# Package ‘bayesSSM’

April 23, 2025

**Type** Package

**Title** Bayesian Methods for State Space Models

**Version** 0.4.7

**Description** Implements methods for Bayesian analysis of State Space Models. Includes implementations the Particle Marginal Metropolis-Hastings algorithm described in Andrieu et al. (2010) <doi:10.1111/j.1467-9868.2009.00736.x> and automatic tuning inspired by Pitt et al. (2012) <doi:10.1016/j.jeconom.2012.06.004> and J. Dahlin and T. B. Schön (2019) <doi:10.18637/jss.v088.c02>.

**License** MIT + file LICENSE

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Imports** MASS, stats, dplyr, future, future.apply

**Suggests** knitr, rmarkdown, testthat (>= 3.0.0), ggplot2, tidyr, extraDistr

**Config/testthat/edition** 3

**URL** `https://github.com/BjarkeHautop/bayesSSM`, `https://bjarkehautop.github.io/bayesSSM/`

**BugReports** `https://github.com/BjarkeHautop/bayesSSM/issues`

**VignetteBuilder** knitr

**Config/Needs/website** rmarkdown

**NeedsCompilation** no

**Author** Bjarke Hautop [aut, cre, cph]

**Maintainer** Bjarke Hautop <`bjarke.hautop@gmail.com`>

**Repository** CRAN

**Date/Publication** 2025-04-23 10:30:02 UTC

# Contents

---

## `default_tune_control` *Create Tuning Control Parameters*

---

### Description

This function creates a list of tuning parameters used by the `pmmh` function. The tuning choices are inspired by Pitt et al. [2012] and Dahlin and Schön [2019].

### Usage

```
default_tune_control(
  pilot_proposal_sd = 0.5,
  pilot_n = 100,
  pilot_m = 2000,
  pilot_target_var = 1,
  pilot_burn_in = 500,
  pilot_reps = 100,
  pilot_algorithm = c("SISAR", "SISR", "SIS"),
  pilot_resample_fn = c("stratified", "systematic", "multinomial")
)
```

### Arguments

| | |
|---|---|
| `pilot_proposal_sd` | Standard deviation for pilot proposals. Default is 0.5. |
| `pilot_n` | Number of pilot particles for particle filter. Default is 100. |
| `pilot_m` | Number of iterations for MCMC. Default is 2000. |
| `pilot_target_var` | The target variance for the posterior log-likelihood evaluated at estimated posterior mean. Default is 1. |
| `pilot_burn_in` | Number of burn-in iterations for MCMC. Default is 500. |
| `pilot_reps` | Number of times a particle filter is run. Default is 100. |
| `pilot_algorithm` | The algorithm used for the pilot particle filter. Default is "SISAR". |
| `pilot_resample_fn` | The resampling function used for the pilot particle filter. Default is "stratified". |

## Value

A list of tuning control parameters.

## References

M. K. Pitt, R. d. S. Silva, P. Giordani, and R. Kohn. On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. Journal of Econometrics, 171(2):134–151, 2012. doi: https://doi.org/10.1016/j.jeconom.2012.06.004

J. Dahlin and T. B. Schön. Getting started with particle Metropolis-Hastings for inference in nonlinear dynamical models. Journal of Statistical Software, 88(2):1–41, 2019. doi: 10.18637/jss.v088.c02

---

# `ess` *Estimate effective sample size (ESS) of MCMC chains.*

## Description

Estimate effective sample size (ESS) of MCMC chains.

## Usage

```
ess(chains)
```

## Arguments

| | |
|---|---|
| `chains` | A matrix (iterations x chains) or a data.frame with a 'chain' column and parameter columns. |

## Details

Uses the formula for ESS proposed by Vehtari et al. (2021).

## Value

The estimated effective sample size (ess) if given a matrix, or a named vector of ESS values if given a data frame.

