix <- fix_label(fit.vmsin.20)
```

---

is.angmcmc *Angular MCMC (*angmcmc*) Object*

Description

Checking for and creating an angmcmc object

Usage

```
is.angmcmc(object)

angmcmc(...)
```

Arguments

object	any R object
...	arguments required to make an angmcmc object. See details

Details

angmcmc objects are classified lists that are created when any of the five mixture model fitting functions, viz., fit_vmmix, fit_wnormmix, fit_vmsinmix, fit_vmcosmix and fit_wnorm2mix is used. An angmcmc object contains a number of elements, including the dataset, the model being fitted on the dataset and dimension of the model (univariate or bivariate), the tuning parameters used, MCMC samples for the mixture model parameters, the (hidden) component or cluster indicators for data points in each iteration and the (iteration-wise) log likelihood and log posterior density values (both calculated upto some normalizing constants). When printed, an angmcmc object returns a brief summary of the function arguments used to produce the object and the average acceptance rate of the proposals (in HMC and RWMH) used over iterations. An angmcmc object can be used as an argument for the diagnostic and post-processing functions available in BAMBI for making further inferences.

Value

logical. Is the input an angmcmc object?

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
is.angmcmc(fit.vmsin.20)
```

`latent_allocation` *Finding latent allocation (component indicators) from an angmcmc object*

Description

Finding latent allocation (component indicators) from an angmcmc object

Usage

```
latent_allocation(object, ...)
```

Arguments

`object`	angular MCMC object.
`...`	passed to pointest to estimate parameters.

Details

In order to find the latent component indicators, estimates of mixing proportions and model parameters are first computed via pointest. Then, a data point is assigned label j, if the j-th component gives highest density for that point.

Value

Returns a vector of length n, where n is the length (if univariate) or number of rows (if bivariate) of the data used in original fit. i-th entry of the output vector provides component label for the i-th data point.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# now find latent allocation
latent_allocation(fit.vmsin.20)
```

`logLik.angmcmc` *Extract Log-Likelihood from angmcmc objects*

Description

Extract Log-Likelihood from angmcmc objects

Usage

```
## S3 method for class 'angmcmc'
logLik(object, method = 1, fn, ...)
```

Arguments

`object`	angular MCMC object.
`method`	interger specifying method of estimating the log likelihood. Must be 1 or 2. Defaults to 1. See details.
`fn`	function to evaluate on the iteration-wise log-likelihood values obtained during MCMC run if `method = 1`; or, if `method = 2`, function to evaluate on the MCMC samples for parameter estimation (passed to pointest). Defaults to `max` if `method = 1` and `mean` if `method = 2`.
`...`	additional arguments to be passed to the function.

Details

There are two ways to estimate the log likelihood from the model. If `method = 1`, then log likelihood is estimated by applying `fn` (defaults to max, if method = 1) direclty on the log likelihood values from observed during the MCMC run. On the other hand, if `method == 2`, then parameter estimates are first computed using `pointest` with `fn` (defaults to "MODE", if `method == 2`) applied on the MCMC samples, and then then log likelihood is evaluated at the parameter estimates.

The degrees of the likelihood function is the total number of free parameters estimated in the mixture models, which is equal to $6K - 1$ for bivariate models (vmsin, vmcos and wnorm2), or $3K - 1$ for univariate models (vm and wnorm), where K denotes the number of components in the mixture model.

Value

Returns an object of class logLik. This is a number (the estimated log likelihood) with attributes "df" (degrees of freedom) and "nobs" (number of observations).

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
logLik(fit.vmsin.20)
```

loo.angmcmc *Leave-one-out cross-validation (LOO) for angmcmc objects*

Description

Leave-one-out cross-validation (LOO) for angmcmc objects

Usage

```
## S3 method for class 'angmcmc'
loo(x, ...)
```

Arguments

x	angmcmc object.
...	additional model specific arguments to be passed to waic from loo. For example, int.displ specifies integer displacement in wnorm and wnorm2 models. See fit_wnormmix and fit_wnorm2mix for more details.

Details

Note that loo.angmcmc calls loo for computation. If the likelihood contribution of each data point for each MCMC iteration is available in object (can be returned by setting return_llik_contri = TRUE) during fit_angmix call), loo.array is used; otherwise loo.function is called. Computation is much faster if the likelihood contributions are available - however, they are very memory intensive, and by default not returned in fit_angmix.

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1, return_llik_contri = TRUE)
library(loo)
loo(fit.vmsin.20)
```

lpdtrace *Trace and autocorrelation plots of log posterior density or log likelihood from an angmcmc object*

Description

Trace and autocorrelation plots of log posterior density or log likelihood from an angmcmc object

Usage

```
lpdtrace(
  object,
  chain.no,
  use.llik = FALSE,
  plot.autocor = FALSE,
  lag.max = NULL,
  ...
)
```

Arguments

object	angular MCMC object.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
use.llik	logical. Should log likelihood be plotted instead of log posterior? Set to FALSE by default.
plot.autocor	logical. Should the autocorrelations be plotted as well?
lag.max	maximum lag for autocorrelation. Passed to acf. Ignored if plot.autocor = FALSE.
...	unused

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# log posterior density trace
lpdtrace(fit.vmsin.20)
# log likelihood trace
lpdtrace(fit.vmsin.20, use.llik = TRUE)
```

---

paramtrace *Trace plot for parameters from an angmcmc object*

Description

Trace plot for parameters from an angmcmc object

Usage

```
paramtrace(object, par.name, comp.label, chain.no, ...)
```

Arguments

object	angular MCMC object.
par.name	vector of names of parameters for which point estimates are to be computed. If NULL, results for all parameters are provided.
comp.label	vector of component labels (positive integers, e.g., 1, 2, ...) for which point estimates are to be computed. If NULL, results for all components are provided.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
...	unused
par	parameter for which trace plot is to be created.

