

# Package ‘SimBIID’

September 19, 2019

**Title** Simulation-Based Inference Methods for Infectious Disease Models

**Version** 0.1.2

**Description** Provides some code to run simulations of state-space models, and then use these in the Approximate Bayesian Computation Sequential Monte Carlo (ABC-SMC) algorithm of Toni et al. (2009) <doi:10.1098/rsif.2008.0172> and a bootstrap particle filter based particle Markov chain Monte Carlo (PMCMC) algorithm (Andrieu et al., 2010 <doi:10.1111/j.1467-9868.2009.00736.x>). Also provides functions to plot and summarise the outputs.

**License** GPL (>= 3)

**URL** <https://github.com/tjmckinley/SimBIID>

**BugReports** <https://github.com/tjmckinley/SimBIID/issues>

**Depends** R (>= 3.5)

**Imports** stats, dplyr, purrr, tibble, ggplot2, tidyr, mvtnorm, grDevices, RColorBrewer, Rcpp, RcppXPtUtils, coda

**Suggests** parallel, GGally, testthat

**LinkingTo** Rcpp, RcppArmadillo

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**NeedsCompilation** yes

**Author** Trevelyan J. McKinley [aut, cre],  
Stefan Widgren [aut] (Author of 'R/mparse.R'),  
Pavol Bauer [cph] (R/mparse.R),  
Robin Eriksson [cph] (R/mparse.R),  
Stefan Engblom [cph] (R/mparse.R)

**Maintainer** Trevelyan J. McKinley <t.mckinley@exeter.ac.uk>

**Repository** CRAN

**Date/Publication** 2019-09-19 14:20:02 UTC

**R topics documented:**

SimBIID-package . . . . .	2
ABCRef . . . . .	3
ABCSCMC . . . . .	5
compileRcpp . . . . .	8
ebola . . . . .	10
mparseRcpp . . . . .	10
plot.ABCSCMC . . . . .	13
plot.PMCMC . . . . .	15
plot.SimBIID_runs . . . . .	17
PMCMC . . . . .	19
predict.PMCMC . . . . .	22
print.ABCSCMC . . . . .	24
print.PMCMC . . . . .	27
print.SimBIID_model . . . . .	29
print.SimBIID_runs . . . . .	29
run . . . . .	30
smallpox . . . . .	32
summary.ABCSCMC . . . . .	33
summary.PMCMC . . . . .	35
window.PMCMC . . . . .	37
<b>Index</b>	<b>40</b>

---

 SimBIID-package

*Simulation-based inference for infectious disease models*


---

**Description**

Package implements various simulation-based inference routines for infectious disease models.

**Details**

Provides some code to run simulations of state-space models, and then

**Author(s)**

Trevelyan J. McKinley <t.mckinley@exeter.ac.uk>

---

ABCRef	<i>Produces ABC reference table</i>
--------	-------------------------------------

---

**Description**

Produces reference table of simulated outcomes for use in various Approximate Bayesian Computation (ABC) algorithms.

**Usage**

```
ABCRef(npart, priors, pars, func, sumNames, parallel = FALSE,
       mc.cores = NA, ...)
```

**Arguments**

npart	The number of particles (must be a positive integer).
priors	A data.frame containing columns parnames, dist, p1 and p2, with number of rows equal to the number of parameters. The column parname simply gives names to each parameter for plotting and summarising. Each entry in the dist column must contain one of c("unif", "norm", "gamma"), and the corresponding p1 and p2 entries relate to the hyperparameters (lower and upper bounds in the uniform case; mean and standard deviation in the normal case; and shape and rate in the gamma case).
pars	A named vector or matrix of parameters to use for the simulations. If pars is a vector then this is repeated 'npart' times, else it must be a matrix with 'npart' rows. You cannot specify both 'pars' and 'priors'.
func	Function that runs the simulator. The first argument must be pars. The function must return a vector of simulated summary measures, or a missing value (NA) if there is an error. The output from the function must be a vector with length equal to length(sumNames).
sumNames	A character vector of summary statistic names.
parallel	A logical determining whether to use parallel processing or not.
mc.cores	Number of cores to use if using parallel processing.
...	Extra arguments to be passed to func.

