# Package: bayesSSM (via r-universe)

May 16, 2026

**Type** Package

**Title** Bayesian Methods for State Space Models

**Version** 0.7.1.9000

**Description** Implements methods for Bayesian analysis of State Space Models. Includes implementations of 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

**LazyData** true

**RoxygenNote** 7.3.3

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

**LinkingTo** Rcpp

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

**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

**Repository** https://bjarkehautop.r-universe.dev

**Date/Publication** 2025-12-16 07:17:55 UTC

**RemoteUrl** https://github.com/bjarkehautop/bayesssm

**RemoteRef** HEAD

**RemoteSha** 40b765c400957048e9b42d05dfc8b0d99dd150c7

# Contents



---

`auxiliary_filter` *Auxiliary Particle Filter (APF)*

---

## Description

The Auxiliary Particle Filter differs from the bootstrap filter by incorporating a look-ahead step: particles are reweighted using an approximation of the likelihood of the next observation prior to resampling. This adjustment can help reduce particle degeneracy and, improve filtering efficiency compared to the bootstrap approach.

## Usage

```
auxiliary_filter(
  y,
  num_particles,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  aux_log_likelihood_fn,
  obs_times = NULL,
  resample_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. If observations are not equally spaced, use the `obs_times` argument. |
| `num_particles` | A positive integer specifying the number of particles. |

`init_fn` A function to initialize the particles. Should take 'num_particles' and return a matrix or vector of initial states. Additional model parameters can be passed via ....

`transition_fn` A function for propagating particles. Should take 'particles' and optionally 't'. Additional model parameters via ....

`log_likelihood_fn` A function that returns the log-likelihood for each particle given the current observation, particles, and optionally 't'. Additional parameters via ....

`aux_log_likelihood_fn` A function that computes the log-likelihood of the next observation given the current particles. It should accept arguments 'y', 'particles', optionally 't', and any additional model-specific parameters via .... It returns a numeric vector of log-likelihoods.

`obs_times` A numeric vector specifying observation time points. Must match the number of rows in `y`, or defaults to `1:nrow(y)`.

`resample_algorithm` A character string specifying the filtering resample algorithm: `"SIS"` for no resampling, `"SISR"` for resampling at every time step, or `"SISAR"` for adaptive resampling when ESS drops below `threshold`. Using `"SISR"` or `"SISAR"` to avoid weight degeneracy is recommended. Default is `"SISAR"`.

`resample_fn` A string indicating the resampling method: `"stratified"`, `"systematic"`, or `"multinomial"`. Default is `"stratified"`.

`threshold` A numeric value specifying the ESS threshold for `"SISAR"`. Defaults to `num_particles / 2`.

`return_particles` Logical; if `TRUE`, returns the full particle and weight histories.

`...` Additional arguments passed to `init_fn`, `transition_fn`, and `log_likelihood_fn`.

## Value

A list with components:

**state_est** Estimated states over time (weighted mean of particles).

**ess** Effective sample size at each time step.

**loglike** Total log-likelihood.

**loglike_history** Log-likelihood at each time step.

**algorithm** The filtering algorithm used.

**particles_history** Matrix of particle states over time (if `return_particles = TRUE`).

**weights_history** Matrix of particle weights over time (if `return_particles = TRUE`).

## The Auxiliary Particle Filter (APF)

The Auxiliary Particle Filter (APF) was introduced by Pitt and Shephard (1999) to improve upon the standard bootstrap filter by incorporating a look ahead step. Before resampling at time $t$, particles are weighted by an auxiliary weight proportional to an estimate of the likelihood of the next observation, guiding resampling to favor particles likely to contribute to future predictions.