Value

Returns a single plot if a single par and a single comp.label is supplied. Otherwise, a series of plots is produced.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# trace plot for kappa1 in component 1
paramtrace(fit.vmsin.20, "kappa1", 1)
# for kappa1 in all components
paramtrace(fit.vmsin.20, "kappa1")
# for all parameters in component 1
paramtrace(fit.vmsin.20, comp.label = 1)
```

---

plot.angmcmc *Summary plots for angmcmc objects*

Description

Summary plots for angmcmc objects

Usage

```
## S3 method for class 'angmcmc'
plot(
  x,
  par.name,
  comp.label,
  chain.no,
  do.paramtrace = TRUE,
```

```
  do.lpdtrace = TRUE,
  use.llik = FALSE,
  ...
)
```

Arguments

x	angmcmc object
par.name	vector of names of parameters for which point estimates are to be computed. If NULL, results for all parameters are provided.
comp.label	vector of component labels (positive integers, e.g., 1, 2, ...) for which point estimates are to be computed. If NULL, results for all components are provided.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
do.paramtrace	logical. Should the trace(s) for the parameter(s) be plotted?
do.lpdtrace	logical. Should the log posterior trace be plotted?
use.llik	logical. Should the log likelihood be plotted instead? Ignored if do.lpdtrace == FALSE.
...	unused

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
plot(fit.vmsin.20)
```

---

pointest *Point estimates for parameters from an angmcmc object*

Description

Point estimates for parameters from an angmcmc object

Usage

```
pointest(object, fn = mean, par.name, comp.label, chain.no, ...)
```

Arguments

object	angular MCMC object.
fn	function, or a single character string specifying its name, to evaluate on MCMC samples to estimate parameters. Defaults to mean, which computes the estimated posterior mean. Note that if fn = "MODE" (warning: not "mode") or fn = "MAP", then the maximum aposteriori estimate (MAP) is calculated.

par.name	vector of names of parameters for which point estimates are to be computed. If NULL, results for all parameters are provided.
comp.label	vector of component labels (positive integers, e.g., 1, 2, ...) for which point estimates are to be computed. If NULL, results for all components are provided.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
...	additional arguments to be passed to the function.

Value

Returns a matrix of point estimates, or vector of point estimates if length(par.name)==1 or length(comp.label)==1.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# estimate parameters by sample mean
(est_mean <- pointest(fit.vmsin.20))
# estimate parameters by sample median
(est_median <- pointest(fit.vmsin.20, fn = median))
# estimate parameters by MAP
(est_median <- pointest(fit.vmsin.20, fn = "MODE"))
```

---

quantile.angmcmc *Quantile estimates for parameters from an angmcmc object*

Description

Quantile estimates for parameters from an angmcmc object

Usage

```
## S3 method for class 'angmcmc'
quantile(x, par.name, comp.label, chain.no, probs = seq(0, 1, 0.25), ...)
```

Arguments

x	angmcmc object
par.name	vector of names of parameters for which point estimates are to be computed. If NULL, results for all parameters are provided.
comp.label	vector of component labels (positive integers, e.g., 1, 2, ...) for which point estimates are to be computed. If NULL, results for all components are provided.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.

probs	numeric vector of probabilities with values in $[0, 1]$. (Values up to '2e-14' outside that range are accepted and moved to the nearby endpoint.)
...	further arguments to pass to quantile. In particular, probs = seq(0, 1, 0.25) is the default vector of quantiles computed for each parameter.

Value

Returns a three dimensional array of quantiles, or a matrix (vector) of quantiles if one (or two) among par.name, comp.label, probs has length 1.

Examples

```
# first fit a vmsin mixture model
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
# 0.025th quantiles
(quant_025 <- quantile(fit.vmsin.20, prob = 0.025))
# 0.975th quantiles
(quant_975 <- quantile(fit.vmsin.20, prob = 0.975))
# default quantiles
(quant_def <- quantile(fit.vmsin.20))
```

---

rvm *The univariate von Mises distribution*

Description

The univariate von Mises distribution

Usage

```
rvm(n, kappa = 1, mu = 0)

dvm(x, kappa = 1, mu = 0, log = FALSE)
```

Arguments

n	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
kappa	vector of concentration (inverse-variance) parameters; kappa > 0.
mu	vector of means.
x	vector of angles (in radians) where the densities are to be evaluated.
log	logical. Should the log density be returned instead?

Details

If mu and kappa are not specified they assume the default values of 0 and 1 respectively.

The univariate von Mises distribution has density

$$f(x) = 1/(2\pi I_0(\kappa)) \exp(\kappa \cos(x - mu))$$

where $I_0(\kappa)$ denotes the modified Bessel function of the first kind with order 0 evaluated at the point κ.

Value

dvm gives the density and rvm generates random deviates.

Examples

```
kappa <- 1:3
mu <- 0:2
x <- 1:10
n <- 10


# when x and both parameters are scalars, dvm returns a single density
dvm(x[1], kappa[1], mu[1])

# when x is a vector but both the parameters are scalars, dmv returns a vector of
# densities calculated at each entry of x with the same parameters
dvm(x, kappa[1], mu[1])

# if x is scalar and at least one of the two paraemters is a vector, both parameters are
# recycled to the same length, and dvm returns a vector of with ith element being the
# density evaluated at x with parameter values kappa[i] and mu[i]
dvm(x[1], kappa, mu)

# if x and at least one of the two paraemters is a vector, x and the two parameters are
# recycled to the same length, and dvm returns a vector of with ith element being the
# density at ith element of the (recycled) x with parameter values kappa[i] and mu[i]
dvm(x, kappa, mu)

# when parameters are all scalars, number of observations generated by rvm is n
rvm(n, kappa[1], mu[1])

# when at least one of the two parameters is a vector, both are recycled to the same length,
# n is ignored, and the number of observations generated by rvm is the same as the length of
# the recycled vectors
rvm(n, kappa, mu)
```

rvmcos *The bivariate von Mises cosine model*

Description

The bivariate von Mises cosine model

Usage

