Package 'BayesPIM'

July 21, 2025

Title Bayesian Prevalence-Incidence Mixture Model

Version 1.0.0

Description Models time-to-event data from interval-censored screening studies. It accounts for latent prevalence at baseline and incorporates misclassification due to imperfect test sensitivity. For usage details, see the package vignette (``BayesPIM_intro''). Further details can be found in T. Klausch, B. I. Lissenberg-Witte, and V. M. Coupe (2024), ``A Bayesian prevalence-incidence mixture model for screening outcomes with misclassification'', <doi:10.48550/arXiv.2412.16065>.

License MIT + file LICENSE

URL https://github.com/thomasklausch2/bayespim

BugReports https://github.com/thomasklausch2/BayesPIM/issues

Encoding UTF-8

RoxygenNote 7.3.2

LinkingTo Rcpp

Imports Rcpp, mvtnorm, MASS, ggamma, doParallel, foreach, parallel, actuar

Depends R (>= 3.5.0), coda

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Thomas Klausch [aut, cre]

Maintainer Thomas Klausch <t.klausch@amsterdamumc.nl>

Repository CRAN

Date/Publication 2025-03-22 11:40:02 UTC

Contents

`bayes.2S` *Fitting Bayesian Prevalence-Incidence Mixture Model*

Description

Estimates the Pattern Mixture model of Klausch et al. (2025) using a Bayesian Gibbs sampler. The model is formulated as an interval-censored survival model over successive intervals, with the possibility of missed events due to imperfect test sensitivity. In addition, baseline tests at time zero may fail to detect pre-study events (prevalence).

Usage

```
bayes.2S(
  Vobs,
  Z.X = NULL,
  Z.W = NULL,
  r = NULL,
  dist.X = "weibull",
  kappa = 0.5,
  update.kappa = FALSE,
  kappa.prior = NULL,
  ndraws = 1000,
  prop.sd.X = NULL,
  chains = 3,
  thining = 1,
  parallel = TRUE,
  update.till.converge = FALSE,
  maxit = Inf,
  conv.crit = "upper",
  min_effss = chains * 10,
  beta.prior = "norm",
  beta.prior.X = 1,
  sig.prior.X = 1,
  tau.w = 1,
  fix.sigma.X = FALSE,
```

```
  prev.run = NULL,
  update.burnin = TRUE,
  ndraws.update = NULL,
  prev = TRUE,
  vanilla = FALSE,
  ndraws.naive = 10000,
  naive.run.prop.sd.X = prop.sd.X,
  par.exp = FALSE,
  collapsed.g = TRUE,
  k.prior = 1,
  fix.k = FALSE
)
```

Arguments

`Vobs`	A list of length n of numeric vectors representing screening times. The first element of each vector should always be `0` and the last element `Inf` in the case of right censoring.
`Z.X`	A numeric matrix of dimension $n \times p_x$ containing covariates for the AFT incidence model. Missing values are not allowed.
`Z.W`	A numeric matrix of dimension $n \times p_w$ containing covariates for the probit prevalence model. Missing values are not allowed.
`r`	A binary vector of length n indicating whether a baseline test was conducted (`1` for yes, `0` for no / missing baseline test).
`dist.X`	Character. Specifies the distribution for the time-to-incidence variable; choices are `'weibull'`, `'lognormal'`, or `'loglog'` (log-logistic).
`kappa`	Numeric. The test sensitivity value to be used if `update.kappa = FALSE`; otherwise, the starting value for estimating κ.
`update.kappa`	Logical. If `TRUE`, the test sensitivity (κ) is updated during the Gibbs sampler.
`kappa.prior`	A numeric vector of length 2. When specified, a Beta distribution prior is used for κ, centered at `kappa.prior[1]` with standard deviation `kappa.prior[2]`. If `NULL`, a uniform prior (Beta(1,1)) is used.
`ndraws`	Integer. The total number of MCMC draws for the main Gibbs sampler.
`prop.sd.X`	Numeric. The standard deviation for the proposal (jumping) distribution in the Metropolis sampler used for updating β_{xj}. Can be searched for automatically using search.prop.sd.
`chains`	Integer. The number of MCMC chains to run.
`thining`	Integer. The thinning interval for the MCMC sampler.
`parallel`	Logical. If `TRUE`, parallel processing is enabled for the Gibbs sampler. Each chain is assigned to one core (via the `foreach` package). Alternatively, use bayes.2S_seq which employs a `for` loop over the chains.
`update.till.converge`	Logical. If `TRUE`, the model is updated iteratively until convergence criteria are met. Convergence is assessed using the Gelman-Rubin diagnostic ($R < 1.1$) and a minimum effective sample size (`min_effss`) for each parameter, respectively.

`maxit` Numeric. The maximum number of MCMC draws allowed before interrupting the update process when `update.till.converge` is enabled. Default is `Inf` (i.e., no automatic interruption).

`conv.crit` Character. Specifies whether the convergence check uses the point estimate (`'point'`) or the upper bound (`'upper'`) of the Gelman-Rubin diagnostic estimate ($\hat{R}$).

`min_effss` Integer. The minimum effective sample size required for each parameter before convergence is accepted during iterative updating.

`beta.prior` Character. Specifies the type of prior for the incidence regression coefficients (β_{xj}); options are `'norm'` for normal and `'t'` for student-t.

`beta.prior.X` Numeric. The hyperparameter for the prior distribution of the regression coefficients (β_{xj}) in the AFT incidence model. For a normal prior, this is the standard deviation; for a student-t prior, it represents the degrees of freedom. The default produces a standard-normal prior.

`sig.prior.X` Numeric. The hyperparameter (standard deviation) for a half-normal prior on the scale parameter (σ) of the AFT incidence model.

`tau.w` Numeric. The hyperparameter (standard deviation) for the normal prior distribution of the regression coefficients (β_{wj}) in the probit prevalence model. The default produces a standard-normal prior.

`fix.sigma.X` Logical. If `TRUE`, the scale parameter (σ) in the AFT incidence model is fixed at the value provided in `sig.prior.X`; if `FALSE`, it is updated.

`prev.run` Optional. An object of class `BayesPIM` containing results from a previous run. When provided, data and prior settings are adopted from the previous run, and the MCMC chain continues from the last draw.

`update.burnin` Logical. If `TRUE` (default) and `prev.run` is provided, the burn-in period is updated to half of the total draws (sum of previous and current runs); otherwise, the burn-in is maintained as half of the draws from the initial run.

`ndraws.update` Integer. The number of MCMC draws for updating a previous run or for convergence updates. If unspecified, `ndraws` is used.

`prev` Logical. If `TRUE`, prevalence adjustment is applied; if `FALSE`, prevalence is assumed to be zero.