Specifically, if $w_{t-1}^i$ are the normalized weights and $x_{t-1}^i$ are the particles at time $t-1$, then auxiliary weights are computed as

$$\tilde{w}_t^i \propto w_{t-1}^i \, p(y_t | \mu_t^i),$$

where $\mu_t^i$ is a predictive summary (e.g., the expected next state) of the particle $x_{t-1}^i$. Resampling is performed using $\tilde{w}_t^i$ instead of $w_{t-1}^i$. This can reduce the variance of the importance weights at time $t$ and help mitigate particle degeneracy, especially if the auxiliary weights are chosen well.

Default resampling method is stratified resampling, which has lower variance than multinomial resampling (Douc et al., 2005).

## Model Specification

Particle filter implementations in this package assume a discrete-time state-space model defined by:

- A sequence of latent states $x_0, x_1, \ldots, x_T$ evolving according to a Markov process.
- Observations $y_1, \ldots, y_T$ that are conditionally independent given the corresponding latent states.

The model is specified as:

$$x_0 \sim \mu_\theta$$
$$x_t \sim f_\theta(x_t \mid x_{t-1}), \quad t = 1, \ldots, T$$
$$y_t \sim g_\theta(y_t \mid x_t), \quad t = 1, \ldots, T$$

where $\theta$ denotes model parameters passed via ....

The user provides the following functions:

- `init_fn`: draws from the initial distribution $\mu_\theta$.
- `transition_fn`: generates or evaluates the transition density $f_\theta$.
- `weight_fn`: evaluates the observation likelihood $g_\theta$.

## References

Pitt, M. K., & Shephard, N. (1999). Filtering via simulation: Auxiliary particle filters. Journal of the American Statistical Association, 94(446), 590–599. doi:10.1080/01621459.1999.10474153

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(num_particles) rnorm(num_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)
}
aux_log_likelihood_fn <- function(y, particles) {
  # Predict next state (mean stays same) and compute log p(y | x)
  mean_forecast <- particles # since E[x'] = x in this model
```

```
  dnorm(y, mean = mean_forecast, sd = 1, log = TRUE)
}

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

result <- auxiliary_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  aux_log_likelihood_fn = aux_log_likelihood_fn
)
plot(result$state_est,
  type = "l", col = "blue", main = "APF: State Estimates",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)

# ---- With parameters ----
init_fn <- function(num_particles) rnorm(num_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)
}
aux_log_likelihood_fn <- function(y, particles, mu, sigma) {
  # Forecast mean of x' given x, then evaluate p(y | forecast)
  forecast <- particles + mu
  dnorm(y, mean = forecast, sd = sigma, log = TRUE)
}

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

result <- auxiliary_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  aux_log_likelihood_fn = aux_log_likelihood_fn,
  mu = 1,
  sigma = 1
)
plot(result$state_est,
  type = "l", col = "blue", main = "APF with Parameters",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)
```

```
# ---- With observation gaps ----
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)
obs_times <- c(1, 2, 3, 5, 6, 7, 8, 9, 10) # Missing at t = 4
data_obs <- data[obs_times]

init_fn <- function(num_particles) rnorm(num_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)
}
aux_log_likelihood_fn <- function(y, particles, mu, sigma) {
  forecast <- particles + mu
  dnorm(y, mean = forecast, sd = sigma, log = TRUE)
}

num_particles <- 100
result <- auxiliary_filter(
  y = data_obs,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  aux_log_likelihood_fn = aux_log_likelihood_fn,
  obs_times = obs_times,
  mu = 1,
  sigma = 1
)
plot(result$state_est,
  type = "l", col = "blue", main = "APF with Observation Gaps",
  ylim = range(c(result$state_est, data))
)
points(data_obs, col = "red", pch = 20)
```