```
rvmcos(
  n,
  kappa1 = 1,
  kappa2 = 1,
  kappa3 = 0,
  mu1 = 0,
  mu2 = 0,
  method = "naive"
)

dvmcos(
  x,
  kappa1 = 1,
  kappa2 = 1,
  kappa3 = 0,
  mu1 = 0,
  mu2 = 0,
  log = FALSE,
  ...
)
```

Arguments

n	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
kappa1, kappa2, kappa3	vectors of concentration parameters; kappa1, kappa2 > 0.
mu1, mu2	vectors of mean parameters.
method	Rejection sampling method to be used. Available choices are "naive" (default) or "vmprop". See details.
x	bivariate vector or a two-column matrix with each row being a bivariate vector of angles (in radians) where the densities are to be evaluated.
log	logical. Should the log density be returned instead?
...	additional arguments to be passed to dvmcos. See details.

Details

The bivariate von Mises cosine model density at the point $x = (x_1, x_2)$ is given by

$$f(x) = C_c(\kappa_1, \kappa_2, \kappa_3) \exp(\kappa_1 \cos(T_1) + \kappa_2 \cos(T_2) + \kappa_3 \cos(T_1 - T_2))$$

where

$$T_1 = x_1 - \mu_1; T_2 = x_2 - \mu_2$$

and $C_c(\kappa_1, \kappa_2, \kappa_3)$ denotes the normalizing constant for the cosine model.

Because C_c involves an infinite alternating series with product of Bessel functions, if `kappa3 < -5` or `max(kappa1, kappa2, abs(kappa3)) > 50`, C_c is evaluated numerically via (quasi) Monte carlo method for numerical stability. These (quasi) random numbers can be provided through the argument `qrnd`, which must be a two column matrix, with each element being a (quasi) random number between 0 and 1. Alternatively, if `n_qrnd` is provided (and `qrnd` is missing), a two dimensional sobol sequence of size `n_qrnd` is generated via the function sobol from the R package `qrng`. If none of `qrnd` or `n_qrnd` is available, a two dimensional sobol sequence of size 1e4 is used. By default Monte Carlo approximation is used only if `kappa3 < -5` or `max(kappa1, kappa2, abs(kappa3)) > 50`. However, a forced Monte Carlo approximation can be made (irrespective of the choice of `kappa1`, `kappa2` and `kappa3`) by setting `force_approx_const = TRUE`. See examples.

Value

`dvmcos` gives the density and `rvmcos` generates random deviates.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
x <- diag(2, 2)
n <- 10

# when x is a bivariate vector and parameters are all scalars,
# dvmcos returns single density
dvmcos(x[1, ], kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when x is a two column matrix and parameters are all scalars,
# dmvsin returns a vector of densities calculated at the rows of
# x with the same parameters
dvmcos(x, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# if x is a bivariate vector and at least one of the parameters is
# a vector, all parameters are recycled to the same length, and
# dvmcos returns a vector with ith element being the density
# evaluated at x with parameter values kappa1[i], kappa2[i],
# kappa3[i], mu1[i] and mu2[i]
dvmcos(x[1, ], kappa1, kappa2, kappa3, mu1, mu2)

# if x is a two column matrix and at least one of the parameters is
```

```
# a vector, rows of x and the parameters are recycled to the same
# length, and dvmcos returns a vector with ith element being the
# density evaluated at ith row of x with parameter values kappa1[i],
# kappa2[i], # kappa3[i], mu1[i] and mu2[i]
dvmcos(x, kappa1, kappa2, kappa3, mu1, mu2)

# when parameters are all scalars, number of observations generated
# by rvmcos is n
rvmcos(n, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when at least one of the parameters is a vector, all parameters are
# recycled to the same length, n is ignored, and the number of
# observations generated by rvmcos is the same as the length of the
# recycled vectors
rvmcos(n, kappa1, kappa2, kappa3, mu1, mu2)



## Visualizing (quasi) Monte Carlo based approximations of
## the normalizing constant through density evaluations.

# "good" setup, where the analytic formula for C_c can be
# calculated without numerical issues
# kappa1 = 1, kappa2 = 1, kappa3 = -2, mu1 = pi, mu2 = pi

n_qrnd <-  (1:500)*20
# analytic
good.a <- dvmcos(c(3,3), 1, 1, -2, pi, pi, log=TRUE)
# using quasi Monte Carlo
good.q <- sapply(n_qrnd,
                 function(j)
                   dvmcos(c(3,3), 1, 1, -2, pi, pi,
                          log=TRUE, n_qrnd = j,
                          force_approx_const = TRUE))
# using ordinary Monte Carlo
set.seed(1)
good.r <- sapply(n_qrnd,
                 function(j)
                   dvmcos(c(3,3), 1, 1, -2, pi, pi,
                          log=TRUE,
                          qrnd = matrix(runif(2*j), ncol = 2),
                          force_approx_const = TRUE))


plot(n_qrnd, good.q, ylim = range(good.a, good.q, good.r),
     col = "orange", type = "l",
     ylab = "",
     main = "dvmcos(c(3,3), 1, 1, -2, pi, pi, log = TRUE)")
points(n_qrnd, good.r, col = "skyblue", type = "l")
abline(h = good.a, lty = 2, col = "grey")
legend("topright",
       legend = c("Sobol", "Random", "Analytic"),
       col = c("orange", "skyblue", "grey"),
```

```
       lty = c(1, 1, 2))


# "bad" setup, where the calculating C_c
# numerically using the analytic formula is problematic
# kappa1 = 100, kappa2 = 100, kappa3 = -200, mu1 = pi, mu2 = pi

n_qrnd <-  (1:500)*20

# using quasi Monte Carlo
bad.q <- sapply(n_qrnd,
                function(j)
                  dvmcos(c(3,3), 100, 100, -200, pi, pi,
                         log=TRUE, n_qrnd = j,
                         force_approx_const = TRUE))
# using ordinary Monte Carlo
set.seed(1)
bad.r <- sapply(n_qrnd,
                function(j)
                  dvmcos(c(3,3), 100, 100, -200, pi, pi,
                         log=TRUE,
                         qrnd = matrix(runif(2*j), ncol = 2),
                         force_approx_const = TRUE))


plot(n_qrnd, bad.q, ylim = range(bad.q, bad.r),
     col = "orange", type = "l",
     ylab = "",
     main = "dvmcos(c(3,3), 100, 100, -200, pi, pi, log = TRUE)")
points(n_qrnd, bad.r, col = "skyblue", type = "l")
legend("topright",
       legend = c("Sobol", "Random"),
       col = c("orange", "skyblue"), lty = 1)
```