`vanilla` Logical. If `TRUE`, a vanilla run is performed that assumes no prevalence adjustment and fixes $\kappa = 1$, equivalent to a standard Bayesian interval-censored survival regression.

`ndraws.naive` Integer. The number of MCMC draws for a preliminary vanilla run used to obtain starting values. Increase if initial values lead to issues (e.g., an infinite posterior).

`naive.run.prop.sd.X`
Numeric. The standard deviation for the proposal distribution used in the vanilla run. Adjust only if the acceptance rate is significantly off target, as indicated by an interruption message.

`par.exp` Logical. If `TRUE`, the parameter expansion technique of Liu & Wiu (1999) with a Haar prior is employed for updating the regression coefficients (β_{wj}) in the prevalence model. Experimental: tests suggest that it does not improve convergence or reduce autocorrelation.

collapsed.g	Logical. If TRUE, the latent prevalence class membership update in the Gibbs sampler is integrated (collapsed) over the latent incidence time variable. This setting is recommended to improve convergence.
k.prior	Experimental prior parameter for generalized gamma; currently not used.
fix.k	Experimental fixing of prior parameter for generalized gamma; currently not used.

Details

This Bayesian prevalence-incidence mixture model (PIM) characterizes the time to incidence using an accelerated failure time (AFT) model of the form:

$$\log(x_i) = \boldsymbol{z}'_{xi}\boldsymbol{\beta}_x + \sigma\epsilon_i$$

where ϵ_i is chosen such that x_i follows a weibull, lognormal, or loglog (log-logistic) distribution, as specified by the dist argument. The covariate vector $\boldsymbol{z}_{xi}$ for individual i is provided in the Z.X matrix.

Baseline prevalence is modeled using a probit formulation $Pr(g_i = 1|\boldsymbol{z}_{wi}) = Pr(w_i > 0|\boldsymbol{z}_{wi})$ with

$$w_i = \boldsymbol{z}'_{wi}\boldsymbol{\beta}_w + \psi_i$$

where ψ_i follows a standard normal distribution, and the covariate vector $\boldsymbol{z}_{wi}$ is given in the Z.W matrix. The latent variable w_i determines prevalence status, such that $g_i = 1$ if $w_i > 0$ and $g_i = 0$ otherwise.

The argument Vobs provides the observed testing times for all individuals. It is a list of numeric vectors, where each vector starts with 0 (representing the baseline time) and is followed by one or more screening times. The final entry is Inf in the case of right censoring or indicates the time of a positive test if an event is observed. Specifically:

- If the baseline test is positive, the vector consists solely of c(0).
- If the baseline test is negative and right censoring occurs before the first regular screening, the vector is c(0, Inf).
- Otherwise, the vector ends with Inf in the case of right censoring (e.g., c(0, 1, 3, 6, Inf)) or ends at the event time (e.g., c(0, 1, 3, 6) for an event detected at time 6).

By convention, every vector in Vobs starts with 0. However, the binary vector r of length n indicates whether the baseline test was conducted (r[i] = 1) or missing (r[i] = 0) for each individual i in Vobs. For further details on coding, see Section 2 of the main paper.

Test sensitivity can be fixed to a value kappa by setting update.kappa = FALSE, or it can be estimated if update.kappa = TRUE. When estimated, a Beta prior is used, centered on the first element of kappa.prior, with a standard deviation equal to its second element. An internal optimization process finds the Beta prior hyperparameters that best match this choice. If the chosen prior is not feasible, unexpected behavior may occur. If kappa.prior is not specified (the default), an uninformative uniform(0,1) prior is used. In general, we advise against using an uninformative prior, but this default avoids favoring any specific informative prior.

The Gibbs sampler runs for ndraws iterations for each of chains total chains. The Metropolis step used for sampling the parameters of the incidence model applies a normal proposal (jumping) distribution with a standard deviation prop.sd.X, which must be selected by trial and error. An optimal acceptance rate is approximately 23%, which can be computed per MCMC run from the model output. Alternatively, the function search.prop.sd provides a heuristic for selecting an effective proposal distribution standard deviation.

If parallel = TRUE, the Gibbs sampler runs in parallel with one chain per CPU (if possible), using the foreach package. If this package causes issues on some operating systems, set parallel = FALSE or use the bayes.2S_seq function, which iterates over 1:chains using a for loop. This sequential function may also be useful in Monte Carlo simulations that parallelize experimental replications using foreach.

We recommend running at least two chains in parallel, and preferably more, to facilitate standard MCMC diagnostics such as the Gelman-Rubin R statistic. Additionally, we suggest first running the sampler for a moderate number of iterations to assess its behavior before using the updating functionality in prev.run to extend sampling (see below).

The option update.till.convergence = TRUE allows bayes.2S to run until convergence. Convergence is achieved when $R < 1.1$ for all parameters and the minimum effective sample size min_effs is reached. The sampler continues updating until convergence is attained or maxit is reached.

The priors for the regression coefficients in the prevalence and incidence models can be controlled using beta.prior, beta.prior.X, sig.prior.X, and tau.w. Specifically:

- beta.prior determines the prior type for β_{xj} (either normal or Student-t t).
- beta.prior.X specifies either the standard deviation (for normal priors) or degrees of freedom (for Student-t priors). The default is a standard normal prior.
- A half-normal prior is used for σ, with sig.prior.X controlling the standard deviation.
- A zero-centered normal prior is assigned to β_{wj}, with tau.w controlling its standard deviation (default: standard normal).

Sometimes model fitting can be improved by fixing the σ parameter to a value, which is achieved through setting fix.sigma.X = TRUE. Then, the value specified as sig.prior.X is regarded as the correct value for σ, akin to a point prior on this value. The functionality can also be used to obtain the exponential distribution, aking to a Markov model. For this choose dist='weibull', sig.prior.X = 1, and fix.sigma.X=TRUE.

The prev.run argument allows updating a previous run with additional MCMC draws. The MCMC chain resumes from the last draws, continues, and merges with the original run. If an initial model was fit using mod <- bayes.2S(...), it can be updated using mod_update <- bayes.2S(prev.run = mod). By default, ndraws additional iterations are added unless otherwise specified via ndraws.update. When updating, the number of discarded burn-in draws can be adjusted to half of the total draws (update.burnin = TRUE) or remain at the initial number (update.burnin = FALSE).

The Gibbs sampler requires starting values, which are obtained from an initial Bayesian interval-censored survival model using the specified dist distribution. The jumping distribution variance and the number of MCMC draws for this initialization are controlled via ndraws.naive and naive.run.prop.sd.X. The default values typically suffice but may need adjustment if initialization fails (e.g., increasing ndraws.naive or tuning naive.run.prop.sd.X). If starting values are found but still lead to an infinite posterior at initialization, the error "Bad starting values" is returned. Then it usually sufficces to re-run bayes.2S with an increased ndraws.naive value.