---

`bootstrap_filter` *Bootstrap Particle Filter (BPF)*

---

## Description

Implements a bootstrap particle filter for sequential Bayesian inference in state space models using sequential Monte Carlo methods.

## Usage

```
bootstrap_filter(
  y,
  num_particles,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  obs_times = NULL,
  resample_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. If observations are not equally spaced, use the obs_times argument. |
| num_particles | A positive integer specifying the number of particles. |
| init_fn | A function to initialize the particles. Should take 'num_particles' and return a matrix or vector of initial states. Additional model parameters can be passed via .... |
| transition_fn | A function for propagating particles. Should take 'particles' and optionally 't'. Additional model parameters via .... |
| log_likelihood_fn | A function that returns the log-likelihood for each particle given the current observation, particles, and optionally 't'. Additional parameters via .... |
| obs_times | A numeric vector specifying observation time points. Must match the number of rows in y, or defaults to 1:nrow(y). |
| resample_algorithm | A character string specifying the filtering resample algorithm: "SIS" for no resampling, "SISR" for resampling at every time step, or "SISAR" for adaptive resampling when ESS drops below threshold. Using "SISR" or "SISAR" to avoid weight degeneracy is recommended. Default is "SISAR". |
| resample_fn | A string indicating the resampling method: "stratified", "systematic", or "multinomial". Default is "stratified". |
| threshold | A numeric value specifying the ESS threshold for "SISAR". Defaults to num_particles / 2. |

`return_particles`
: Logical; if `TRUE`, returns the full particle and weight histories.

`...`
: Additional arguments passed to `init_fn`, `transition_fn`, and `log_likelihood_fn`.

## Value

A list with components:

**state_est** Estimated states over time (weighted mean of particles).

**ess** Effective sample size at each time step.

**loglike** Total log-likelihood.

**loglike_history** Log-likelihood at each time step.

**algorithm** The filtering algorithm used.

**particles_history** Matrix of particle states over time (if `return_particles = TRUE`).

**weights_history** Matrix of particle weights over time (if `return_particles = TRUE`).

## The Effective Sample Size (ESS) is defined as

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

where $w_i$ are the normalized weights of the particles.

Default resampling method is stratified resampling, which has lower variance than multinomial resampling (Douc et al., 2005).

## Model Specification

Particle filter implementations in this package assume a discrete-time state-space model defined by:

- A sequence of latent states $x_0, x_1, \ldots, x_T$ evolving according to a Markov process.
- Observations $y_1, \ldots, y_T$ that are conditionally independent given the corresponding latent states.

The model is specified as:

$$x_0 \sim \mu_\theta$$
$$x_t \sim f_\theta(x_t \mid x_{t-1}), \quad t = 1, \ldots, T$$
$$y_t \sim g_\theta(y_t \mid x_t), \quad t = 1, \ldots, T$$

where $\theta$ denotes model parameters passed via ....

The user provides the following functions:

- `init_fn`: draws from the initial distribution $\mu_\theta$.
- `transition_fn`: generates or evaluates the transition density $f_\theta$.
- `weight_fn`: evaluates the observation likelihood $g_\theta$.

## References

Gordon, N. J., Salmond, D. J., & Smith, A. F. M. (1993). Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEE Proceedings F (Radar and Signal Processing), 140(2), 107–113. doi:10.1049/ipf2.1993.0015

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(num_particles) rnorm(num_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)
}

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

# Run the particle filter using default settings.
result <- bootstrap_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn
)
plot(result$state_est,
  type = "l", col = "blue", main = "State Estimates",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)

# With parameters
init_fn <- function(num_particles) rnorm(num_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)
}

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

# Run the bootstrap particle filter using default settings.
result <- bootstrap_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$state_est,
  type = "l", col = "blue", main = "State Estimates",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)

# With observations gaps
init_fn <- function(num_particles) rnorm(num_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_obs <- data[obs_times]

num_particles <- 100
# Specify observation times in the bootstrap particle filter using obs_times
result <- bootstrap_filter(
  y = data_obs,
  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$state_est,
  type = "l", col = "blue", main = "State Estimates",
  ylim = range(c(result$state_est, data))
)
```

```
points(data_obs, col = "red", pch = 20)
```

---

`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_resample_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_resample_algorithm`
: The resample_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 functions*

---

## Description

The package provides several particle filter implementations for state-space models for estimating the intractable marginal likelihood $p(y_{1:T} \mid \theta)$:

- `auxiliary_filter`
- `bootstrap_filter`
- `resample_move_filter`

The simplest one is the `bootstrap_filter`, and is thus recommended as a starting point.

---

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

---

## Description

This function implements a Particle Marginal Metropolis-Hastings (PMMH) resample_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(
  pf_wrapper,
  y,
  m,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  log_priors,
  pilot_init_params,
  burn_in,
  num_chains = 4,
  obs_times = NULL,
  resample_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

| | |
|---|---|
| pf_wrapper | A particle filter wrapper function. See particle_filter for the particle filters implemented in this package. |
| y | A numeric vector or matrix of observations. Each row represents an observation at a time step. If observations are not equally spaced, use the obs_times argument. |
| m | An integer specifying the number of MCMC iterations for each chain. |
| init_fn | A function to initialize the particles. Should take 'num_particles' and return a matrix or vector of initial states. Additional model parameters can be passed via .... |
| transition_fn | A function for propagating particles. Should take 'particles' and optionally 't'. Additional model parameters via .... |
| log_likelihood_fn | A function that returns the log-likelihood for each particle given the current observation, particles, and optionally 't'. Additional parameters via .... |
| 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 specifying observation time points. Must match the number of rows in y, or defaults to 1:nrow(y). |
| resample_algorithm | A character string specifying the resampling algorithm to use in the particle filter. Options are: #'<br>• SIS: Sequential Importance Sampling (without resampling).<br>• SISR: Sequential Importance Sampling with resampling at every time step.<br>• 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). Can be modified by specifying the threshold argument (in ...), see also particle_filter. |
| resample_fn | A string indicating the resampling method: "stratified", "systematic", or "multinomial". Default is "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"`, `"logit"`, and `"identity"`. If `NULL`, the `"identity"` transformation is used for all parameters.

`tune_control`
: A list of pilot tuning controls (e.g., `pilot_m`, `pilot_reps`). See `default_tune_control`.

`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.

`...`
: Additional arguments passed to `init_fn`, `transition_fn`, and `log_likelihood_fn`.