`rvmcosmix` *The bivariate von Mises cosine model mixtures*

Description

The bivariate von Mises cosine model mixtures

Usage

```
rvmcosmix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix, method = "naive", ...)

dvmcosmix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix, log = FALSE, ...)
```

Arguments

n	number of observations.
kappa1, kappa2, kappa3	vectors of concentration parameters; kappa1, kappa2 > 0 for each component.
mu1, mu2	vectors of mean parameters.
pmix	vector of mixture proportions.
method	Rejection sampling method to be used. Available choices are "naive" (default) or "vmprop". See details.
...	additional arguments to be passed to dvmcos. See details.
x	matrix of angles (in radians) where the density is to be evaluated, with each row being a single bivariate vector of angles.
log	logical. Should the log density be returned instead?

Details

All the argument vectors pmix, kappa1, kappa2, kappa3, mu1 and mu2 must be of the same length (= component size of the mixture model), with j-th element corresponding to the j-th component of the mixture distribution.

The bivariate von Mises cosine model mixture distribution with component size K = length(pmix) has density

$$g(x) = \sum p[j] * f(x; \kappa_1[j], \kappa_2[j], \kappa_3[j], \mu_1[j], \mu_2[j])$$

where the sum extends over j; $p[j]$; $\kappa_1[j], \kappa_2[j], \kappa_3[j]$; and $\mu_1[j], \mu_2[j]$ respectively denote the mixing proportion, the three concentration parameters and the two mean parameter for the j-th cluster, $j = 1, ..., K$, and $f(.; \kappa_1, \kappa_2, \kappa_3, \mu_1, \mu_2)$ denotes the density function of the von Mises cosine model with concentration parameters $\kappa_1, \kappa_2, \kappa_3$ and mean parameters μ_1, μ_2.

Value

dvmcosmix computes the density and rvmcosmix generates random deviates from the mixture density.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
pmix <- c(0.3, 0.4, 0.3)
x <- diag(2, 2)
n <- 10

# mixture densities calculated at the rows of x
dvmcosmix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix)

# number of observations generated from the mixture distribution is n
rvmcosmix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix)
```

`rvmmix` *The univariate von Mises mixtures*

Description

The univariate von Mises mixtures

Usage

```
rvmmix(n, kappa, mu, pmix)

dvmmix(x, kappa, mu, pmix, log = FALSE)
```

Arguments

`n`	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
`kappa`	vector of component concentration (inverse-variance) parameters, `kappa > 0`.
`mu`	vector of component means.
`pmix`	vector of mixing proportions.
`x`	vector of angles (in radians) where the densities are to be evaluated.
`log`	logical. Should the log density be returned instead?

Details

`pmix`, `mu` and `kappa` must be of the same length, with j-th element corresponding to the j-th component of the mixture distribution.

The univariate von Mises mixture distribution with component size `K = length(pmix)` has density

$$g(x) = p[1] * f(x; \kappa[1], \mu[1]) + ... + p[K] * f(x; \kappa[K], \mu[K])$$

where $p[j], \kappa[j], \mu[j]$ respectively denote the mixing proportion, concentration parameter and the mean parameter for the j-th component and $f(.; \kappa, \mu)$ denotes the density function of the (univariate) von Mises distribution with mean parameter μ and concentration parameter κ.

Value

`dvmmix` computes the density and `rvmmix` generates random deviates from the mixture density.

Examples

```
kappa <- 1:3
mu <- 0:2
pmix <- c(0.3, 0.3, 0.4)
x <- 1:10
n <- 10

# mixture densities calculated at each point in x
dvmmix(x, kappa, mu, pmix)

# number of observations generated from the mixture distribution is n
rvmmix(n, kappa, mu, pmix)
```

`rvmsin` *The bivariate von Mises sine model*

Description

The bivariate von Mises sine model

Usage

```
rvmsin(
  n,
  kappa1 = 1,
  kappa2 = 1,
  kappa3 = 0,
  mu1 = 0,
  mu2 = 0,
  method = "naive"
)

dvmsin(x, kappa1 = 1, kappa2 = 1, kappa3 = 0, mu1 = 0, mu2 = 0, log = FALSE)
```

Arguments

`n`	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
`kappa1, kappa2, kappa3`	vectors of concentration parameters; `kappa1, kappa2 > 0`.
`mu1, mu2`	vectors of mean parameters.
`method`	Rejection sampling method to be used. Available choices are `"naive"` (default) or `"vmprop"`. See details.
`x`	bivariate vector or a two-column matrix with each row being a bivariate vector of angles (in radians) where the densities are to be evaluated.
`log`	logical. Should the log density be returned instead?

Details

The bivariate von Mises sine model density at the point $x = (x_1, x_2)$ is given by

$$f(x) = C_s(\kappa_1, \kappa_2, \kappa_3) \exp(\kappa_1 \cos(T_1) + \kappa_2 \cos(T_2) + \kappa_3 \sin(T_1) \sin(T_2))$$

where

$$T_1 = x_1 - \mu_1; T_2 = x_2 - \mu_2$$

and $C_s(\kappa_1, \kappa_2, \kappa_3)$ denotes the normalizing constant for the sine model.

Two different rejection sampling methods are implemented for random generation. If `method = "vmprop"`, then first the y-marginal is drawn from the associated marginal density, and then x is generated from the conditional distributio of x given y. The marginal generation of y is implemented in a rejection sampling scheme with proposal being either von Mises (if the target marginal density is unimodal), or a mixture of von Mises (if bimodal), with optimally chosen concentration. This the method suggested in Mardia et al. (2007). On the other hand, when `method = "naive"` (default) a (naive) bivariate rejection sampling scheme with (bivariate) uniform propsoal is used.

Note that although `method = "vmprop"` may provide better efficiency when the density is highly concentrated, it does have an (often substantial) overhead due to the optimziation step required to find a reasonable proposal concentration parameter. This can compensate the efficiency gains of this method, especially when `n` is not large.

Value

`dvmsin` gives the density and `rvmsin` generates random deviates.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
x <- diag(2, 2)
n <- 10

# when x is a bivariate vector and parameters are all scalars,
# dvmsin returns single density
dvmsin(x[1, ], kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when x is a two column matrix and parameters are all scalars,
# dmvsin returns a vector of densities calculated at the rows of
# x with the same parameters
dvmsin(x, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# if x is a bivariate vector and at least one of the parameters is
# a vector, all parameters are recycled to the same length, and
# dvmsin returns a vector of with ith element being the density
# evaluated at x with parameter values kappa1[i], kappa2[i],
# kappa3[i], mu1[i] and mu2[i]
dvmsin(x[1, ], kappa1, kappa2, kappa3, mu1, mu2)
```

```
# if x is a two column matrix and at least one of the parameters is
# a vector, rows of x and the parameters are recycled to the same
# length, and dvmsin returns a vector of with ith element being the
# density evaluated at ith row of x with parameter values kappa1[i],
# kappa2[i], # kappa3[i], mu1[i] and mu2[i]
dvmsin(x[1, ], kappa1, kappa2, kappa3, mu1, mu2)

# when parameters are all scalars, number of observations generated
# by rvmsin is n
rvmsin(n, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when at least one of the parameters is a vector, all parameters are
# recycled to the same length, n is ignored, and the number of
# observations generated by rvmsin is the same as the length of the
# recycled vectors
rvmsin(n, kappa1, kappa2, kappa3, mu1, mu2)
```