**Details**

Runs simulations for a large number of particles, either pre-specified or sampled from the a set of given prior distributions. Returns a table of summary statistics for each particle. Useful for deciding on initial tolerances during an [ABC SMC](#) run, or for producing a reference table to use in e.g. the ABC with Random Forests approach of Raynal et al. (2017).

**Value**

An data.frame object with npart rows, where the first p columns correspond to the proposed parameters, and the remaining columns correspond to the simulated outputs.

## References

Raynal, L, Marin J-M, Pudlo P, Ribatet M, Robert CP and Estoup A. (2017) <ArXiv:1605.05537>

## Examples

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)
model <- compileRcpp(model)

## generate function to run simulators
## and produce final epidemic size and time
## summary statistics
simRef <- function(pars, model) {
  ## run model over a 100 day period with
  ## one initial infective in a population
  ## of 120 individuals
  sims <- model(pars, 0, 100, c(119, 1, 0))

  ## return vector of summary statistics
  c(finaltime = sims[2], finalsize = sims[5])
}

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma"),
  dist = rep("gamma", 2),
  stringsAsFactors = FALSE
)
priors$p1 <- c(10, 10)
priors$p2 <- c(10^4, 10^2)

## produce reference table by sampling from priors
## (add additional arguments to 'func' at the end)
refTable <- ABCRef(
  npart = 100,
  priors = priors,
  func = simRef,
  sumNames = c("finaltime", "finalsize"),
  model = model
)
refTable
```

---

ABCSCMC

*Runs ABC-SMC algorithm*


---

### Description

Runs the Approximate Bayesian Computation Sequential Monte Carlo (ABC-SMC) algorithm of Toni et al. (2009) for fitting infectious disease models to time series count data.

### Usage

```
ABCSCMC(x, ...)
```

```
## S3 method for class 'ABCSCMC'
```

```
ABCSCMC(x, tols = NULL, ptols = NULL, ngen = 1,
        parallel = FALSE, mc.cores = NA, ...)
```

```
## Default S3 method:
```

```
ABCSCMC(x, priors, func, u, tols = NULL, ptols = NULL,
        ngen = 1, npart = 100, parallel = FALSE, mc.cores = NA, ...)
```

### Arguments

<code>x</code>	An ABCSCMC object or a named vector with entries containing the observed summary statistics to match to. Names must match to 'tol's'.
<code>...</code>	Further arguments to pass to <code>func</code> . (Not used if extending runs.)
<code>tol's</code>	A vector or matrix of tolerances, with the number of rows defining the number of generations required, and columns defining the summary statistics to match to. If a vector, then the length determines the summary statistics. The columns/entries must match to those in 'x'.
<code>ptols</code>	The proportion of simulated outcomes at each generation to use to derive adaptive tolerances.
<code>ngen</code>	The number of generations of ABC-SMC to run.
<code>parallel</code>	A logical determining whether to use parallel processing or not.
<code>mc.cores</code>	Number of cores to use if using parallel processing.
<code>priors</code>	A <code>data.frame</code> containing columns <code>parnames</code> , <code>dist</code> , <code>p1</code> and <code>p2</code> , with number of rows equal to the number of parameters. The column <code>parname</code> simply gives names to each parameter for plotting and summarising. Each entry in the <code>dist</code> column must contain one of <code>c("unif", "norm", "gamma")</code> , and the corresponding <code>p1</code> and <code>p2</code> entries relate to the hyperparameters (lower and upper bounds in the uniform case; mean and standard deviation in the normal case; and shape and rate in the gamma case).

func	Function that runs the simulator and checks whether the simulation matches the data. The first four arguments must be pars, data, tols and u. If the simulations do not match the data then the function must return an NA, else it must return a vector of simulated summary measures. In this latter case the output from the function must be a vector with length equal to ncol(data) and with entries in the same order as the columns of data.
u	A named vector of initial states.
npart	An integer specifying the number of particles.

### Details

Samples initial particles from the specified prior distributions and then runs a series of generations of ABC-SMC. The generations can either be specified with a set of fixed tolerances, or by setting the tolerances at each new generation as a quantile of the tolerances of the accepted particles at the previous generation. Passing an ABCSCMC object into the ABCSCMC() function acts as a continuation method, allowing further generations to be run.

### Value

An ABCSCMC object, essentially a list containing:

- pars: a list of matrix objects containing the accepted particles. Each element of the list corresponds to a generation of ABC-SMC, with each matrix being of dimension npart x npars;
- output: a list of matrix objects containing the simulated summary statistics. Each element of the list corresponds to a generation of ABC-SMC, with each matrix being of dimension npart x ncol(data);
- weights: a list of vector objects containing the particle weights. Each element of the list corresponds to a generation of ABC-SMC, with each vector being of length npart;
- ESS: a list of effective sample sizes. Each element of the list corresponds to a generation of ABC-SMC, with each vector being of length npart;
- accrate: a vector of length nrow(tols) containing the acceptance rates for each generation of ABC;
- tols: a copy of the tols input;
- ptols: a copy of the ptols input;
- priors: a copy of the priors input;
- data: a copy of the data input;
- func: a copy of the func input;
- u a copy of the u input;
- addargs: a copy of the ... inputs.

### References

- Toni T, Welch D, Strelkowa N, Ipsen A and Stumpf MP (2009) <doi:10.1098/rsif.2008.0172>  
 McKinley TJ, Cook AR and Deardon R (2009) <doi:10.2202/1557-4679.1171>

**See Also**

[print.ABCSCMC](#), [plot.ABCSCMC](#), [summary.ABCSCMC](#)

**Examples**