Value

A list containing the following elements:

`par.X.all`	An `mcmc.list` of MCMC samples for the incidence and prevalence model parameters.
`par.X.bi`	An `mcmc.list` of MCMC samples for the incidence and prevalence model parameters after burn-in removal.
`X`	A matrix of posterior draws for the latent event times x_i, with one column per chain.
`C`	A matrix of posterior draws for prevalence class membership g_i, with one column per chain.
`ac.X`	A matrix with MCMC draws in rows and chains in columns, where each row indicates whether the Metropolis sampler accepted (1) or rejected (0) a sample.
`ac.X.cur`	Same as `ac.X`, but only for the last update.
`dat`	A data frame containing the last observed interval.
`priors`	A list of prior specifications for the model parameters, including `beta.prior.X` (incidence regression coefficients) and `sig.prior.X` (scale parameter for the AFT model).
`runtime`	The total runtime of the MCMC sampler.

Additionally, most input arguments are returned as part of the output for reference.

References

T. Klausch, B. I. Lissenberg-Witte, and V. M. Coupe (2024). "A Bayesian prevalence-incidence mixture model for screening outcomes with misclassification.", doi:10.48550/arXiv.2412.16065.

J. S. Liu and Y. N. Wu, "Parameter Expansion for Data Augmentation," Journal of the American Statistical Association, vol. 94, no. 448, pp. 1264–1274, 1999, doi:10.2307/2669940.

Examples

```
library(BayesPIM)

# Generate data according to the Klausch et al. (2024) PIM
set.seed(2025)
dat <- gen.dat(kappa = 0.7, n = 1e3, theta = 0.2,
               p = 1, p.discrete = 1,
               beta.X = c(0.2, 0.2), beta.W = c(0.2, 0.2),
               v.min = 20, v.max = 30, mean.rc = 80,
               sigma.X = 0.2, mu.X = 5, dist.X = "weibull",
               prob.r = 1)

# Initial model fit with fixed test sensitivity kappa (approx. 1-3 minutes runtime)
mod <- bayes.2S(Vobs = dat$Vobs,
                Z.X = dat$Z,
                Z.W = dat$Z,
                r = dat$r,
```

```
                kappa = 0.7,
                update.kappa = FALSE,
                ndraws = 1e4,
                chains = 2,
                prop.sd.X = 0.008,
                parallel = TRUE,
                dist.X = "weibull")


# Inspect results
mod$runtime  # Runtime of the Gibbs sampler
plot(trim.mcmc(mod$par.X.all, thining = 10)) # MCMC chains including burn-in, also see ?trim.mcmc
plot(trim.mcmc(mod$par.X.bi, thining = 10))   # MCMC chains excluding burn-in
apply(mod$ac.X, 2, mean)  # Acceptance rates per chain
gelman.diag(mod$par.X.bi)  # Gelman convergence diagnostics

# Model updating
mod_update <- bayes.2S(prev.run = mod)      # Adds ndraws additional MCMC draws
mod_update <- bayes.2S(prev.run = mod,
                       ndraws.update = 1e3) # Adds ndraws.update additional MCMC draws

# Example with kappa estimated/updated
mod2 <- bayes.2S(Vobs = dat$Vobs,
                 Z.X = dat$Z,
                 Z.W = dat$Z,
                 r = dat$r,
                 kappa = 0.7,
                 update.kappa = TRUE,
                 kappa.prior = c(0.7, 0.1), # Beta prior, mean = 0.7, s.d. = 0.1
                 ndraws = 1e4,
                 chains = 2,
                 prop.sd.X = 0.008,
                 parallel = TRUE,
                 dist.X = "weibull")

# Inspect results
mod2$runtime # runtime of Gibbs sampler
plot( trim.mcmc( mod2$par.X.all, thining = 10) ) # kappa returned as part of the mcmc.list
```

`bayes.2S_seq` *bayes.2S_seq: Bayesian Prevalence-Incidence Mixture Model (sequential processing)*

Description

Estimates the Pattern Mixture model of Klausch et al. (2025) using a Bayesian Gibbs sampler. Identical to bayes.2S but uses a `for` loop over MCMC chains instead of `foreach`.

Usage

```
bayes.2S_seq(
  Vobs,
  Z.X = NULL,
  Z.W = NULL,
  r = NULL,
  dist.X = "weibull",
  kappa = 0.5,
  update.kappa = FALSE,
  kappa.prior = NULL,
  ndraws = 1000,
  prop.sd.X = NULL,
  chains = 3,
  thining = 1,
  parallel = TRUE,
  update.till.converge = FALSE,
  maxit = Inf,
  conv.crit = "upper",
  min_effss = chains * 10,
  beta.prior = "norm",
  beta.prior.X = 1,
  sig.prior.X = 1,
  tau.w = 1,
  fix.sigma.X = FALSE,
  prev.run = NULL,
  update.burnin = TRUE,
  ndraws.update = NULL,
  prev = TRUE,
  vanilla = FALSE,
  ndraws.naive = 5000,
  naive.run.prop.sd.X = prop.sd.X,
  par.exp = FALSE,
  collapsed.g = TRUE,
  k.prior = 1,
  fix.k = FALSE
)
```

Arguments

`Vobs`	A list of length n of numeric vectors representing screening times. The first element of each vector should always be `0` and the last element `Inf` in the case of right censoring.
`Z.X`	A numeric matrix of dimension $n \times p_x$ containing covariates for the AFT incidence model. Missing values are not allowed.
`Z.W`	A numeric matrix of dimension $n \times p_w$ containing covariates for the probit prevalence model. Missing values are not allowed.
`r`	A binary vector of length n indicating whether a baseline test was conducted (`1` for yes, `0` for no / missing baseline test).

`dist.X` Character. Specifies the distribution for the time-to-incidence variable; choices are `'weibull'`, `'lognormal'`, or `'loglog'` (log-logistic).

`kappa` Numeric. The test sensitivity value to be used if `update.kappa = FALSE`; otherwise, the starting value for estimating κ.

`update.kappa` Logical. If `TRUE`, the test sensitivity (κ) is updated during the Gibbs sampler.

`kappa.prior` A numeric vector of length 2. When specified, a Beta distribution prior is used for κ, centered at `kappa.prior[1]` with standard deviation `kappa.prior[2]`. If `NULL`, a uniform prior (Beta(1,1)) is used.

`ndraws` Integer. The total number of MCMC draws for the main Gibbs sampler.

`prop.sd.X` Numeric. The standard deviation for the proposal (jumping) distribution in the Metropolis sampler used for updating β_{xj}. Can be searched for automatically using search.prop.sd.

