

# Package ‘bkmrhat’

October 12, 2020

**Title** Parallel Chain Tools for Bayesian Kernel Machine Regression

**Version** 1.0.0

**Date** 2020-10-12

**Author** Alexander Keil [aut, cre]

**Maintainer** Alexander Keil <akeil@unc.edu>

**Description** Bayesian kernel machine regression (from the 'bkmr' package) is a Bayesian semi-parametric generalized linear model approach under identity and probit links. There are a number of functions in this package that extend Bayesian kernel machine regression fits to allow multiple-chain inference and diagnostics, which leverage functions from the 'future', 'rstan', and 'coda' packages. Reference: Bobb, J. F., Henn, B. C., Valeri, L., & Coull, B. A. (2018). Statistical software for analyzing the health effects of multiple concurrent exposures via Bayesian kernel machine regression. ; <doi:10.1186/s12940-018-0413-y>.

**License** GPL (>= 3)

**Depends** coda, R (>= 3.5.0)

**Imports** bkmr, future, rstan

**Suggests** knitr, markdown

**VignetteBuilder** knitr

**Encoding** UTF-8

**Language** en-US

**LazyData** true

**RoxygenNote** 7.1.0

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2020-10-12 08:00:03 UTC

## R topics documented:

as.mcmc.bkmrfit	2
as.mcmc.list.bkmrfit.list	3
ExtractPIPs_parallel	4
kmbayes_combine	5
kmbayes_continue	6
kmbayes_diagnose	7
kmbayes_parallel	8
kmbayes_parallel_continue	9
OverallRiskSummaries_parallel	10
predict.bkmrfit	10
PredictorResponseBivar_parallel	11
PredictorResponseUnivar_parallel	12
SamplePred_parallel	12
SingVarRiskSummaries_parallel	13
<b>Index</b>	<b>14</b>

---

as.mcmc.bkmrfit	<i>Convert bkmrfit to mcmc object for coda MCMC diagnostics</i>
-----------------	---

---

### Description

Converts a `kmrfit` (from the `bkmr` package) into an `mcmc` object from the `coda` package. The `coda` package enables many different types of single chain MCMC diagnostics, including `geweke.diag`, `traceplot` and `effectiveSize`. Posterior summarization is also available, such as `HPDinterval` and `summary.mcmc`.

### Usage

```
## S3 method for class 'bkmrfit'
as.mcmc(x, iterstart = 1, thin = 1, ...)
```

### Arguments

<code>x</code>	object of type <code>kmrfit</code> (from <code>bkmr</code> package)
<code>iterstart</code>	first iteration to use (e.g. for implementing burnin)
<code>thin</code>	keep 1/thin % of the total iterations (at regular intervals)
<code>...</code>	unused

### Value

An `mcmc` object

## Examples

```
# following example from https://jenfb.github.io/bkmr/overview.html

set.seed(111)
library(coda)
library(bkmr)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
set.seed(111)
fitkm <- kmbayes(y = y, Z = Z, X = X, iter = 500, verbose = FALSE,
  varsel = FALSE)
mcmcobj <- as.mcmc(fitkm, iterstart=251)
summary(mcmcobj) # posterior summaries of model parameters
# compare with default from bkmr package, which omits first 1/2 of chain
summary(fitkm)
# note this only works on multiple chains (see kmbayes_parallel)
# gelman.diag(mcmcobj)
# lots of functions in the coda package to use
traceplot(mcmcobj)
# will also fail with delta functions (when using variable selection)
try(geweke.plot(mcmcobj))
```

---

```
as.mcmc.list.bkmrfit.list
```

*Convert multi-chain bkmrfit to mcmc.list for coda MCMC diagnostics*

---

## Description

Converts a `kmrfit.list` (from the `bkmrhat` package) into an `mcmc.list` object from the `coda` package. The `coda` package enables many different types of MCMC diagnostics, including `geweke.diag`, `traceplot` and `effectiveSize`. Posterior summarization is also available, such as `HPDinterval` and `summary.mcmc`. Using multiple chains is necessary for certain MCMC diagnostics, such as `gelman.diag`, and `gelman.plot`.