`rvmsinmix` *The bivariate von Mises sine model mixtures*

Description

The bivariate von Mises sine model mixtures

Usage

```
rvmsinmix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix, method = "naive")

dvmsinmix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix, log = FALSE)
```

Arguments

`n`	number of observations.
`kappa1, kappa2, kappa3`	vectors of concentration parameters; `kappa1, kappa2 > 0` for each component.
`mu1, mu2`	vectors of mean parameters.
`pmix`	vector of mixture proportions.
`method`	Rejection sampling method to be used. Available choices are `"naive"` (default) or `"vmprop"`. See details.
`x`	matrix of angles (in radians) where the density is to be evaluated, with each row being a single bivariate vector of angles.
`log`	logical. Should the log density be returned instead?

Details

All the argument vectors `pmix`, `kappa1`, `kappa2`, `kappa3`, `mu1` and `mu2` must be of the same length (= component size of the mixture model), with j-th element corresponding to the j-th component of the mixture distribution.

The bivariate von Mises sine model mixture distribution with component size `K = length(p.mix)` has density

$$g(x) = \sum p[j] * f(x; \kappa_1[j], \kappa_2[j], \kappa_3[j], \mu_1[j], \mu_2[j])$$

where the sum extends over j; $p[j]$; $\kappa_1[j], \kappa_2[j], \kappa_3[j]$; and $\mu_1[j], \mu_2[j]$ respectively denote the mixing proportion, the three concentration parameters and the two mean parameter for the j-th component, $j = 1, ..., K$, and $f(.; \kappa_1, \kappa_2, \kappa_3, \mu_1, \mu_2)$ denotes the density function of the von Mises sine model with concentration parameters $\kappa_1, \kappa_2, \kappa_3$ and mean parameters μ_1, μ_2.

Value

`dvmsinmix` computes the density (vector if x is a two column matrix with more than one row) and `rvmsinmix` generates random deviates from the mixture density.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
pmix <- c(0.3, 0.4, 0.3)
x <- diag(2, 2)
n <- 10

# mixture densities calculated at the rows of x
dvmsinmix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix)

# number of observations generated from the mixture distribution is n
rvmsinmix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix)
```

---

rwnorm *The univariate Wrapped Normal distribution*

Description

The univariate Wrapped Normal distribution

Usage

```
rwnorm(n = 1, kappa = 1, mu = 0)

dwnorm(x, kappa = 1, mu = 0, int.displ, log = FALSE)
```

Arguments

n	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
kappa	vector of concentration (inverse-variance) parameters; kappa > 0.
mu	vector of means.
x	vector of angles (in radians) where the densities are to be evaluated.
int.displ	integer displacement. If int.displ = M, then the infinite sum in the density is approximated by a sum over 2*M + 1 elements. (See Details.) The allowed values are 1, 2, 3, 4 and 5. Default is 3.
log	logical. Should the log density be returned instead?

Details

If mu and kappa are not specified they assume the default values of 0 and 1 respectively.

The univariate wrapped normal distribution has density

$$f(x) = \sqrt{(}\kappa/(2\pi)) \sum \exp(-\kappa/2(x - \mu(2\pi\omega))^2)$$

where the sum extends over all integers ω, and is approximated by a sum over ω in $\{-M, -M + 1, ..., M - 1, M\}$ if int.displ = M.

Value

dwnorm gives the density and rwnorm generates random deviates.

Examples

```
kappa <- 1:3
mu <- 0:2
x <- 1:10
n <- 10


# when x and both parameters are scalars, dwnorm returns a single density
dwnorm(x[1], kappa[1], mu[1])

# when x is a vector but both the parameters are scalars, dmv returns a vector of
# densities calculated at each entry of x with the same parameters
dwnorm(x, kappa[1], mu[1])

# if x is scalar and at least one of the two paraemters is a vector, both parameters are
# recycled to the same length, and dwnorm returns a vector of with ith element being the
# density evaluated at x with parameter values kappa[i] and mu[i]
dwnorm(x[1], kappa, mu)

# if x and at least one of the two paraemters is a vector, x and the two parameters are
# recycled to the same length, and dwnorm returns a vector of with ith element being the
# density at ith element of the (recycled) x with parameter values kappa[i] and mu[i]
```

```
dwnorm(x, kappa, mu)