`chains` Integer. The number of MCMC chains to run.

`thining` Integer. The thinning interval for the MCMC sampler.

`parallel` Logical. If `TRUE`, parallel processing is enabled for the Gibbs sampler. Each chain is assigned to one core (via the `foreach` package). Alternatively, use bayes.2S_seq which employs a `for` loop over the chains.

`update.till.converge` Logical. If `TRUE`, the model is updated iteratively until convergence criteria are met. Convergence is assessed using the Gelman-Rubin diagnostic ($R < 1.1$) and a minimum effective sample size (`min_effss`) for each parameter, respectively.

`maxit` Numeric. The maximum number of MCMC draws allowed before interrupting the update process when `update.till.converge` is enabled. Default is `Inf` (i.e., no automatic interruption).

`conv.crit` Character. Specifies whether the convergence check uses the point estimate (`'point'`) or the upper bound (`'upper'`) of the Gelman-Rubin diagnostic estimate ($\hat{R}$).

`min_effss` Integer. The minimum effective sample size required for each parameter before convergence is accepted during iterative updating.

`beta.prior` Character. Specifies the type of prior for the incidence regression coefficients (β_{xj}); options are `'norm'` for normal and `'t'` for student-t.

`beta.prior.X` Numeric. The hyperparameter for the prior distribution of the regression coefficients (β_{xj}) in the AFT incidence model. For a normal prior, this is the standard deviation; for a student-t prior, it represents the degrees of freedom. The default produces a standard-normal prior.

`sig.prior.X` Numeric. The hyperparameter (standard deviation) for a half-normal prior on the scale parameter (σ) of the AFT incidence model.

`tau.w` Numeric. The hyperparameter (standard deviation) for the normal prior distribution of the regression coefficients (β_{wj}) in the probit prevalence model. The default produces a standard-normal prior.

`fix.sigma.X` Logical. If `TRUE`, the scale parameter (σ) in the AFT incidence model is fixed at the value provided in `sig.prior.X`; if `FALSE`, it is updated.

`prev.run` Optional. An object of class `BayesPIM` containing results from a previous run. When provided, data and prior settings are adopted from the previous run, and the MCMC chain continues from the last draw.

`update.burnin` Logical. If `TRUE` (default) and `prev.run` is provided, the burn-in period is updated to half of the total draws (sum of previous and current runs); otherwise, the burn-in is maintained as half of the draws from the initial run.

`ndraws.update` Integer. The number of MCMC draws for updating a previous run or for convergence updates. If unspecified, `ndraws` is used.

`prev` Logical. If `TRUE`, prevalence adjustment is applied; if `FALSE`, prevalence is assumed to be zero.

`vanilla` Logical. If `TRUE`, a vanilla run is performed that assumes no prevalence adjustment and fixes $\kappa = 1$, equivalent to a standard Bayesian interval-censored survival regression.

`ndraws.naive` Integer. The number of MCMC draws for a preliminary vanilla run used to obtain starting values. Increase if initial values lead to issues (e.g., an infinite posterior).

`naive.run.prop.sd.X` Numeric. The standard deviation for the proposal distribution used in the vanilla run. Adjust only if the acceptance rate is significantly off target, as indicated by an interruption message.

`par.exp` Logical. If `TRUE`, the parameter expansion technique of Liu & Wiu (1999) with a Haar prior is employed for updating the regression coefficients (β_{wj}) in the prevalence model. Experimental: tests suggest that it does not improve convergence or reduce autocorrelation.

`collapsed.g` Logical. If `TRUE`, the latent prevalence class membership update in the Gibbs sampler is integrated (collapsed) over the latent incidence time variable. This setting is recommended to improve convergence.

`k.prior` Experimental prior parameter for generalized gamma; currently not used.

`fix.k` Experimental fixing of prior parameter for generalized gamma; currently not used.

Details

This function is dentical to bayes.2S with the only difference being that the `chains` MCMC chains are run in sequence using a `for` loop instead of parallel processing. This can be useful if operating systems do not support `foreach` or for simulation studies that parallize replication of experiments using `foreach` and/or need a worker that does not apply `foreach` internally.

Value

A list containing the following elements:

`par.X.all` An `mcmc.list` of MCMC samples for the incidence and prevalence model parameters.

`par.X.bi` An `mcmc.list` of MCMC samples for the incidence and prevalence model parameters after burn-in removal.

X	A matrix of posterior draws for the latent event times x_i, with one column per chain.
C	A matrix of posterior draws for prevalence class membership g_i, with one column per chain.
ac.X	A matrix with MCMC draws in rows and chains in columns, where each row indicates whether the Metropolis sampler accepted (1) or rejected (0) a sample.
ac.X.cur	Same as ac.X, but only for the last update.
dat	A data frame containing the last observed interval.
priors	A list of prior specifications for the model parameters, including beta.prior.X (incidence regression coefficients) and sig.prior.X (scale parameter for the AFT model).
runtime	The total runtime of the MCMC sampler.

Additionally, most input arguments are returned as part of the output for reference.

References

T. Klausch, B. I. Lissenberg-Witte, and V. M. Coupe (2024). "A Bayesian prevalence-incidence mixture model for screening outcomes with misclassification.", doi:10.48550/arXiv.2412.16065.

J. S. Liu and Y. N. Wu, "Parameter Expansion for Data Augmentation," Journal of the American Statistical Association, vol. 94, no. 448, pp. 1264–1274, 1999, doi: 10.2307/2669940.

Examples

```
library(BayesPIM)

# Generate data according to the Klausch et al. (2025) PIM
set.seed(2025)
dat <- gen.dat(kappa = 0.7, n = 1e3, theta = 0.2,
               p = 1, p.discrete = 1,
               beta.X = c(0.2, 0.2), beta.W = c(0.2, 0.2),
               v.min = 20, v.max = 30, mean.rc = 80,
               sigma.X = 0.2, mu.X = 5, dist.X = "weibull",
               prob.r = 1)

# Initial model fit with fixed test sensitivity kappa (approx. 4-12 minutes runtime)
mod <- bayes.2S_seq(Vobs = dat$Vobs,
                Z.X = dat$Z,
                Z.W = dat$Z,
                r = dat$r,
                kappa = 0.7,
                update.kappa = FALSE,
                ndraws = 1e4,
                chains = 4,
                prop.sd.X = 0.008,
                parallel = TRUE,
                dist.X = "weibull")

# Inspect results
```

```
mod$runtime  # Runtime of the Gibbs sampler
plot(trim.mcmc(mod$par.X.all, thining = 10)) # MCMC chains including burn-in, also see ?trim.mcmc
plot(trim.mcmc(mod$par.X.bi, thining = 10))   # MCMC chains excluding burn-in
apply(mod$ac.X, 2, mean)  # Acceptance rates per chain
gelman.diag(mod$par.X.bi)  # Gelman convergence diagnostics

# Model updating
mod_update <- bayes.2S(prev.run = mod)      # Adds ndraws additional MCMC draws
mod_update <- bayes.2S(prev.run = mod,
                       ndraws.update = 1e3) # Adds ndraws.update additional MCMC draws

# Example with kappa estimated/updated
mod2 <- bayes.2S_seq(Vobs = dat$Vobs,
                 Z.X = dat$Z,
                 Z.W = dat$Z,
                 r = dat$r,
                 kappa = 0.7,
                 update.kappa = TRUE,
                 kappa.prior = c(0.7, 0.1), # Beta prior, mean = 0.7, s.d. = 0.1
                 ndraws = 1e4,
                 chains = 4,
                 prop.sd.X = 0.008,
                 parallel = TRUE,
                 dist.X = "weibull")

# Inspect results
mod2$runtime # runtime of Gibbs sampler
plot( trim.mcmc( mod2$par.X.all, thining = 10) ) # kappa returned as part of the mcmc.list
```