## Usage

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

## Arguments

```
x          object of type kmrfit.list (from bkmrhat package)
...        arguments to as.mcmc.bkmrfit
```

**Value**

An `mcmc.list` object

**Examples**

```
# following example from https://jenfb.github.io/bkmr/overview.html

set.seed(111)
library(coda)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
set.seed(111)
Sys.setenv(R_FUTURE_SUPPORTSMULTICORE_UNSTABLE="quiet")
future::plan(strategy = future::multiprocess, workers=2)
# run 2 parallel Markov chains (more usually better)
fitkm.list <- kmbayes_parallel(nchains=2, y = y, Z = Z, X = X, iter = 1000,
  verbose = FALSE, varsel = FALSE)
mcmcobj = as.mcmc.list(fitkm.list)
summary(mcmcobj)
# Gelman/Rubin diagnostics won't work on certain objects,
# like delta parameters (when using variable selection),
# so the rstan version of this will work better (does not give errors)
try(gelman.diag(mcmcobj))
# lots of functions in the coda package to use
plot(mcmcobj)
# both of these will also fail with delta functions (when using variable selection)
try(gelman.plot(mcmcobj))
try(geweke.plot(mcmcobj))

closeAllConnections()
```

---

ExtractPIPs\_parallel *Posterior inclusion probabilities by chain*

---

**Description**

Posterior inclusion probabilities by chain

**Usage**

```
ExtractPIPs_parallel(x, ...)
```

**Arguments**

`x` bkmrfit.list object from `kmbayes_parallel`  
`...` arguments to `CalcPIPs`

**Value**

data.frame with all chains together

---

kmbayes_combine	<i>Combine multiple BKMR chains</i>
-----------------	-------------------------------------

---

**Description**

Combine multiple chains comprising BKMR fits at different starting values.

**Usage**

```
kmbayes_combine(fitkm.list, burnin = 0, reorder = TRUE)
```

```
comb_bkmrfits(fitkm.list, burnin = 0, reorder = TRUE)
```

**Arguments**

fitkm.list	output from <a href="#">kmbayes_parallel</a>
burnin	add in custom burnin (number of burnin iterations per chain)
reorder	ensures that the first half of the combined chain contains only the first half of each individual chain - this allows unaltered use of standard functions from bkmr package, which automatically trims the first half of the iterations. This can be used for posterior summaries, but certain diagnostics may not work well (autocorrelation, effective sample size) so the diagnostics should be done on the individual chains #' @param ... arguments to <a href="#">as.mcmc.bkmrfit</a>

**Details**

Chains are not combined fully sequentially

**Value**

a bkmrplusfit object, which inherits from bkmrfit (from the [kmbayes](#) function) with multiple chains combined into a single object and additional parameters given by chain and iters, which index the specific chains and iterations for each posterior sample in the bkmrplusfit object

**Examples**

```
# following example from https://jenfb.github.io/bkmr/overview.html
set.seed(111)
library(bkmr)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
```

```

set.seed(111)
Sys.setenv(R_FUTURE_SUPPORTSMULTICORE_UNSTABLE="quiet")
future::plan(strategy = future::multiprocess, workers=2)
# run 4 parallel Markov chains (low iterations used for illustration)
fitkm.list <- kmbayes_parallel(nchains=2, y = y, Z = Z, X = X, iter = 500,
  verbose = FALSE, varsel = TRUE)
bigkm = kmbayes_combine(fitkm.list)
ests = ExtractEsts(bigkm)
ExtractPIPs(bigkm)
pred.resp.univar <- PredictorResponseUnivar(fit = bigkm)
risks.overall <- OverallRiskSummaries(fit = bigkm, y = y, Z = Z, X = X,
  qs = seq(0.25, 0.75, by = 0.05), q.fixed = 0.5, method = "exact")

# additional objects that are not in a standard bkmrfit object:
summary(bigkm$iters)
table(bigkm$chain)

closeAllConnections()

```

---

kmbayes\_continue

*Continue sampling from existing bkmr fit*


---

## Description

Use this when you've used MCMC sampling with the [kmbayes](#) function, but you did not take enough samples and do not want to start over.

## Usage

```
kmbayes_continue(fit, ...)
```

## Arguments

<code>fit</code>	output from <a href="#">kmbayes</a>
<code>...</code>	arguments to <a href="#">kmbayes_continue</a>

## Details

Note this does not fully start from the prior values of the MCMC chains. The [kmbayes](#) function does not allow full specification of the kernel function parameters, so this will restart the chain at the last values of all fixed effect parameters, and start the kernel  $r$  parameters at the arithmetic mean of all  $r$  parameters from the last step in the previous chain.

## Value

a `bkmrfit.continued` object, which inherits from `bkmrfit` objects similar to [kmbayes](#) output, and which can be used to make inference using functions from the `bkmr` package.