```
## set up SIR simulationmodel
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)
model <- compileRcpp(model)

## generate function to run simulators
## and return summary statistics
simSIR <- function(pars, data, tols, u, model) {

  ## run model
  sims <- model(pars, 0, data[2] + tols[2], u)

  ## this returns a vector of the form:
  ## completed (1/0), t, S, I, R (here)
  if(sims[1] == 0) {
    ## if simulation rejected
    return(NA)
  } else {
    ## extract finaltime and finalsize
    finaltime <- sims[2]
    finalsize <- sims[5]
  }

  ## return vector if match, else return NA
  if(all(abs(c(finalsize, finaltime) - data) <= tols)){
    return(c(finalsize, finaltime))
  } else {
    return(NA)
  }
}

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma"),
  dist = rep("gamma", 2),
  stringsAsFactors = FALSE
)
```

```

priors$p1 <- c(10, 10)
priors$p2 <- c(10^4, 10^2)

## define the targeted summary statistics
data <- c(
  finalsize = 30,
  finaltime = 76
)

## set initial states (1 initial infection
## in population of 120)
iniStates <- c(S = 119, I = 1, R = 0)

## set initial tolerances
tols <- c(
  finalsize = 50,
  finaltime = 50
)

## run 2 generations of ABC-SMC
## setting tolerance to be 50th
## percentile of the accepted
## tolerances at each generation
post <- ABCSMC(
  x = data,
  priors = priors,
  func = simSIR,
  u = iniStates,
  tols = tols,
  ptol = 0.2,
  ngen = 2,
  npart = 50,
  model = model
)
post

## run one further generation
post <- ABCSMC(post, ptols = 0.5, ngen = 1)
post
summary(post)

## plot posteriors
plot(post)

## plot outputs
plot(post, "output")

```



**Description**

Compiles an object of class `SimBIID_model` into an `XPtr` object for use in `Rcpp` functions, or an object of class function for calling directly from R.

**Usage**

```
compileRcpp(model)
```

**Arguments**

`model`            An object of class `SimBIID_model`.

**Value**

An object of class `XPtr` that points to the compiled function, or an R function object for calling directly from R.

**See Also**

[mparseRcpp](#)

**Examples**

```
## set up SIR simulationmodel
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)