## Details

The PMMH resample_algorithm is essentially a Metropolis Hastings algorithm, where instead of using the intractable marginal likelihood $p(y_{1:T} \mid \theta)$ it instead uses the estimated likelihood using a particle filter (see `particle_filter` for available particle filters). Values are proposed using a multivariate normal distribution in the transformed space (specified using 'param_transform'). The proposal covariance and the number of particles is chosen based on a pilot run. The number of particles is chosen such that the variance of the log-likelihood estimate at the estimated posterior mean is approximately 1 (with a minimum of 50 particles and a maximum of 1000).

## 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(num_particles) {
  rnorm(num_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 = cos(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)
phi <- 0.8
sigma_x <- 1
sigma_y <- 0.5

init_state <- rnorm(1, mean = 0, sd = 1)
x[1] <- phi * init_state + sin(init_state) + rnorm(1, mean = 0, sd = sigma_x)
y[1] <- x[1] + rnorm(1, mean = 0, sd = sigma_y)
for (t in 2:t_val) {
  x[t] <- phi * x[t - 1] + sin(x[t - 1]) + rnorm(1, mean = 0, sd = sigma_x)
  y[t] <- cos(x[t]) + rnorm(1, mean = 0, sd = sigma_y)
}
x <- c(init_state, x)

# Should use higher MCMC iterations in practice (m)
pmmh_result <- pmmh(
  pf_wrapper = bootstrap_filter,
  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(
  pf_wrapper = bootstrap_filter,
  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

Displays a concise summary of parameter estimates from a PMMH output object, including means, standard deviations, medians, 95% credible intervals, effective sample sizes (ESS), and Rhat. This provides a quick overview of the posterior distribution and convergence diagnostics.

### 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)
```

---

## resample_move_filter *Resample-Move Particle Filter (RMPF)*

---

### Description

The Resample-Move Particle Filter differs from standard resampling methods by including a Metropolis–Hastings move step after resampling. This additional step can increase particle diversity and, in some contexts, help mitigate sample impoverishment.

### Usage

```
resample_move_filter(
  y,
  num_particles,
  init_fn,
  transition_fn,
  log_likelihood_fn,
  move_fn,
  obs_times = NULL,
  resample_fn = c("stratified", "systematic", "multinomial"),
```

```
  return_particles = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| y | A numeric vector or matrix of observations. Each row represents an observation at a time step. If observations are not equally spaced, use the obs_times argument. |
| num_particles | A positive integer specifying the number of particles. |
| init_fn | A function to initialize the particles. Should take 'num_particles' and return a matrix or vector of initial states. Additional model parameters can be passed via .... |
| transition_fn | A function for propagating particles. Should take 'particles' and optionally 't'. Additional model parameters via .... |
| log_likelihood_fn | A function that returns the log-likelihood for each particle given the current observation, particles, and optionally 't'. Additional parameters via .... |
| move_fn | A function that moves the resampled particles. Takes 'particles', optionally 't', and returns updated particles. Can use ... for model-specific arguments. |
| obs_times | A numeric vector specifying observation time points. Must match the number of rows in y, or defaults to 1:nrow(y). |
| resample_fn | A string indicating the resampling method: "stratified", "systematic", or "multinomial". Default is "stratified". |
| return_particles | Logical; if TRUE, returns the full particle and weight histories. |
| ... | Additional arguments passed to init_fn, transition_fn, and log_likelihood_fn. |

## Value

A list with components:

**state_est** Estimated states over time (weighted mean of particles).

**ess** Effective sample size at each time step.

**loglike** Total log-likelihood.

**loglike_history** Log-likelihood at each time step.

**algorithm** The filtering algorithm used.

**particles_history** Matrix of particle states over time (if return_particles = TRUE).

**weights_history** Matrix of particle weights over time (if return_particles = TRUE).

### The Resample-Move Particle Filter (RMPF)

The Resample-Move Particle Filter enhances the standard particle filtering framework by introducing a move step after resampling. After resampling at time $t$, particles $\{x_t^{(i)}\}_{i=1}^N$ are propagated via a Markov kernel $K_t(x' \mid x)$ that leaves the target posterior $p(x_t \mid y_{1:t})$ invariant:

$$x_t^{(i)} \sim K_t(\cdot \mid x_t^{(i)}).$$

This move step often uses a Metropolis-Hastings update that preserves the posterior distribution as the invariant distribution of $K_t$.

The goal of the move step is to mitigate particle impoverishment — the collapse of diversity caused by resampling selecting only a few unique particles — by rejuvenating particles and exploring the state space more thoroughly. This leads to improved approximation of the filtering distribution and reduces Monte Carlo error.