**See Also**[kmbayes\\_parallel](#)**Examples**

```
set.seed(111)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
## Not run:
fitty1 = bkmr::kmbayes(y=y,Z=Z,X=X, est.h=TRUE, tier=100)
# do some diagnostics here to see if 100 iterations (default) is enough
# add 100 additional iterations (for illustration - still will not be enough)
fitty2 = kmbayes_continue(fitty1, iter=100)
cobj = as.mcmc(fitty2)
varnames(cobj)

## End(Not run)
```

---

`kmbayes_diagnose`*MCMC diagnostics using rstan*

---

**Description**

Give MCMC diagnostics from the `rstan` package using the [Rhat](#), [ess\\_bulk](#), and [ess\\_tail](#) functions. Note that r-hat is only reported for `bkmrfit.list` objects from [kmbayes\\_parallel](#)

**Usage**

```
kmbayes_diagnose(kmobj, ...)
```

```
kmbayes_diag(kmobj, ...)
```

**Arguments**

<code>kmobj</code>	Either an object from <a href="#">kmbayes</a> or from <a href="#">kmbayes_parallel</a>
<code>...</code>	arguments to <a href="#">monitor</a>

**Examples**

```
set.seed(111)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
```

```

set.seed(111)
Sys.setenv(R_FUTURE_SUPPORTSMULTICORE_UNSTABLE="quiet")
future::plan(strategy = future::multiprocess)
fitkm.list <- kmbayes_parallel(nchains=2, y = y, Z = Z, X = X, iter = 1000,
  verbose = FALSE, varsel = TRUE)
kmbayes_diag(fitkm.list)
kmbayes_diag(fitkm.list[[1]]) # just the first chain

closeAllConnections()

```

---

kmbayes\_parallel      *Run multiple BKMR chains in parallel*

---

### Description

Fit parallel chains from the [kmbayes](#) function. These chains leverage parallel processing from the `future` package, which can speed fitting and enable diagnostics that rely on multiple Markov chains from dispersed initial values.

### Usage

```
kmbayes_parallel(nchains = 4, ...)
```

### Arguments

nchains	number of parallel chains
...	arguments to <code>kmbayes</code>

### Value

a "bkmrfit.list" object, which is just an R list object in which each entry is a "bkmrfit" object [kmbayes](#)

### Examples

```

set.seed(111)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
set.seed(111)
Sys.setenv(R_FUTURE_SUPPORTSMULTICORE_UNSTABLE="quiet")
future::plan(strategy = future::multiprocess, workers=2)
# only 50 iterations fit to save installation time
fitkm.list <- kmbayes_parallel(nchains=2, y = y, Z = Z, X = X, iter = 50,
  verbose = FALSE, varsel = TRUE)
closeAllConnections()

```



---

`kmbayes_parallel_continue`*Continue sampling from existing `bkmr_parallel` fit*

---

## Description

Use this when you've used MCMC sampling with the `kmbayes_parallel` function, but you did not take enough samples and do not want to start over.

## Usage

```
kmbayes_parallel_continue(fitkm.list, ...)
```

## Arguments

<code>fitkm.list</code>	output from <code>kmbayes_parallel</code>
<code>...</code>	arguments to <code>kmbayes_continue</code>

## Value

a `bkmrfit.list` object, which is just a list of `bkmrfit` objects similar to `kmbayes_parallel`

## See Also

[kmbayes\\_parallel](#)

## Examples

```
set.seed(111)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
## Not run:
Sys.setenv(R_FUTURE_SUPPORTSMULTICORE_UNSTABLE="quiet")
future::plan(strategy = future::multiprocess, workers=2)
fitty1p = kmbayes_parallel(nchains=2, y=y,Z=Z,X=X)

fitty2p = kmbayes_parallel_continue(fitty1p, iter=3000)
cobj = as.mcmc.list(fitty2p)
plot(cobj)

## End(Not run)
```

---

OverallRiskSummaries\_parallel  
*Overall summary by chain*

---

**Description**

Overall summary by chain

**Usage**

```
OverallRiskSummaries_parallel(x, ...)
```

**Arguments**

x                    bkmrfit.list object from [kmbayes\\_parallel](#)  
...                    arguments to [OverallRiskSummaries](#)

**Value**

data.frame with all chains together

---

predict.bkmrfit            *Posterior mean/sd predictions*

---

**Description**

Provides observation level predictions based on the posterior mean, or, alternatively, yields the posterior standard deviations of predictions for an observation. This function is useful for interfacing with ensemble machine learning packages such as SuperLearner, which utilize only point estimates.