## References

Vehtari et al. (2021). Rank-normalization, folding, and localization: An improved R-hat for assessing convergence of MCMC. Available at: https://doi.org/10.1214/20-BA1221

### Examples

```
# With a matrix:
chains <- matrix(rnorm(3000), nrow = 1000, ncol = 3)
ess(chains)

# With a data frame:
chains_df <- data.frame(
  chain = rep(1:3, each = 1000),
  param1 = rnorm(3000),
  param2 = rnorm(3000)
)
ess(chains_df)
```

---

## `particle_filter` *Particle Filter*

### Description

This function implements a bootstrap particle filter for estimating the hidden states in a state space model using sequential Monte Carlo methods. Three filtering variants are supported:

1. **SIS:** Sequential Importance Sampling (without resampling).
2. **SISR:** Sequential Importance Sampling with resampling at every time step.
3. **SISAR:** SIS with adaptive resampling based on the Effective Sample Size (ESS). Resampling is triggered when the ESS falls below a given threshold (default `particles / 2`).

It is recommended to use either SISR or SISAR to avoid weight degeneracy.

### Usage

```
particle_filter(
  y,
  num_particles,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  obs_times = NULL,
  algorithm = c("SISAR", "SISR", "SIS"),
  resample_fn = c("stratified", "systematic", "multinomial"),
  threshold = NULL,
  return_particles = TRUE,
  ...
)
```

### Arguments

| | |
|---|---|
| `y` | A numeric vector or matrix of observations. Each row represents an observation at a time step. |
| `num_particles` | A positive integer specifying the number of particles. |
| `init_fn` | A function that initializes the particle states. It should take the current particles as its first argument and return a vector or matrix of initial particle states. |
| `transition_fn` | A function describing the state transition model. It should take the current particles and the current time step as arguments and return the propagated particles. |
| `log_likelihood_fn` | A function that computes the log likelihoods for the particles. It should accept an observation, the current particles, and the current time step as arguments and return a numeric vector of log likelihood values. |
| `obs_times` | A numeric vector indicating the time points at which observations in `y` are available. Must be of the same length as the number of rows in `y`. If not specified, it is assumed that observations are available at consecutive time steps, i.e., `obs_times = 1:nrow(y)`. |
| `algorithm` | A character string specifying the particle filtering algorithm to use. Must be one of `"SISAR"`, `"SISR"`, or `"SIS"`. Defaults to `"SISAR"`. |
| `resample_fn` | A character string specifying the resampling method. Must be one of `"stratified"`, `"systematic"`, or `"multinomial"`. Defaults to `"stratified"`. |
| `threshold` | A numeric value specifying the ESS threshold for triggering resampling in the `"SISAR"` algorithm. If not provided, it defaults to `particles / 2`. |
| `return_particles` | A logical value indicating whether to return the full particle history. Defaults to `TRUE`. |
| `...` | Additional arguments passed to `init_fn`, `transition_fn`, and `log_likelihood_fn`. |

### Details

The particle filter is a sequential Monte Carlo method that approximates the posterior distribution of the state in a state space model. The three supported algorithms differ in their approach to resampling:

1. **SIS:** Particles are propagated and weighted without any resampling, which may lead to weight degeneracy over time.
2. **SISR:** Resampling is performed at every time step to combat weight degeneracy.
3. **SISAR:** Resampling is performed adaptively; particles are resampled only when the Effective Sample Size (ESS) falls below a specified threshold (defaulting to `particles / 2`).

The Effective Sample Size (ESS) in context of particle filters is defined as

$$ESS = \left(\sum_{i=1}^{\text{n}} w_i^2\right)^{-1},$$

where $n$ is the number of particles and $w_i$ are the normalized weights of the particles.

The default resampling method is stratified resampling, as Douc et al., 2005 showed that it always gives a lower variance compared to multinomial resampling.

## Value

A list containing:

**state_est** A numeric vector of estimated states over time, computed as the weighted average of particles.

**ess** A numeric vector of the Effective Sample Size (ESS) at each time step.

**loglike** The accumulated log-likelihood of the observations given the model.

**loglike_history** A numeric vector of the log-likelihood at each time step.

**algorithm** A character string indicating the filtering algorithm used.

**particles_history** (Optional) A list of particle state matrices over time (one per time step), returned if `return_particles` is `TRUE`.

**weights_history** (Optional) A list of particle weight vectors over time (one per time step), returned if `return_particles` is `TRUE`.