`gen.dat` *gen.dat: Simulate Screening Data for a Prevalence-Incidence Mixture Model*

Description

Generates synthetic data according to the Bayesian prevalence-incidence mixture (PIM) framework of Klausch et al. (2025) with interval-censored screening outcomes. The function simulates continuous or discrete baseline covariates, event times from one of several parametric families, and irregular screening schedules, yielding interval-censored observations suitable for testing or demonstrating PIM-based or other interval-censored survival methods.

Usage

```
gen.dat(
  kappa = 0.7,
  n = 1000,
  p = 2,
  p.discrete = 0,
```

```
  r = 0,
  s = 1,
  sigma.X = 1/2,
  mu.X = 4,
  beta.X = NULL,
  beta.W = NULL,
  theta = 0.15,
  v.min = 1,
  v.max = 6,
  mean.rc = 40,
  dist.X = "weibull",
  k = 1,
  sel.mod = "probit",
  prob.r = 0
)
```

Arguments

kappa	Numeric. Test sensitivity parameter κ used when generating misclassification. A value of 1 implies perfect sensitivity.
n	Integer. Sample size.
p	Integer. Number of continuous baseline covariates to simulate.
p.discrete	Integer. If 1, include an additional discrete covariate Z_{discrete} from $\text{Bernoulli}(0.5)$; otherwise, none.
r	Numeric. Correlation coefficient(s) used to build the covariance matrix of continuous covariates. If p > 1, off-diagonal entries of the correlation matrix are set to r.
s	Numeric. Standard deviation(s) of the continuous covariates. If p > 1, all continuous covariates share the same s.
sigma.X	Numeric. Scale parameter σ_X in the AFT model for $\log(x_i)$.
mu.X	Numeric. Intercept β_{x0} in the AFT model. In the linear predictor, it appears as $\log(x_i) = \beta_{x0} + \beta_x^\top Z_i + \sigma_X \epsilon_i$. Practically, mu.X is prepended to beta.X when forming the full parameter vector.
beta.X	Numeric vector. The coefficients β_x for the AFT model. Combined with mu.X, the log-scale model is cbind(1, Z_i) %*% c(mu.X, beta.X).
beta.W	Numeric vector. The coefficients β_w for the prevalence model. The intercept β_{w0} is derived from theta.
theta	Numeric. Baseline prevalence parameter on the probability scale. Under: • sel.mod = "probit": $\beta_{w0} = \text{qnorm}(\theta)$. • sel.mod = "logit": $\beta_{w0} = \log(\theta/(1-\theta))$.
v.min	Numeric. Minimum spacing for irregular screening intervals.
v.max	Numeric. Maximum spacing for irregular screening intervals.
mean.rc	Numeric. Mean of the exponential distribution controlling a random right-censoring time t_{rc} after the first screening.

dist.X	Character. Distribution for survival times x_i: "weibull", "lognormal", "loglog" (log-logistic), or "gengamma" (generalized gamma).
k	Numeric. Shape parameter for "gengamma" only.
sel.mod	Character. Either "probit" or "logit", specifying the link function for the prevalence model.
prob.r	Numeric. Probability that a baseline test is performed ($r_i = 1$). If prob.r = 0, no baseline tests are done.

Details

The data-generating process includes:

1. **Covariates Z:** Continuous covariates are simulated using a correlation structure specified by r and a common standard deviation s. If p.discrete = 1, a single discrete covariate is added, drawn from $\mathrm{Bernoulli}(0.5)$.
2. **Event Times X:** An Accelerated Failure Time (AFT) model is used:

$$\log(x_i) = \beta_{x0} + \beta_x^\top z_{xi} + \sigma_X \, \epsilon_i,$$

   where β_{x0} is the intercept (set by mu.X) and β_x are the other regression coefficients (provided via beta.X). The error term ϵ_i is drawn from the distribution chosen by dist.X: "weibull", "lognormal", "loglog" (log-logistic), or "gengamma" (generalized gamma). For "gengamma", the shape parameter k is additionally used.
3. **Irregular Screening Schedules V_i:** Each individual has multiple screening times generated randomly between v.min and v.max, ending in right censoring or the time of detection. These screening times (including a 0 for baseline and Inf for censoring) are returned in Vobs.
4. **Prevalence Indicator g_i:** Baseline prevalence is modeled via either a probit or logit link, consistent with:

$$w_i = \beta_{w0} + \beta_w^\top z_{wi} + \psi_i,$$

   where β_{w0} is determined by theta, and β_w by beta.W. Specifically:
   - If sel.mod = "probit", then $\beta_{w0} = \mathrm{qnorm}(\theta)$.
   - If sel.mod = "logit", then $\beta_{w0} = \log(\theta/(1-\theta))$.

   We set $g_i = 1$ if $w_i > 0$, and $g_i = 0$ otherwise.
5. **Baseline Test Missingness r_i:** A baseline test indicator $r_i \in \{0, 1\}$ is generated via $\mathrm{Bernoulli}$(prob.r), so $r_i = 1$ means the baseline test is performed and $r_i = 0$ means it is missing.
6. **Test Sensitivity κ:** A misclassification parameter κ (test sensitivity) can be specified via kappa. If $\kappa < 1$, some truly positive cases are missed.

Value

A list with the following elements:

Vobs A list of length n, each entry containing screening times. The first element is 0 (baseline), and Inf may indicate right censoring.

X.true Numeric vector of length n giving the true (latent) event times x_i.

`Z` Numeric matrix of dimension $n \times p$ (plus an extra column if `p.discrete = 1`) containing the covariates.