## compile model to be run directly
model <- compileRcpp(model)
model

## set initial states (1 initial infection
## in population of 120)
iniStates <- c(S = 119, I = 1, R = 0)

## set parameters
pars <- c(beta = 0.001, gamma = 0.1)

## run compiled model
model(pars, 0, 100, iniStates)
```

---

ebola	<i>Time series counts of ebola cases</i>
-------	--

---

**Description**

A dataset containing time series counts for the number of new individuals exhibiting clinical signs, and the number of new removals each day for the 1995 Ebola epidemic in the Democratic Republic of Congo

**Usage**

ebola

**Format**

A data frame with 192 rows and 3 variables:

**time** days from 1st January 1995

**clin\_signs** number of new clinical cases at each day

**removals** number of new removals at each day

**Source**

Khan AS et al. (1999) <doi:10.1086/514306>

---

mparseRcpp	<i>Parse custom model using SimInf style markup</i>
------------	---

---

**Description**

Parse custom model using SimInf style markup. Does not have full functionality of mparse. Currently only supports simulations on a single node.

**Usage**

```
mparseRcpp(transitions = NULL, compartments = NULL, pars = NULL,  
  obsProcess = NULL, addVars = NULL, stopCrit = NULL,  
  tspan = FALSE, incidence = FALSE, afterTstar = NULL, PF = FALSE,  
  runFromR = TRUE)
```

**Arguments**

transitions	character vector containing transitions on the form "X -> ... -> Y". The left (right) side is the initial (final) state and the propensity is written in between the ->-signs. The special symbol @ is reserved for the empty set. For example, transitions = c("S -> k1*S*I -> I", "I -> k2*I -> R") expresses a SIR model.
compartments	contains the names of the involved compartments, for example, compartments = c("S", "I", "R").
pars	a character vector containing the names of the parameters.
obsProcess	data.frame determining the observation process. Columns must be in the order: dataNames, dist, p1, p2. dataNames is a character denoting the name of the variable that will be output from the observation process; dist is a character specifying the distribution of the observation process (must be one of "unif", "pois", "norm" or "binom" at the current time); p1 is the first parameter (the lower bound in the case of "unif", the rate in the case of "pois", the mean in the case of "norm" or the size in the case of "binom"); and finally p2 is the second parameter (the upper bound in the case of "unif", NA in the case of "pois", the standard deviation in the case of "norm", and prob in the case of "binom").
addVars	a character vector where the names specify the additional variables to be added to the function call. These can be used to specify variables that can be used for e.g. additional stopping criteria.
stopCrit	A character vector including additional stopping criteria for rejecting simulations early. These will be inserted within <code>if(CRIT){out[0] = 0; return out;}</code> statements within the underlying Rcpp code, which a return value of 0 corresponds to rejecting the simulation. Variables in CRIT must match either those in compartments and/or addVars.
tspan	A logical determining whether to return time series counts or not.
incidence	A logical specifying whether to return incidence curves in addition to counts.
afterTstar	A character containing code to insert after each new event time is generated.
PF	A logical determining whether to compile the code for use in a particle filter.
runFromR	logical determining whether code is to be compiled to run directly in R, or whether to be compiled as an XPtr object for use in Rcpp.

**Details**

Uses a SimInf style markup to create compartmental state-space written using Rcpp. A helper run function exists to compile and run the model. Another helper function, compileRcpp, can compile the model to produce a function that can be run directly from R, or compiled into an external pointer (using the RcppXPtrUtils package).

**Value**

An object of class SimBIID\_model, which is essentially a list containing elements:

- code: parsed code to compile;

- transitions: copy of transitions argument;
- compartments: copy of compartments argument;
- pars: copy of pars argument;
- obsProcess: copy of obsProcess argument;
- stopCrit: copy of stopCrit argument;
- addVars: copy of addVars argument;
- tspan: copy of tspan argument;
- incidence: copy of incidence argument;
- afterTstar: copy of afterTstar argument;
- PF: copy of PF argument;
- runFromR: copy of runFromR argument.

This can be compiled into an XPtr or function object using `compileRcpp()`.

### See Also

[run](#), [compileRcpp](#), [print.SimBIID\\_model](#)

### Examples

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)

## compile and run model
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0)
)
sims
```

---

plot.ABCSMC	<i>Plots ABCSMC objects</i>
-------------	-----------------------------

---

**Description**

Plot method for ABCSMC objects.