# when parameters are all scalars, number of observations generated by rwnorm is n
rwnorm(n, kappa[1], mu[1])

# when at least one of the two parameters is a vector, both are recycled to the same length,
# n is ignored, and the number of observations generated by rwnorm is the same as the length
# of the recycled vectors
rwnorm(n, kappa, mu)
```

---

rwnorm2 *The bivariate Wrapped Normal distribution*

Description

The bivariate Wrapped Normal distribution

Usage

```
rwnorm2(n, kappa1 = 1, kappa2 = 1, kappa3 = 0, mu1 = 0, mu2 = 0, ...)

dwnorm2(
  x,
  kappa1 = 1,
  kappa2 = 1,
  kappa3 = 0,
  mu1 = 0,
  mu2 = 0,
  int.displ,
  log = FALSE
)
```

Arguments

n	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
kappa1, kappa2, kappa3	vectors of concentration parameters; kappa1 , kappa2 > 0, and kappa3^2 < kappa1*kappa2.
mu1, mu2	vectors of mean parameters.
...	additional arguments passed to rmvnorm from package mvtnorm
x	bivariate vector or a two-column matrix with each row being a bivariate vector of angles (in radians) where the densities are to be evaluated.
int.displ	integer displacement. If int.displ = M, then each infinite sum in the density is approximated by a finite sum over 2*M + 1 elements. (See Details.) The allowed values are 1, 2, 3, 4 and 5. Default is 3.
log	logical. Should the log density be returned instead?

Details

The bivariate wrapped normal density at the point $x = (x_1, x_2)$ is given by,

$$f(x) = \sqrt{((\kappa_1\kappa_2 - (\kappa_3)^2))}/(2\pi) \sum \exp(-1/2 * (\kappa_1(T_1)^2 + \kappa_2(T_2)^2 + 2\kappa_3(T_1)(T_2)))$$

where

$$T_1 = T_1(x, \mu, \omega) = (x_1 - \mu_1(2\pi\omega_1))$$
$$T_2 = T_2(x, \mu, \omega) = (x_2 - \mu_1(2\pi\omega_2))$$

the sum extends over all pairs of integers $\omega = (\omega_1, \omega_2)$, and is approximated by a sum over (ω_1, ω_2) in $\{-M, -M+1, ..., M-1, M\}^2$ if `int.displ =` M.

Note that above density is essentially the "wrapped" version of a bivariate normal density with mean

$$\mu = (\mu_1, \mu_2)$$

and dispersion matrix $\Sigma = \Delta^{-1}$, with Δ being a 2×2 symmetric, positive definite matrix with diagonal entries κ_1 and κ_2 and the off-diagonal entries κ_3.

Value

`dwnorm2` gives the density and `rwnorm2` generates random deviates.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
x <- diag(2, 2)
n <- 10

# when x is a bivariate vector and parameters are all scalars,
# dwnorm2 returns single density
dwnorm2(x[1, ], kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when x is a two column matrix and parameters are all scalars,
# dmvsin returns a vector of densities calculated at the rows of
# x with the same parameters
dwnorm2(x, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# if x is a bivariate vector and at least one of the parameters is
# a vector, all parameters are recycled to the same length, and
# dwnorm2 returns a vector of with ith element being the density
# evaluated at x with parameter values kappa1[i], kappa2[i],
# kappa3[i], mu1[i] and mu2[i]
dwnorm2(x[1, ], kappa1, kappa2, kappa3, mu1, mu2)

# if x is a two column matrix and at least one of the parameters is
# a vector, rows of x and the parameters are recycled to the same
# length, and dwnorm2 returns a vector of with ith element being the
```

```
# density evaluated at ith row of x with parameter values kappa1[i],
# kappa2[i], # kappa3[i], mu1[i] and mu2[i]
dwnorm2(x, kappa1, kappa2, kappa3, mu1, mu2)

# when parameters are all scalars, number of observations generated
# by rwnorm2 is n
rwnorm2(n, kappa1[1], kappa2[1], kappa3[1], mu1[1], mu2[1])

# when at least one of the parameters is a vector, all parameters are
# recycled to the same length, n is ignored, and the number of
# observations generated by rwnorm2 is the same as the length of the
# recycled vectors
rwnorm2(n, kappa1, kappa2, kappa3, mu1, mu2)
```

`rwnorm2mix` *The bivariate Wrapped Normal mixtures*

Description

The bivariate Wrapped Normal mixtures

Usage

```
rwnorm2mix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix, ...)

dwnorm2mix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix, int.displ, log = FALSE)
```

Arguments

`n`	number of observations.
`kappa1, kappa2, kappa3`	vectors of concentration parameters; `kappa1 , kappa2 > 0, kappa3^2 < kappa1*kappa2` for each component.
`mu1, mu2`	vectors of mean parameters.
`pmix`	vector of mixture proportions.
`...`	additional arguments passed to rmvnorm from package `mvtnorm`
`x`	matrix of angles (in radians) where the density is to be evaluated, with each row being a single bivariate vector of angles.
`int.displ`	integer displacement. If `int.displ` = M, then each infinite sum in the density is approximated by a finite sum over 2*M + 1 elements. (See Details.) The allowed values are 1, 2, 3, 4 and 5. Default is 3.
`log`	logical. Should the log density be returned instead?

Details

All the argument vectors `pmix`, `kappa1`, `kappa2`, `kappa3`, `mu1` and `mu2` must be of the same length, with j-th element corresponding to the j-th component of the mixture distribution.

The bivariate wrapped normal mixture distribution with component size `K = length(pmix)` has density

$$g(x) = \sum p[j] * f(x; \kappa_1[j], \kappa_2[j], \kappa_3[j], \mu_1[j], \mu_2[j])$$

where the sum extends over j; $p[j]$; $\kappa_1[j], \kappa_2[j], \kappa_3[j]$; and $\mu_1[j], \mu_2[j]$ respectively denote the mixing proportion, the three concentration parameters and the two mean parameter for the j-th component, $j = 1, ..., K$, and $f(.; \kappa_1, \kappa_2, \kappa_3, \mu_1, \mu_2)$ denotes the density function of the wrapped normal distribution with concentration parameters $\kappa_1, \kappa_2, \kappa_3$ and mean parameters μ_1, μ_2.