The `move_fn` argument represents this transition kernel and should take the current particle set as input and return the updated particles. Additional model-specific parameters may be passed via ....

Default resampling method is stratified resampling, which has lower variance than multinomial resampling (Douc et al., 2005).

In this implementation, resampling is performed at every time step using the specified method (default: stratified), followed immediately by the move step. This follows the standard Resample-Move framework as described by Gilks and Berzuini (2001). Unlike other particle filtering variants that may use an ESS threshold to decide whether to resample, RMPF requires resampling at every step to ensure the effectiveness of the subsequent rejuvenation step.

### Model Specification

Particle filter implementations in this package assume a discrete-time state-space model defined by:

- A sequence of latent states $x_0, x_1, \ldots, x_T$ evolving according to a Markov process.
- Observations $y_1, \ldots, y_T$ that are conditionally independent given the corresponding latent states.

The model is specified as:

$$x_0 \sim \mu_\theta$$

$$x_t \sim f_\theta(x_t \mid x_{t-1}), \quad t = 1, \ldots, T$$

$$y_t \sim g_\theta(y_t \mid x_t), \quad t = 1, \ldots, T$$

where $\theta$ denotes model parameters passed via ....

The user provides the following functions:

- `init_fn`: draws from the initial distribution $\mu_\theta$.
- `transition_fn`: generates or evaluates the transition density $f_\theta$.
- `weight_fn`: evaluates the observation likelihood $g_\theta$.

## References

Gilks, W. R., & Berzuini, C. (2001). Following a moving target—Monte Carlo inference for dynamic Bayesian models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63(1), 127–146. doi:10.2307/2670179

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(num_particles) rnorm(num_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)
}

# Define a simple random-walk Metropolis move function
move_fn <- function(particle, y) {
  proposal <- particle + rnorm(1, 0, 0.1)
  log_p_current <- log_likelihood_fn(y = y, particles = particle)
  log_p_proposal <- log_likelihood_fn(y = y, particles = proposal)
  if (log(runif(1)) < (log_p_proposal - log_p_current)) {
    return(proposal)
  } else {
    return(particle)
  }
}

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

result <- resample_move_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  move_fn = move_fn
)
plot(result$state_est,
  type = "l", col = "blue", main = "RMPF State Estimates",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)


# With parameters
init_fn <- function(num_particles) rnorm(num_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)
}
move_fn <- function(particle, y, sigma) {
  proposal <- particle + rnorm(1, 0, 0.1)
  log_p_curr <- log_likelihood_fn(y = y, particles = particle, sigma = sigma)
  log_p_prop <- log_likelihood_fn(y = y, particles = proposal, sigma = sigma)
  if (log(runif(1)) < (log_p_prop - log_p_curr)) {
    return(proposal)
  } else {
    return(particle)
  }
}

y <- cumsum(rnorm(50))
num_particles <- 100

result <- resample_move_filter(
  y = y,
  num_particles = num_particles,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  move_fn = move_fn,
  mu = 1,
  sigma = 1
)
plot(result$state_est,
  type = "l", col = "blue", main = "RMPF with Parameters",
  ylim = range(c(result$state_est, y))
)
points(y, col = "red", pch = 20)


# With observation gaps
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)
obs_times <- c(1, 2, 3, 5, 6, 7, 8, 9, 10) # skip t=4
data_obs <- data[obs_times]

init_fn <- function(num_particles) rnorm(num_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)
}
move_fn <- function(particle, y, sigma) {
  proposal <- particle + rnorm(1, 0, 0.1)
  log_p_cur <- log_likelihood_fn(y = y, particles = particle, sigma = sigma)
  log_p_prop <- log_likelihood_fn(y = y, particles = proposal, sigma = sigma)
  if (log(runif(1)) < (log_p_prop - log_p_cur)) {
    return(proposal)
  } else {
    return(particle)
  }
}

result <- resample_move_filter(
  y = data_obs,
  num_particles = 100,
  init_fn = init_fn,
  transition_fn = transition_fn,
  log_likelihood_fn = log_likelihood_fn,
  move_fn = move_fn,
  obs_times = obs_times,
  mu = 1,
  sigma = 1
)
plot(result$state_est,
  type = "l", col = "blue", main = "RMPF with Observation Gaps",
  ylim = range(c(result$state_est, data))
)
points(data_obs, col = "red", pch = 20)
```

---

`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 *Summary method for PMMH output*

---

## Description

This function returns summary statistics for PMMH output objects, including means, standard deviations, medians, credible intervals, and diagnostics.

## 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