**Usage**

```
## S3 method for class 'ABCSMC'
plot(x, type = c("post", "output"), gen = NA,
     joint = FALSE, transfunc = NA, ...)
```

**Arguments**

x	An ABCSMC object.
type	Takes the value "post" if you want to plot posterior distributions. Takes the value "output" if you want to plot the simulated outputs.
gen	A vector of generations to plot. If left missing then defaults to all generations.
joint	A logical describing whether joint or marginal distributions are wanted.
transfunc	Is a function object where the arguments to the function must match all or a subset of the parameters in the model. This function needs to return a <code>data.frame</code> object with columns containing the transformed parameters.
...	Not used here.

**Value**

A plot of the ABC posterior distributions for different generations, or the distributions of the simulated summary measures for different generations.

**See Also**

[ABCSMC](#), [print.ABCSMC](#), [summary.ABCSMC](#)

**Examples**

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
```

```

    compartments = compartments,
    pars = pars
  )
model <- compileRcpp(model)

## generate function to run simulators
## and return summary statistics
simSIR <- function(pars, data, tols, u, model) {

  ## run model
  sims <- model(pars, 0, data[2] + tols[2], u)

  ## this returns a vector of the form:
  ## completed (1/0), t, S, I, R (here)
  if(sims[1] == 0) {
    ## if simulation rejected
    return(NA)
  } else {
    ## extract finaltime and finalsize
    finaltime <- sims[2]
    finalsize <- sims[5]
  }

  ## return vector if match, else return NA
  if(all(abs(c(finalsize, finaltime) - data) <= tols)){
    return(c(finalsize, finaltime))
  } else {
    return(NA)
  }
}

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma"),
  dist = rep("gamma", 2),
  stringsAsFactors = FALSE
)
priors$p1 <- c(10, 10)
priors$p2 <- c(10^4, 10^2)

## define the targeted summary statistics
data <- c(
  finalsize = 30,
  finaltime = 76
)

## set initial states (1 initial infection
## in population of 120)
iniStates <- c(S = 119, I = 1, R = 0)

## set initial tolerances
tol <- c(
  finalsize = 50,

```

```
    finaltime = 50
  )

  ## run 2 generations of ABC-SMC
  ## setting tolerance to be 50th
  ## percentile of the accepted
  ## tolerances at each generation
  post <- ABCSMC(
    x = data,
    priors = priors,
    func = simSIR,
    u = iniStates,
    tols = tols,
    ptol = 0.2,
    ngen = 2,
    npart = 50,
    model = model
  )
  post

  ## run one further generation
  post <- ABCSMC(post, ptols = 0.5, ngen = 1)
  post
  summary(post)

  ## plot posteriors
  plot(post)

  ## plot outputs
  plot(post, "output")
```

---

plot.PMCMC

*Plots PMCMC objects*

---

## Description

Plot method for PMCMC objects.

## Usage

```
## S3 method for class 'PMCMC'
plot(x, type = c("post", "trace"), joint = FALSE,
     transfunc = NA, ask = TRUE, ...)
```

## Arguments

x                    A PMCMC object.

type	Takes the value "post" if you want to plot posterior distributions. Takes the value "trace" if you want to plot the trace plots.
joint	A logical describing whether joint or marginal distributions are wanted.
transfunc	Is a function object where the arguments to the function must match all or a subset of the parameters in the model. This function needs to return a <code>data.frame</code> object with columns containing the transformed parameters.
ask	Should the user ask before moving onto next trace plot.
...	Not used here.

### Value

A plot of the (approximate) posterior distributions obtained from fitting a particle Markov chain Monte Carlo algorithm, or provides corresponding trace plots.

### See Also

[PMCMC](#), [print.PMCMC](#), [predict.PMCMC](#), [summary.PMCMC](#) [window.PMCMC](#)

### Examples