Value

`dwnorm2mix` computes the density and `rwnorm2mix` generates random deviates from the mixture density.

Examples

```
kappa1 <- c(1, 2, 3)
kappa2 <- c(1, 6, 5)
kappa3 <- c(0, 1, 2)
mu1 <- c(1, 2, 5)
mu2 <- c(0, 1, 3)
pmix <- c(0.3, 0.4, 0.3)
x <- diag(2, 2)
n <- 10

# mixture densities calculated at the rows of x
dwnorm2mix(x, kappa1, kappa2, kappa3, mu1, mu2, pmix)

# number of observations generated from the mixture distribution is n
rwnorm2mix(n, kappa1, kappa2, kappa3, mu1, mu2, pmix)
```

---

rwnormmix *The univariate Wrapped Normal mixtures*

Description

The univariate Wrapped Normal mixtures

Usage

```
rwnormmix(n = 1, kappa, mu, pmix)

dwnormmix(x, kappa, mu, pmix, int.displ = 3, log = FALSE)
```

Arguments

n	number of observations. Ignored if at least one of the other parameters have length k > 1, in which case, all the parameters are recycled to length k to produce k random variates.
kappa	vector of component concentration (inverse-variance) parameters, kappa > 0.
mu	vector of component means.
pmix	vector of mixing proportions.
x	vector of angles (in radians) where the densities are to be evaluated.
int.displ	integer displacement. If int.displ = M, then the infinite sum in the density is approximated by a sum over 2*M + 1 elements. (See Details.) The allowed values are 1, 2, 3, 4 and 5. Default is 3.
log	logical. Should the log density be returned instead?

Details

pmix, mu and kappa must be of the same length, with j-th element corresponding to the j-th component of the mixture distribution.

The univariate wrapped normal mixture distribution with component size K = length(pmix) has density

$$g(x) = p[1] * f(x; \kappa[1], \mu[1]) + ... + p[K] * f(x; \kappa[K], \mu[K])$$

where $p[j], \kappa[j], \mu[j]$ respectively denote the mixing proportion, concentration parameter and the mean parameter for the j-th component and $f(.; \kappa, \mu)$ denotes the density function of the (univariate) wrapped normal distribution with mean parameter μ and concentration parameter κ.

Value

dwnormmix computes the density and rwnormmix generates random deviates from the mixture density.

Examples

```
kappa <- 1:3
mu <- 0:2
pmix <- c(0.3, 0.3, 0.4)
x <- 1:10
n <- 10

# mixture densities calculated at each point in x
dwnormmix(x, kappa, mu, pmix)

# number of observations generated from the mixture distribution is n
rwnormmix(n, kappa, mu, pmix)
```

select_chains *Select chains from angmcmc objects*

Description

Select chains from angmcmc objects

Usage

```
select_chains(object, chain.no, ...)
```

Arguments

object	angular MCMC object.
chain.no	labels of chains to be retained in the final sample. If missing, all chains are used.
...	unused

Value

Returns another angmcmc object with only selected chains passed through chain.no

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             L = c(10, 12), chains_parallel = FALSE,
                             n.chains = 2)
fit.vmsin.20
fit.vmsin.20.1 <- select_chains(fit.vmsin.20, 1)
fit.vmsin.20.1
```

summary.angmcmc *Summary statistics for parameters from an angmcmc object*

Description

Summary statistics for parameters from an angmcmc object

Usage

```
## S3 method for class 'angmcmc'
summary(object, par.name, comp.label, chain.no, ...)
```

Arguments

object	angular MCMC object.
par.name	vector of names of parameters for which point estimates are to be computed. If NULL, results for all parameters are provided.
comp.label	vector of component labels (positive integers, e.g., 1, 2, ...) for which point estimates are to be computed. If NULL, results for all components are provided.
chain.no	vector of chain numbers whose samples are to be be used. in the estimation. By default all chains are used.
...	additional arguments affecting the summary produced.

Details

Computes (after thinning and discarding burn-in) point estimates with 95% posterior credible sets for all components and all parameters, together with the sample averages of log likelihood and log posterior density.

Value

Returns a list with elements estimate, lower, upper, llik and lpd. estimate is itself a list with three elements: mean, median and mode providing the sample mean, sample median and (sample) MAP estimates.

Note that summary.angmcmc has its own print method, providing a table the estimated mean and 95% credible intervals for each parameter

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1)
summary(fit.vmsin.20)
```

---

surface_model *Surface for bivariate angular mixture model densities*

Description

Surface for bivariate angular mixture model densities

Usage

```
surface_model(
  model = "vmsin",
  kappa1,
  kappa2,
  kappa3,
```

```
  mu1,
  mu2,
  pmix = rep(1/length(kappa1), length(kappa1)),
  xpoints = seq(0, 2 * pi, length.out = 30),
  ypoints = seq(0, 2 * pi, length.out = 30),
  log.density = FALSE,
  xlab = "x",
  ylab = "y",
  zlab = ifelse(log.density, "Log Density", "Density"),
  main,
  ...
)
```

Arguments

`model`	bivariate angular model whose mixture is of interest. Must be one of "vmsin", "vmcos" and "wnorm2".
`kappa1`, `kappa2`, `kappa3`, `mu1`, `mu2`, `pmix`	model parameters and mixing proportions. See the respective mixture model densities (dvmsinmix, dvmcosmix, dwnorm2mix) for more details.
`xpoints`	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
`ypoints`	Points on the first (x-) coordinate where the density is to be evaluated. Default to seq(0, 2*pi, length.out=100).
`log.density`	logical. Should log density be used for the plot?
`xlab`, `ylab`, `zlab`, `main`	graphical parameters passed to `lattice::wireframe`
`...`	additional arguments passed to `lattice::wireframe`

Examples

```
surface_model('vmsin', 1, 1, 1.5, pi, pi)
surface_model('vmcos', 1, 1, 1.5, pi, pi)
```

---

tim8 *Backbone Dihedral Angles of Triose Phosphate Isomerase (8TIM)*

Description

A dataset consisting of 490 pairs of backbone dihedral angles (in radian scale $[0, 2\pi)$) (ϕ, ψ) for the protein Triose Phosphate Isomerase (8TIM). The angles were obtained first by using the DSSP software on the PDB file for 8TIM to get the backbone angles (in degrees), and then by converting all angles into radians. Due to the presence of different secondary structures (helices, sheets and loops) in the protein, the angular data show considerable variability, and is multimodal with noticeably distinct clusters.