`C` Binary vector of length `n`, indicating whether an individual is truly positive at baseline ($g_i = 1$).

`r` Binary vector of length `n`, indicating whether the baseline test was performed ($r_i = 1$) or missing ($r_i = 0$).

`p.W` Numeric vector of length `n` giving the true prevalence probabilities, $P(g_i = 1)$.

References

T. Klausch, B. I. Lissenberg-Witte, and V. M. Coupé, "A Bayesian prevalence-incidence mixture model for screening outcomes with misclassification," arXiv:2412.16065.

Examples

```
# Generate a small dataset for testing
set.seed(2025)
sim_data <- gen.dat(n = 100, p = 1, p.discrete = 1,
                    sigma.X = 0.5, mu.X = 2,
                    beta.X = c(0.2, 0.2), beta.W = c(0.5, -0.2),
                    theta = 0.2,
                    dist.X = "weibull", sel.mod = "probit")
str(sim_data)
```

---

get.IC_2S *Compute Information Criteria for a Bayesian Prevalence-Incidence Mixture Model*

Description

Computes and returns information criteria for a fitted Bayesian prevalence-incidence mixture model, including the Widely Applicable Information Criterion 1 (WAIC-1), WAIC-2, and the Deviance Information Criterion (DIC). These criteria are commonly used for model comparison and evaluation in Bayesian analysis. See Gelman et al. (2014) for further details on these criteria.

Usage

```
get.IC_2S(mod, samples = nrow(mod$par.X.bi[[1]]), cores = NULL)
```

Arguments

`mod`	A fitted prevalence-incidence mixture model of class `bayes.2S`.
`samples`	The number of MCMC samples to use. Maximum is the number of post-burn-in samples available in the `bayes.2S` object.
`cores`	The number of cores for parallel processing using `foreach`. If `NULL` (default), all available cores will be used.

Details

This function calculates information criteria for a fitted Bayesian prevalence-incidence mixture model (`bayes.2S`). The information criteria include:

- **WAIC-1**: Based on the sum of posterior variances of log-likelihood contributions.
- **WAIC-2**: Similar to WAIC-1 but incorporates an alternative variance estimate.
- **DIC**: Measures model fit by penalizing complexity via the effective number of parameters.

The computation is performed by evaluating log-likelihood values for MCMC samples. By default, all MCMC samples after burn-in are used, though a subset can be specified via the `samples` argument.

Parallelization is available via the `foreach` package, utilizing multiple cores if `cores` is set accordingly. If `cores = NULL`, all available cores will be used.

Value

A `matrix` containing WAIC-1, WAIC-2, and DIC values for the model.

References

Gelman, A., Hwang, J., & Vehtari, A. (2014). Understanding predictive information criteria for Bayesian models. Stat Comput, 24(6), 997–1016.

Examples

```
# Generate data according to the Klausch et al. (2024) PIM
set.seed(2025)
dat = gen.dat(kappa = 0.7, n= 1e3, theta = 0.2,
              p = 1, p.discrete = 1,
              beta.X = c(0.2,0.2), beta.W = c(0.2,0.2),
              v.min = 20, v.max = 30, mean.rc = 80,
              sigma.X = 0.2, mu.X=5, dist.X = "weibull",
              prob.r  = 1)

# An initial model fit with fixed test sensitivity kappa (approx. 1-3 minutes, depending on machine)
mod = bayes.2S( Vobs = dat$Vobs,
                Z.X = dat$Z,
                Z.W = dat$Z,
                r= dat$r,
                kappa = 0.7,
                update.kappa = FALSE,
                ndraws= 1e4,
                chains = 2,
                prop.sd.X = 0.008,
                parallel = TRUE,
                dist.X = 'weibull'
)

# Get information criteria
get.IC_2S(mod, samples = 1e3, cores = 2)
```

`get.ppd.2S` *Posterior Predictive Cumulative Incidence Function*

Description

Computes the posterior predictive cumulative incidence function (CIF) from a bayes.2S prevalence-incidence mixture model. The function can return *quantiles* corresponding to user-specified *percentiles* (i.e., time points at which the cumulative probability reaches certain thresholds), or vice versa (*percentiles* at user-specified *quantiles*). Additionally, it allows for marginal or conditional CIFs of either the mixture population (including prevalence as a point-mass at time zero) or the non-prevalent (healthy) subpopulation.

Usage

```
get.ppd.2S(
  mod,
  fix_Z.X = NULL,
  fix_Z.W = NULL,
  pst.samples = 1000,
  perc = seq(0, 1, 0.01),
  type = "x",
  ppd.type = "percentiles",
  quant = NULL
)
```

Arguments

`mod`	A fitted prevalence-incidence mixture model of class `bayes.2S`.
`fix_Z.X`	Either `NULL` for a marginal CIF or a numeric vector of length `ncol(Z.X)` to request a conditional CIF. Numeric entries fix those covariates at the given value, whereas `NA` entries are integrated out. See *Details*.
`fix_Z.W`	Same as `fix_Z.X` but for the prevalence model covariates; must be `NULL` to obtain a marginal CIF.
`pst.samples`	Integer; number of posterior samples to draw when computing the posterior predictive CIF. Must not exceed the total available posterior samples in `mod`. Larger values can improve precision but increase computation time.
`perc`	A numeric vector of cumulative probabilities (i.e., percentiles in (0,1)) for which time points are returned when `ppd.type = "quantiles"`.
`type`	Character; `"xstar"` for the mixture CIF (prevalence + incidence), or `"x"` for the non-prevalent (healthy) population CIF.
`ppd.type`	Character; `"percentiles"` to return cumulative probabilities at times `quant`, or `"quantiles"` to return time points at cumulative probabilities `perc`.
`quant`	A numeric vector of time points for which the function returns cumulative probabilities when `ppd.type = "percentiles"`.

Details

For a prevalence-incidence mixture model, some fraction of the population may already have experienced the event (prevalent cases) at baseline, while the remaining (healthy) fraction has not. This function estimates the CIF in two ways:

- `type = "xstar"` (mixture CIF): Includes a point-mass at time zero representing baseline prevalence, with incidence beginning thereafter.
- `type = "x"` (non-prevalent CIF): Excludes prevalent cases, so it only shows incidence among the initially healthy subpopulation.

You may request a *marginal* CIF by setting both `fix_Z.X = NULL` and `fix_Z.W = NULL`, thus integrating over all covariates. Alternatively, a *conditional* CIF can be obtained by partially or fully specifying fixed covariate values in `fix_Z.X` (and optionally `fix_Z.W`) while integrating out the unspecified covariates (`NA` entries).

The function operates in two main modes:

- `ppd.type = "quantiles"`: Given a set of `perc` (cumulative probabilities), returns corresponding *quantiles* (time points).
- `ppd.type = "percentiles"`: Given a set of `quant` (time points), returns corresponding *percentiles* (cumulative probabilities).