**Usage**

```
## S3 method for class 'bkmrfit'
predict(object, ptype = c("mean", "sd.fit"), ...)
```

**Arguments**

object                fitted object of class inheriting from "bkmrfit".  
ptype                 "mean" or "sd.fit", where "mean" yields posterior mean prediction for every observation in the data, and "sd.fit" yields the posterior standard deviation for every observation in the data.  
...                    arguments to [SamplePred](#)

**Value**

vector of predictions the same length as the outcome in the bkmrfit object

**Examples**

```
# following example from https://jenfb.github.io/bkmr/overview.html

library(bkmr)
set.seed(111)
dat <- bkmr::SimData(n = 50, M = 4)
y <- dat$y
Z <- dat$Z
X <- dat$X
set.seed(111)
fitkm <- kmbayes(y = y, Z = Z, X = X, iter = 200, verbose = FALSE,
  varsel = TRUE)
postmean = predict(fitkm)
postmean2 = predict(fitkm, Znew=Z/2)
# mean difference in posterior means
mean(postmean-postmean2)
```

---

PredictorResponseBivar\_parallel

*Bivariate predictor response by chain*

---

**Description**

Bivariate predictor response by chain

**Usage**

```
PredictorResponseBivar_parallel(x, ...)
```

**Arguments**

x                    bkmrfit.list object from [kmbayes\\_parallel](#)  
...                   arguments to [PredictorResponseBivar](#)

**Value**

data.frame with all chains together

---

PredictorResponseUnivar\_parallel

*Univariate predictor response summary by chain*

---

**Description**

Univariate predictor response summary by chain

**Usage**

PredictorResponseUnivar\_parallel(x, ...)

**Arguments**

x                    bkmrfit.list object from [kmbayes\\_parallel](#)  
...                   arguments to [PredictorResponseUnivar](#)

**Value**

data.frame with all chains together

---

SamplePred\_parallel    *Posterior samples of  $E(Y|h(Z),X,\beta)$  by chain*

---

**Description**

Posterior samples of  $E(Y|h(Z),X,\beta)$  by chain

**Usage**

SamplePred\_parallel(x, ...)

**Arguments**

x                    bkmrfit.list object from [kmbayes\\_parallel](#)  
...                   arguments to [CalcPIPs](#)

**Value**

data.frame with all chains together

---

`SingVarRiskSummaries_parallel`  
*Single variable summary by chain*

---

**Description**

Single variable summary by chain

**Usage**

```
SingVarRiskSummaries_parallel(x, ...)
```

**Arguments**

`x`                    `bkmrfit.list` object from [kmbayes\\_parallel](#)  
`...`                    arguments to [SingVarRiskSummaries](#)

**Value**

`data.frame` with all chains together

# Index

as.mcmc.bkmrfit, [2, 3, 5](#)  
as.mcmc.list.bkmrfit.list, [3](#)

CalcPIPs, [4, 12](#)  
comb\_bkmrfits (kmbayes\_combine), [5](#)

effectiveSize, [2, 3](#)  
ess\_bulk, [7](#)  
ess\_tail, [7](#)  
ExtractPIPs\_parallel, [4](#)

gelman.diag, [3](#)  
gelman.plot, [3](#)  
geweke.diag, [2, 3](#)

HPDinterval, [2, 3](#)

kmbayes, [5–8](#)  
kmbayes\_combine, [5](#)  
kmbayes\_continue, [6, 6, 9](#)  
kmbayes\_diag (kmbayes\_diagnose), [7](#)  
kmbayes\_diagnose, [7](#)  
kmbayes\_parallel, [4, 5, 7, 8, 9–13](#)  
kmbayes\_parallel\_continue, [9](#)

mcmc, [2](#)  
mcmc.list, [3, 4](#)  
monitor, [7](#)

OverallRiskSummaries, [10](#)  
OverallRiskSummaries\_parallel, [10](#)

predict.bkmrfit, [10](#)  
PredictorResponseBivar, [11](#)  
PredictorResponseBivar\_parallel, [11](#)  
PredictorResponseUnivar, [12](#)  
PredictorResponseUnivar\_parallel, [12](#)

Rhat, [7](#)

SamplePred, [10](#)

SamplePred\_parallel, [12](#)  
SingVarRiskSummaries, [13](#)  
SingVarRiskSummaries\_parallel, [13](#)  
summary.mcmc, [2, 3](#)

traceplot, [2, 3](#)