## References

Douc, R., Cappé, O., & Moulines, E. (2005). Comparison of Resampling Schemes for Particle Filtering. Accessible at: https://arxiv.org/abs/cs/0507025

## Examples

```
init_fn <- function(particles) rnorm(particles, 0, 1)
transition_fn <- function(particles) particles + rnorm(length(particles))
log_likelihood_fn <- function(y, particles) {
  dnorm(y, mean = particles, sd = 1, log = TRUE)
}

# Generate data
y <- cumsum(rnorm(50))
num_particles <- 100

# Run the particle filter using default settings.
result <- particle_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn
)
plot(result$state_est, type = "l", col = "blue", main = "State Estimates")

# With parameters
init_fn <- function(particles) rnorm(particles, 0, 1)
transition_fn <- function(particles, mu) {
  particles + rnorm(length(particles), mean = mu)
}
log_likelihood_fn <- function(y, particles, sigma) {
  dnorm(y, mean = particles, sd = sigma, log = TRUE)
}
```

```
# Generate data
y <- cumsum(rnorm(50))
num_particles <- 100

# Run the particle filter using default settings.
result <- particle_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  mu = 1,
  sigma = 1
)
plot(result$state_est, type = "l", col = "blue", main = "State Estimates")

# With observations gaps
init_fn <- function(particles) rnorm(particles, 0, 1)
transition_fn <- function(particles, mu) {
  particles + rnorm(length(particles), mean = mu)
}
log_likelihood_fn <- function(y, particles, sigma) {
  dnorm(y, mean = particles, sd = sigma, log = TRUE)
}

# Generate data using DGP
simulate_ssm <- function(num_steps, mu, sigma) {
  x <- numeric(num_steps)
  y <- numeric(num_steps)
  x[1] <- rnorm(1, mean = 0, sd = sigma)
  y[1] <- rnorm(1, mean = x[1], sd = sigma)
  for (t in 2:num_steps) {
    x[t] <- mu * x[t - 1] + sin(x[t - 1]) + rnorm(1, mean = 0, sd = sigma)
    y[t] <- x[t] + rnorm(1, mean = 0, sd = sigma)
  }
  y
}

data <- simulate_ssm(10, mu = 1, sigma = 1)
# Suppose we have data for t=1,2,3,5,6,7,8,9,10 (i.e., missing at t=4)

obs_times <- c(1, 2, 3, 5, 6, 7, 8, 9, 10)
data <- data[obs_times]

num_particles <- 100
# Run the particle filter
# Specify observation times in the particle filter using obs_times
result <- particle_filter(
  y = data,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
```

```
  log_likelihood_fn = log_likelihood_fn,
  obs_times = obs_times,
  mu = 1,
  sigma = 1,
)
plot(result$state_est, type = "l", col = "blue", main = "State Estimates")
```

---

## pmmh *Particle Marginal Metropolis-Hastings (PMMH) for State-Space Models*

---

### Description

This function implements a Particle Marginal Metropolis-Hastings (PMMH) algorithm to perform Bayesian inference in state-space models. It first runs a pilot chain to tune the proposal distribution and the number of particles for the particle filter, and then runs the main PMMH chain.

### Usage

```
pmmh(
  y,
  m,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  log_priors,
  pilot_init_params,
  burn_in,
  num_chains = 4,
  obs_times = NULL,
  algorithm = c("SISAR", "SISR", "SIS"),
  resample_fn = c("stratified", "systematic", "multinomial"),
  param_transform = NULL,
  tune_control = default_tune_control(),
  verbose = FALSE,
  return_latent_state_est = FALSE,
  seed = NULL,
  num_cores = 1
)
```

### Arguments

| | |
|---|---|
| `y` | A numeric vector or matrix of observations. Each row represents an observation at a time step. |
| `m` | An integer specifying the total number of MCMC iterations. |
| `init_fn` | A function to initialize the state-space model. |
| `transition_fn` | A function that defines the state transition of the state-space model. |