Value

A `list` with some or all of the following elements:

`med.cdf` • If `ppd.type = "quantiles"`, `med.cdf` contains the median quantiles (time points) across posterior samples for each percentile in `perc`.
• If `ppd.type = "percentiles"`, `med.cdf` contains the median cumulative probabilities across posterior samples for each time point in `quant`.

`med.cdf.ci` A 2-row matrix with the 2.5% and 97.5% posterior quantiles for the estimated `med.cdf`, reflecting uncertainty.

`quant` If `ppd.type = "percentiles"`, this is a copy of the input `quant` (time points).

`perc` If `ppd.type = "quantiles"`, this is a copy of the input `perc` (cumulative probabilities).

Examples

```
# Generate data according to the Klausch et al. (2024) PIM
set.seed(2025)
dat <- gen.dat(kappa = 0.7, n = 1e3, theta = 0.2,
               p = 1, p.discrete = 1,
               beta.X = c(0.2, 0.2), beta.W = c(0.2, 0.2),
               v.min = 20, v.max = 30, mean.rc = 80,
               sigma.X = 0.2, mu.X = 5, dist.X = "weibull",
               prob.r  = 1)

# Fit a bayes.2S model (example with moderate ndraws = 2e4)
mod <- bayes.2S(Vobs = dat$Vobs, Z.X = dat$Z, Z.W = dat$Z, r = dat$r,
                kappa = 0.7, update.kappa = FALSE, ndraws = 1e4,
                chains = 2, prop.sd.X = 0.008, parallel = TRUE,
```

```
                dist.X = "weibull")

###################
# (1) Provide percentiles, get back quantiles (times)
###################
cif_nonprev <- get.ppd.2S(mod, pst.samples = 1e3, type = "x",
                          ppd.type = "quantiles", perc = seq(0, 1, 0.01))
cif_mix     <- get.ppd.2S(mod, pst.samples = 1e3, type = "xstar",
                          ppd.type = "quantiles", perc = seq(0, 1, 0.01))

# Plot: Non-prevalent stratum CIF vs. mixture CIF (marginal)
oldpar <- par(no.readonly = TRUE)
par(mfrow = c(1,2))

plot(cif_nonprev$med.cdf, cif_nonprev$perc, type = "l", xlim = c(0,300), ylim = c(0,1),
     xlab = "Time", ylab = "Cumulative Incidence")
lines(cif_nonprev$med.cdf.ci[1,], cif_nonprev$perc, lty = 2)
lines(cif_nonprev$med.cdf.ci[2,], cif_nonprev$perc, lty = 2)

plot(cif_mix$med.cdf, cif_mix$perc, type = "l", xlim = c(0,300), ylim = c(0,1),
     xlab = "Time", ylab = "Cumulative Incidence")
lines(cif_mix$med.cdf.ci[1,], cif_mix$perc, lty = 2)
lines(cif_mix$med.cdf.ci[2,], cif_mix$perc, lty = 2)

###################
# (2) Provide quantiles (times), get back percentiles (cumulative probabilities)
###################
cif2_nonprev <- get.ppd.2S(mod, pst.samples = 1e3, type = "x",
                           ppd.type = "percentiles", quant = 1:300)
cif2_mix     <- get.ppd.2S(mod, pst.samples = 1e3, type = "xstar",
                           ppd.type = "percentiles", quant = 1:300)

# Plot: Non-prevalent vs. mixture CIF using times on the x-axis
plot(cif2_nonprev$quant, cif2_nonprev$med.cdf, type = "l", xlim = c(0,300), ylim = c(0,1),
     xlab = "Time", ylab = "Cumulative Incidence")
lines(cif2_nonprev$quant, cif2_nonprev$med.cdf.ci[1,], lty = 2)
lines(cif2_nonprev$quant, cif2_nonprev$med.cdf.ci[2,], lty = 2)

plot(cif2_mix$quant, cif2_mix$med.cdf, type = "l", xlim = c(0,300), ylim = c(0,1),
     xlab = "Time", ylab = "Cumulative Incidence")
lines(cif2_mix$quant, cif2_mix$med.cdf.ci[1,], lty = 2)
lines(cif2_mix$quant, cif2_mix$med.cdf.ci[2,], lty = 2)

###################
# (3) Conditional CIFs by fixing some covariates
###################
cif_mix_m1 <- get.ppd.2S(mod, fix_Z.X = c(-1, NA), pst.samples = 1e3,
                         type = "xstar", ppd.type = "quantiles", perc = seq(0,1,0.01))
cif_mix_0  <- get.ppd.2S(mod, fix_Z.X = c(0, NA),  pst.samples = 1e3,
                         type = "xstar", ppd.type = "quantiles", perc = seq(0,1,0.01))
cif_mix_p1 <- get.ppd.2S(mod, fix_Z.X = c(1, NA),  pst.samples = 1e3,
                         type = "xstar", ppd.type = "quantiles", perc = seq(0,1,0.01))
```

```
# Plot: mixture CIF for three different values of the first covariate
par(mfrow = c(1,1))
plot(cif_mix_m1$med.cdf, cif_mix_m1$perc, type = "l", xlim = c(0,300), ylim = c(0,1),
     xlab = "Time", ylab = "Cumulative Incidence", col=1)
lines(cif_mix_0$med.cdf,  cif_mix_m1$perc, col=2)
lines(cif_mix_p1$med.cdf, cif_mix_m1$perc, col=3)

par(oldpar)
```

`search.prop.sd` *Automated Heuristic Search of a Proposal Standard Deviation for* `bayes.2S`

Description

The bayes.2S Gibbs sampler uses a Metropolis step for sampling the incidence model parameters and requires specifying a standard deviation for the normal proposal (jumping) distribution. This function uses a heuristic algorithm to find a proposal distribution standard deviation such that the Metropolis sampler accepts proposed draws at an acceptance rate within the user-defined interval (by default around 20–25%).

Usage

```
search.prop.sd(m, ndraws = 1000, succ.min = 3, acc.bounds.X = c(0.2, 0.25))
```

Arguments

`m`	A model object of class `bayes.2S`.
`ndraws`	Starting number of MCMC iterations after which the acceptance rate is first evaluated. Defaults to 1000.
`succ.min`	The algorithm doubles the number of MCMC draws `succ.min` times (each time the acceptance rate is within `acc.bounds.X`), ensuring stability. Defaults to 3.
`acc.bounds.X`	A numeric vector of length two specifying the lower and upper bounds for the acceptable acceptance rate. Defaults to `c(0.2, 0.25)`.