```
## set up data to pass to PMCMC
flu_dat <- data.frame(
  t = 1:14,
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)
)

## set up observation process
obs <- data.frame(
  dataNames = "Robs",
  dist = "pois",
  p1 = "R + 1e-5",
  p2 = NA,
  stringsAsFactors = FALSE
)

## set up model (no need to specify tspan
## argument as it is set in PMCMC())
transitions <- c(
  "S -> beta * S * I / (S + I + R + R1) -> I",
  "I -> gamma * I -> R",
  "R -> gamma1 * R -> R1"
)

compartments <- c("S", "I", "R", "R1")
pars <- c("beta", "gamma", "gamma1")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  obsProcess = obs
)
```



```
)

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma", "gamma1"),
  dist = rep("unif", 3),
  stringsAsFactors = FALSE)
priors$p1 <- c(0, 0, 0)
priors$p2 <- c(5, 5, 5)

## define initial states
iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

set.seed(50)

## run PMCMC algorithm
post <- PMCMC(
  x = flu_dat,
  priors = priors,
  func = model,
  u = iniStates,
  npart = 25,
  niter = 5000,
  nprintsum = 1000
)

## plot MCMC traces
plot(post, "trace")

## continue for some more iterations
post <- PMCMC(post, niter = 5000, nprintsum = 1000)

## plot traces and posteriors
plot(post, "trace")
plot(post)

## remove burn-in
post <- window(post, start = 5000)

## summarise posteriors
summary(post)
```

---

plot.SimBIID\_runs

*Plots SimBIID\_runs objects*

---

### **Description**

Plot method for SimBIID\_runs objects.

**Usage**

```
## S3 method for class 'SimBIID_runs'
plot(x, which = c("all", "t"), type = c("runs",
    "sums"), rep = NA, quant = 0.9, data = NULL, matchData = NULL,
    ...)
```

**Arguments**

x	An SimBIID_runs object.
which	A character vector of states to plot. Can be "all" to plot all states (and final event times), or "t" to plot final event times.
type	Character stating whether to plot full simulations over time ("runs") or summaries ("sums").
rep	An integer vector of simulation runs to plot.
quant	A vector of quantiles (> 0.5) to plot if type == "runs".
data	A data.frame containing time series count data, with the first column called t, followed by columns of time-series counts.
matchData	A character vector containing matches between the columns of data and the columns of the model runs. Each entry must be of the form e.g. "SD = SR", where SD is the name of the column in data, and SR is the name of the column in x.
...	Not used here.

**Value**

A plot of individual simulations and/or summaries of repeated simulations extracted from SimBIID\_runs object.

**See Also**

[mparseRcpp](#), [print.SimBIID\\_runs](#), [run](#)

**Examples**

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  tspan = TRUE
)
```

```

## run 100 replicate simulations and
## plot outputs
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0),
  tspan = seq(1, 100, length.out = 10),
  nrep = 100
)
plot(sims, quant = c(0.55, 0.75, 0.9))

## add replicate 1 to plot
plot(sims, quant = c(0.55, 0.75, 0.9), rep = 1)

```

---

PMCMC

*Runs particle MCMC algorithm*


---

### Description

Runs particle Markov chain Monte Carlo (PMCMC) algorithm using a bootstrap particle filter for fitting infectious disease models to time series count data.

### Usage

```

PMCMC(x, ...)

## S3 method for class 'PMCMC'
PMCMC(x, niter = 1000, nprintsum = 100, adapt = TRUE,
      adaptmixprop = 0.05, nupdate = 100, ...)

## Default S3 method:
PMCMC(x, priors, func, u, npart = 100, iniPars = NA,
      fixpars = FALSE, niter = 1000, nprintsum = 100, adapt = TRUE,
      propVar = NA, adaptmixprop = 0.05, nupdate = 100, ...)

