# Package ‘bkmrhat’

October 12, 2022

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

**Version** 1.1.3

**Date** 2022-03-29

**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, data.table, future, rstan

**Suggests** knitr, markdown

**VignetteBuilder** knitr

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 7.1.2

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2022-03-29 08:50:05 UTC

## R topics documented:

---

## `as.mcmc.bkmrfit` *Convert bkmrfit to mcmc object for coda MCMC diagnostics*

### 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

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

future::plan(strategy = future::multisession, 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 `ExtractPIPs` |

### Value

data.frame with all chains together

---

## `kmbayes_combine` *Combine multiple BKMR chains*

### Description

Combine multiple chains comprising BKMR fits at different starting values.

### Usage

```
kmbayes_combine(
  fitkm.list,
  burnin = NULL,
  excludeburnin = FALSE,
  reorder = TRUE
)

comb_bkmrfits(fitkm.list, burnin = NULL, excludeburnin = FALSE, reorder = TRUE)
```

### Arguments

| | |
|---|---|
| `fitkm.list` | output from `kmbayes_parallel` |
| `burnin` | (numeric, or default=NULL) add in custom burnin (number of burnin iterations per chain). If NULL, then default to half of the chain |
| `excludeburnin` | (logical, default=FALSE) should burnin iterations be excluded from the final chains? Note that all bkmr package functions automatically exclude burnin from calculations. |
| `reorder` | (logical, default=TRUE) 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 `as.mcmc.bkmrfit` |

### 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)

future::plan(strategy = future::multisession, 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)
# use bkmr defaults for burnin, but keep them
bigkm = kmbayes_combine(fitkm.list, excludeburnin=FALSE)
ests = ExtractEsts(bigkm) # defaults to keeping second half of samples
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$iters) # note that this reflects how fits are re-ordered to reflect burnin
table(bigkm$chain)


closeAllConnections()
```

---

# `kmbayes_combine_lowmem` *Combine multiple BKMR chains in lower memory settings*

## Description

Combine multiple chains comprising BKMR fits at different starting values. This function writes some results to disk, rather than trying to process fully within memory which, in some cases, will result in avoiding "out of memory" errors that can happen with kmbayes_combine.

## Usage

```
kmbayes_combine_lowmem(
  fitkm.list,
  burnin = NULL,
  excludeburnin = FALSE,
  reorder = TRUE
)
```

```
comb_bkmrfits_lowmem(
  fitkm.list,
  burnin = NULL,
  excludeburnin = FALSE,
  reorder = TRUE
)
```

### Arguments

| | |
|---|---|
| `fitkm.list` | output from `kmbayes_parallel` |
| `burnin` | (numeric, or default=NULL) add in custom burnin (number of burnin iterations per chain). If NULL, then default to half of the chain |
| `excludeburnin` | (logical, default=FALSE) should burnin iterations be excluded from the final chains? Note that all bkmr package functions automatically exclude burnin from calculations. |
| `reorder` | (logical, default=TRUE) 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 `as.mcmc.bkmrfit` |

### Details

Chains are not combined fully sequentially (see "reorder")

### 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)

future::plan(strategy = future::multisession, 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)
```

```
# use bkmr defaults for burnin, but keep them
bigkm = kmbayes_combine_lowmem(fitkm.list, excludeburnin=FALSE)
ests = ExtractEsts(bigkm) # defaults to keeping second half of samples
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$iters) # note that this reflects how fits are re-ordered to reflect burnin
table(bigkm$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

| | |
|---|---|
| `fit` | output from `kmbayes` |
| `...` | arguments to `kmbayes_continue` |

### 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` parmeters 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, iter=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 diagnostistics 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

| | |
|---|---|
| `kmobj` | Either an object from `kmbayes` or from `kmbayes_parallel` |
| `...` | arguments to `monitor` |

## Examples

```

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

future::plan(strategy = future::multisession)
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 kmbayes |

### 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)

future::plan(strategy = future::multisession, 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

| | |
|---|---|
| `fitkm.list` | output from `kmbayes_parallel` |
| `...` | arguments to `kmbayes_continue` |

## 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:

future::plan(strategy = future::multisession, 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 `SamplePred` |

### 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