Details

Starting from an initial `bayes.2S` model object `m`, the function attempts to calibrate the standard deviation of the proposal distribution. Specifically, it:

1. Runs an initial update of `ndraws` iterations and computes an acceptance rate.
2. If the acceptance rate lies within `acc.bounds.X`, the number of MCMC draws `ndraws` is doubled, and the process repeats.
3. Otherwise, the proposal standard deviation σ is adjusted based on whether the acceptance rate p is below the lower bound a or above the upper bound b of `acc.bounds.X`.

4. The formula for adjustment is:

$$\sigma \leftarrow \sigma \times (1 - \frac{(a-p)}{a}) \quad \text{if } p < a, \quad \sigma \leftarrow \sigma \times (1 + \frac{(p-b)}{b}) \quad \text{if } p > b.$$

By default, if the acceptance rate falls within $[0.2, 0.25]$, that σ is considered acceptable, and the process continues until `succ.min` consecutive successes (doubles) are achieved.

Value

A `list` with the following elements:

`prop.sd.X` The final (adjusted) proposal standard deviation.

`ac.X` The acceptance rate in the last iteration.

Examples

```
# Generate data according to Klausch et al. (2025) PIM
set.seed(2025)
dat = gen.dat(kappa = 0.7, n = 1e3, theta = 0.2,
              p = 1, p.discrete = 1,
              beta.X = c(0.2, 0.2), beta.W = c(0.2, 0.2),
              v.min = 20, v.max = 30, mean.rc = 80,
              sigma.X = 0.2, mu.X = 5, dist.X = "weibull",
              prob.r  = 1)

# An initial model fit with a moderate number of ndraws (e.g., 1e3)
mod = bayes.2S(
  Vobs = dat$Vobs, Z.X = dat$Z, Z.W = dat$Z, r = dat$r,
  kappa = 0.7, update.kappa = FALSE, ndraws = 1e3, chains = 2,
  prop.sd.X = 0.005, parallel = TRUE, dist.X = "weibull"
)

# Running the automated search
search.sd <- search.prop.sd(m = mod)
print(search.sd)
```

---

search.prop.sd_seq *Automated Heuristic Search of a Proposal Standard Deviation for* bayes.2S *(sequential processing)*

Description

The bayes.2S Gibbs sampler uses a Metropolis step for sampling the incidence model parameters and requires specifying a standard deviation for the normal proposal (jumping) distribution. This function uses a heuristic algorithm to find a proposal distribution standard deviation such that the Metropolis sampler accepts proposed draws at an acceptance rate within the user-defined interval (by default around 20–25%). This is the sequential processing analogue to search.prop.sd which does parallel processing by default.

Usage

```
search.prop.sd_seq(m, ndraws = 1000, succ.min = 3, acc.bounds.X = c(0.2, 0.25))
```

Arguments

m	A model object of class bayes.2S.
ndraws	Starting number of MCMC iterations after which the acceptance rate is first evaluated. Defaults to 1000.
succ.min	The algorithm doubles the number of MCMC draws succ.min times (each time the acceptance rate is within acc.bounds.X), ensuring stability. Defaults to 3.
acc.bounds.X	A numeric vector of length two specifying the lower and upper bounds for the acceptable acceptance rate. Defaults to c(0.2, 0.25).

Details

Starting from an initial bayes.2S model object m, the function attempts to calibrate the standard deviation of the proposal distribution. Specifically, it:

1. Runs an initial update of ndraws iterations and computes an acceptance rate.
2. If the acceptance rate lies within acc.bounds.X, the number of MCMC draws ndraws is doubled, and the process repeats.
3. Otherwise, the proposal standard deviation σ is adjusted based on whether the acceptance rate p is below the lower bound a or above the upper bound b of acc.bounds.X.
4. The formula for adjustment is:

$$\sigma \leftarrow \sigma \times (1 - \frac{(a-p)}{a}) \quad \text{if } p < a, \quad \sigma \leftarrow \sigma \times (1 + \frac{(p-b)}{b}) \quad \text{if } p > b.$$

By default, if the acceptance rate falls within $[0.2, 0.25]$, that σ is considered acceptable, and the process continues until succ.min consecutive successes (doubles) are achieved.

Value

A list with the following elements:

prop.sd.X The final (adjusted) proposal standard deviation.

ac.X The acceptance rate in the last iteration.

Examples

```
# Generate data according to Klausch et al. (2025) PIM
set.seed(2025)
dat = gen.dat(kappa = 0.7, n = 1e3, theta = 0.2,
              p = 1, p.discrete = 1,
              beta.X = c(0.2, 0.2), beta.W = c(0.2, 0.2),
              v.min = 20, v.max = 30, mean.rc = 80,
              sigma.X = 0.2, mu.X = 5, dist.X = "weibull",
              prob.r  = 1)
```

```
# An initial model fit with a moderate number of ndraws (e.g., 1e3)
mod = bayes.2S(
  Vobs = dat$Vobs, Z.X = dat$Z, Z.W = dat$Z, r = dat$r,
  kappa = 0.7, update.kappa = FALSE, ndraws = 1e3, chains = 2,
  prop.sd.X = 0.005, parallel = TRUE, dist.X = "weibull"
)

# Running the automated search
search.sd <- search.prop.sd_seq(m = mod)
print(search.sd)
```

`trim.mcmc` *Subset MCMC draws (burn-in and thinning)*

Description

Takes an `mcmc.list` object (or a list of MCMC chains) and returns a new `mcmc.list` containing only the specified subset of iterations (from `burnin` to `end`) with the specified thinning interval.

Usage

```
trim.mcmc(obj, burnin = 1, end = nrow(as.matrix(obj[[1]])), thining = 1)
```

Arguments

`obj`	An object of class `mcmc.list` (or a list of matrices) containing MCMC draws.
`burnin`	A numeric scalar giving the starting iteration of the MCMC sample to keep. Defaults to `1`.
`end`	A numeric scalar giving the last iteration of the MCMC sample to keep. Defaults to the number of rows in the first chain of `obj`.
`thining`	A numeric scalar for the thinning interval. Defaults to `1`.

Details

This function subsets each chain of the input `obj` to the specified iteration indices and creates a new `mcmc.list`. If you have multiple MCMC chains, each chain is trimmed in the same way.

Value

An object of class `mcmc.list`, representing the trimmed subset of the original MCMC draws.

Examples

```
# Example with a toy mcmc.list
set.seed(123)
x1 <- matrix(rnorm(2000), ncol = 2)
x2 <- matrix(rnorm(2000), ncol = 2)
mcmc_list <- mcmc.list(mcmc(x1), mcmc(x2))

# Trim and thin the chains
trimmed_mcmc <- trim.mcmc(mcmc_list, burnin = 100, end = 800, thining = 5)
summary(trimmed_mcmc)
```

Index