Usage

```
data(tim8)
```

Format

A data frame with 490 rows and 2 variables (backbone dihedral angles) phi and psi.

Source

8TIM PDB file: http://www.rcsb.org/pdb/explore.do?structureId=8tim.

DSSP software: https://swift.cmbi.umcn.nl/gv/dssp/.

---

vm2_mle *Maximum likelihood estimation of bivariate von Mises parameters*

Description

Maximum likelihood estimation of bivariate von Mises parameters

Usage

```
vm2_mle(data, model = c("vmsin", "vmcos"), ...)
```

Arguments

data	data matrix (if bivarate, in which case it must have two columns) or vector. If outside, the values are transformed into the scale $[0, 2\pi)$. *Note:* BAMBI cannot handle missing data. Missing values must either be removed or properly imputed.
model	Bivariate von Mises model. One of "vmsin", "vmcos" or "indep".
...	Additional arguments. See details.

Details

The parameters kappa1 and kappa2 are optimized in log scales. The method of optimization used (passed to optim) can be specified through method in ... (defaults to "L-BFGS-B"). Note, however, that lower (0) and upper (2*pi) bounds for mu1 and mu2 are specified; so not all methods implemented in optim will work.

Value

An object of class mle-class.

Examples

```
pars <- list(kappa1 = 3, kappa2 = 2, kappa3 = 1.5, mu1 = 0.5, mu2 = 1.5)
nsamp <- 2000
model <- "vmsin"
set.seed(100)
dat_gen <- do.call(paste0("r", model), c(list(n = nsamp), pars))

est <- vm2_mle(dat_gen, model = model)
library(stats4)
coef(est)
vcov(est)
```

`waic.angmcmc` *Watanabe-Akaike Information Criterion (WAIC) for angmcmc objects*

Description

Watanabe-Akaike Information Criterion (WAIC) for angmcmc objects

Usage

```
## S3 method for class 'angmcmc'
waic(x, ...)
```

Arguments

`x`	angmcmc object.
`...`	additional model specific arguments to be passed to waic from loo. For example, `int.displ` specifies integer displacement in wnorm and wnorm2 models. See fit_wnormmix and fit_wnorm2mix for more details.

Details

Given a deviance function $D(\eta) = -2\log(p(y|\eta))$, and an estimate $\eta* = (\sum \eta_i)/n$ of the posterior mean $E(\eta|y)$, where $y = (y_1, ..., y_n)$ denote the data, η is the unknown parameter vector of the model, $\eta_1, ..., \eta_N$ are MCMC samples from the posterior distribution of η given y and $p(y|\eta)$ is the likelihood function, the Watanabe-Akaike Information Criterion (WAIC) is defined as

$$WAIC = LPPD - p_W$$

where

$$LPPD = \sum_{i=1}^{n} \log(N^{-1} \sum_{s=1}^{N} p(y_i|\eta_s))$$

and (form 1 of)

$$p_W = 2\sum_{i=1}^{n} [\log(N^{-1} \sum_{s=1}^{N} p(y_i|\eta_s)) - N^{-1} \sum_{s=1}^{N} \log\ p(y_i|\eta_s)].$$

An alternative form (form 2) for p_W is given by

$$p_W = \sum_{i=1}^{n} \hat{var} \log p(y_i|\eta)$$

where for $i = 1, ..., n$, $\hat{var} \log p(y_i|\eta)$ denotes the estimated variance of $\log p(y_i|\eta)$ based on the realizations $\eta_1, ..., \eta_N$.

Note that waic.angmcmc calls waic for computation. If the likelihood contribution of each data point for each MCMC iteration is available in `object` (can be returned by setting `return_llik_contri = TRUE`) during fit_angmix call), `waic.array` is used; otherwise `waic.function` is called. Computation is much faster if the likelihood contributions are available - however, they are very memory intensive, and by default not returned in fit_angmix.

Value

Computes the WAIC for a given angmcmc object.

Examples

```
# illustration only - more iterations needed for convergence
fit.vmsin.20 <- fit_vmsinmix(tim8, ncomp = 3, n.iter =  20,
                             n.chains = 1, return_llik_contri = TRUE)
library(loo)
waic(fit.vmsin.20)
```

`wind` *Saturna Island wind directions*

Description

A dataset consisting of 239 observations on wind direction in radians (original measurements were in 10s of degrees), measured at Saturna Island, British Columbia, Canada during October 1-10, 2016 (obtained from Environment Canada website). There was a severe storm during October 4-7, which caused significant fluctuations among the wind directions. As a result the angular data show a clear multimodality.

Usage

```
data(wind)
```

Format

A data frame with 239 rows and 2 columns; the column "angle" provides the angular direction (in radian) and the column day provides the days on which the data points were collected (ranges between 1-10, corresponding to October 1-10, 2016).

Source

Environment Canada: https://climate.weather.gc.ca/climate_data/data_quality_e.html.

CBC news on the storm: https://www.cbc.ca/news/canada/british-columbia/storm-bc-1.3795204.

`zero_to_2pi` *Wrap angles into* `[-pi, pi]` *or* `[0, 2*pi]`

Description

Wrap angles into `[-pi, pi]` or `[0, 2*pi]`

Usage

```
zero_to_2pi(x)

minuspi_to_pi(x)
```

Arguments

`x`	numeric vector or matrix or data.frame.

Details

`minuspi_to_pi` wraps `x` into `[-pi, pi]`, while `zero_to_pi` wraps `x` into `[0, 2*pi]`.

Examples

```
dat <- matrix(runif(100, -pi, pi), ncol=2)
dat1 <- zero_to_2pi(dat)
dat2 <- minuspi_to_pi(dat1)
all.equal(dat, dat2)
```

Index