| | |
|---|---|
| `log_likelihood_fn` | A function that calculates the log-likelihood for the state-space model given latent states. |
| `log_priors` | A list of functions for computing the log-prior of each parameter. |
| `pilot_init_params` | A list of initial parameter values. Should be a list of length `num_chains` where each element is a named vector of initial parameter values. |
| `burn_in` | An integer indicating the number of initial MCMC iterations to discard as burn-in. |
| `num_chains` | An integer specifying the number of PMMH chains to run. |
| `obs_times` | A numeric vector indicating the time points at which observations in `y` are available. Must be of the same length as the number of rows in `y`. If not specified, it is assumed that observations are available at consecutive time steps, i.e., `obs_times = 1:nrow(y)`. |
| `algorithm` | A character string specifying the particle filtering algorithm to use. Must be one of `"SISAR"`, `"SISR"`, or `"SIS"`. Defaults to `"SISAR"`. |
| `resample_fn` | A character string specifying the resampling method. Must be one of `"stratified"`, `"systematic"`, or `"multinomial"`. Defaults to `"stratified"`. |
| `param_transform` | An optional character vector that specifies the transformation applied to each parameter before proposing. The proposal is made using a multivariate normal distribution on the transformed scale. Parameters are then mapped back to their original scale before evaluation. Currently supports `"log"`, `"invlogit"`, and `"identity"`. If `NULL`, the `"identity"` transformation is used for all parameters. |
| `tune_control` | A list generated by `default_tune_control` containing tuning parameters for the pilot chain, such as `pilot_m`, `pilot_n`, `pilot_reps`, `pilot_proposal_sd`, `pilot_algorithm`, and `pilot_resample_fn`. |
| `verbose` | A logical value indicating whether to print information about pilot_run tuning. Defaults to `FALSE`. |
| `return_latent_state_est` | A logical value indicating whether to return the latent state estimates for each time step. Defaults to `FALSE`. |
| `seed` | An optional integer to set the seed for reproducibility. |
| `num_cores` | An integer specifying the number of cores to use for parallel processing. Defaults to 1. Each chain is assigned to its own core, so the number of cores cannot exceed the number of chains (`num_chains`). The progress information given to user is limited if using more than one core. |

## Details

The PMMH algorithm is essentially a Metropolis Hastings algorithm where instead of using the exact likelihood it instead uses an estimated using likelihood using a particle filter (see also `particle_filter`). Values are proposed using a multivariate normal distribution in the transformed space. The proposal covariance is estimated using the pilot chain.

## Value

A list containing:

`theta_chain` A dataframe of post burn-in parameter samples.

`latent_state_chain` If `return_latent_state_est` is `TRUE`, a list of matrices containing the latent state estimates for each time step.

`diagnostics` Diagnostics containing ESS and Rhat for each parameter (see `ess` and `rhat` for documentation).

## References

Andrieu et al. (2010). Particle Markov chain Monte Carlo methods. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72(3):269–342. doi: 10.1111/j.1467-9868.2009.00736.x

## Examples

```
init_fn <- function(particles) {
  rnorm(particles, mean = 0, sd = 1)
}
transition_fn <- function(particles, phi, sigma_x) {
  phi * particles + sin(particles) +
    rnorm(length(particles), mean = 0, sd = sigma_x)
}
log_likelihood_fn <- function(y, particles, sigma_y) {
  dnorm(y, mean = particles, sd = sigma_y, log = TRUE)
}
log_prior_phi <- function(phi) {
  dnorm(phi, mean = 0, sd = 1, log = TRUE)
}
log_prior_sigma_x <- function(sigma) {
  dexp(sigma, rate = 1, log = TRUE)
}
log_prior_sigma_y <- function(sigma) {
  dexp(sigma, rate = 1, log = TRUE)
}
log_priors <- list(
  phi = log_prior_phi,
  sigma_x = log_prior_sigma_x,
  sigma_y = log_prior_sigma_y
)
# Generate data
t_val <- 10
x <- numeric(t_val)
y <- numeric(t_val)
x[1] <- rnorm(1, mean = 0, sd = 1)
y[1] <- rnorm(1, mean = x[1], sd = 0.5)
for (t in 2:t_val) {
  x[t] <- 0.8 * x[t - 1] + sin(x[t - 1]) + rnorm(1, mean = 0, sd = 1)
  y[t] <- x[t] + rnorm(1, mean = 0, sd = 0.5)
}
# Should use much higher MCMC iterations in practice (m)
```

```
pmmh_result <- pmmh(
  y = y,
  m = 1000,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  log_priors = log_priors,
  pilot_init_params = list(
    c(phi = 0.8, sigma_x = 1, sigma_y = 0.5),
    c(phi = 1, sigma_x = 0.5, sigma_y = 1)
  ),
  burn_in = 100,
  num_chains = 2,
  param_transform = list(
    phi = "identity",
    sigma_x = "log",
    sigma_y = "log"
  ),
  tune_control = default_tune_control(pilot_m = 500, pilot_burn_in = 100)
)
# Convergence warning is expected with such low MCMC iterations.

# Suppose we have data for t=1,2,3,5,6,7,8,9,10 (i.e., missing at t=4)

obs_times <- c(1, 2, 3, 5, 6, 7, 8, 9, 10)
y <- y[obs_times]

# Specify observation times in the pmmh using obs_times
pmmh_result <- pmmh(
  y = y,
  m = 1000,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  log_priors = log_priors,
  pilot_init_params = list(
    c(phi = 0.8, sigma_x = 1, sigma_y = 0.5),
    c(phi = 1, sigma_x = 0.5, sigma_y = 1)
  ),
  burn_in = 100,
  num_chains = 2,
  obs_times = obs_times,
  param_transform = list(
    phi = "identity",
    sigma_x = "log",
    sigma_y = "log"
  ),
  tune_control = default_tune_control(pilot_m = 500, pilot_burn_in = 100)
)
```

---

## `print.pmmh_output` *Print method for PMMH output*

---

## Description

Print method for PMMH output

## Usage

```
## S3 method for class 'pmmh_output'
print(x, ...)
```

## Arguments

| | |
|---|---|
| `x` | An object of class ‘pmmh_output‘. |
| `...` | Additional arguments. |

## Value

The object ‘x‘ invisibly.

## Examples

```
# Create dummy chains for two parameters across two chains
chain1 <- data.frame(param1 = rnorm(100), param2 = rnorm(100), chain = 1)
chain2 <- data.frame(param1 = rnorm(100), param2 = rnorm(100), chain = 2)
dummy_output <- list(
  theta_chain = rbind(chain1, chain2),
  diagnostics = list(
    ess = c(param1 = 200, param2 = 190),
    rhat = c(param1 = 1.01, param2 = 1.00)
  )
)
class(dummy_output) <- "pmmh_output"
print(dummy_output)
```

---

# `rhat` *Compute split Rhat statistic*

## Description

Compute split Rhat statistic

## Usage

```
rhat(chains)
```

## Arguments

| | |
|---|---|
| `chains` | A matrix (iterations x chains) or a data.frame with a ’chain’ column and parameter columns. |

### Details

Uses the formula for split-Rhat proposed by Gelman et al. (2013).

### Value

Rhat value (matrix input) or named vector of Rhat values.

### References

Gelman et al. (2013). Bayesian Data Analysis, 3rd Edition.

### Examples

```
# Example with matrix
chains <- matrix(rnorm(3000), nrow = 1000, ncol = 3)
rhat(chains)
#' # Example with data frame
chains_df <- data.frame(
  chain = rep(1:3, each = 1000),
  param1 = rnorm(3000),
  param2 = rnorm(3000)
)
rhat(chains_df)
```

---

## `summary.pmmh_output` *Print method for PMMH output*

### Description

Print method for PMMH output

### Usage

```
## S3 method for class 'pmmh_output'
summary(object, ...)
```

### Arguments

| | |
|---|---|
| `object` | An object of class ‘pmmh_output‘. |
| `...` | Additional arguments. |

### Value

A data frame containing summary statistics for each parameter.

## Examples

```
# Create dummy chains for two parameters across two chains
chain1 <- data.frame(param1 = rnorm(100), param2 = rnorm(100), chain = 1)
chain2 <- data.frame(param1 = rnorm(100), param2 = rnorm(100), chain = 2)
dummy_output <- list(
  theta_chain = rbind(chain1, chain2),
  diagnostics = list(
    ess = c(param1 = 200, param2 = 190),
    rhat = c(param1 = 1.01, param2 = 1.00)
  )
)
class(dummy_output) <- "pmmh_output"
summary(dummy_output)
```

# Index