```

### Arguments

x	A PMCMC object, or a data.frame containing time series count data, with the first column called t, followed by columns of time-series counts. The time-series counts columns must be in the order of the 'counts' object in the 'func' function (see below).
...	Not used here.
niter	An integer specifying the number of iterations to run the MCMC.

<code>nprintsum</code>	Prints summary of MCMC to screen every <code>nprintsum</code> iterations. Also defines how often adaptive scaling of proposal variances occur.
<code>adapt</code>	Logical determining whether to use adaptive proposal or not.
<code>adaptmixprop</code>	Mixing proportion for adaptive proposal.
<code>nupdate</code>	Controls when to start adaptive update.
<code>priors</code>	A <code>data.frame</code> containing columns <code>parnames</code> , <code>dist</code> , <code>p1</code> and <code>p2</code> , with number of rows equal to the number of parameters. The column <code>parname</code> simply gives names to each parameter for plotting and summarising. Each entry in the <code>dist</code> column must contain one of <code>c("unif", "norm", "gamma")</code> , and the corresponding <code>p1</code> and <code>p2</code> entries relate to the hyperparameters (lower and upper bounds in the uniform case; mean and standard deviation in the normal case; and shape and rate in the gamma case).
<code>func</code>	A <code>SimBIID_model</code> object or an <code>XPtr</code> to simulation function. If the latter, then this function must take the following arguments in order: <ul style="list-style-type: none"> <li>• <code>NumericVector pars</code>: a vector of parameters;</li> <li>• <code>double tstart</code>: the start time;</li> <li>• <code>double tstop</code>: the end time;</li> <li>• <code>IntegerVector u</code>: a vector of states at time <code>tstart</code>;</li> <li>• <code>IntegerVector counts</code>: a vector of observed counts at <code>tstop</code>.</li> </ul>
<code>u</code>	A named vector of initial states.
<code>npart</code>	An integer specifying the number of particles for the bootstrap particle filter.
<code>iniPars</code>	A named vector of initial values for the parameters of the model. If left unspecified, then these are sampled from the prior distribution(s).
<code>fixpars</code>	A logical determining whether to fix the input parameters (useful for determining the variance of the marginal likelihood estimates).
<code>propVar</code>	A numeric ( <code>npars x npars</code> ) matrix with log (or logistic) covariances to use as (initial) proposal matrix. If left unspecified then defaults to <code>diag(nrow(priors)) * (0.1 ^ 2) / nrow(priors)</code> .

### Details

Function runs a particle MCMC algorithm using a bootstrap particle filter for a given model. If running with `fixpars = TRUE` then this runs `niter` simulations using fixed parameter values. This can be used to optimise the number of particles after a training run. Also has `print()`, `summary()`, `plot()`, `predict()` and `window()` methods.

### Value

If the code throws an error, then it returns a missing value (`NA`). If `fixpars = TRUE` it returns a list of length 2 containing:

- `output`: a matrix with two columns. The first contains the simulated log-likelihood, and the second is a binary indicator relating to whether the simulation was 'skipped' or not (1 = skipped, 0 = not skipped);
- `pars`: a vector of parameters used for the simulations.

If `fixpars = FALSE`, the routine returns a PMCMC object, essentially a list containing:

- `pars`: an mcmc object containing posterior samples for the parameters;
- `u`: a copy of the `u` input;
- `accrate`: the cumulative acceptance rate;
- `npart`: the chosen number of particles;
- `time`: the time taken to run the routine (in seconds);
- `propVar`: the proposal covariance for the parameter updates;
- `data`: a copy of the `x` input;
- `priors`: a copy of the priors input;
- `func`: a copy of the `func` input.

## References

Andrieu C, Doucet A and Holenstein R (2010) <doi:10.1111/j.1467-9868.2009.00736.x>

## See Also

[print.PMCMC](#), [plot.PMCMC](#), [predict.PMCMC](#), [summary.PMCMC](#) [window.PMCMC](#)

## Examples

```
## set up data to pass to PMCMC
flu_dat <- data.frame(
  t = 1:14,
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)
)

## set up observation process
obs <- data.frame(
  dataNames = "Robs",
  dist = "pois",
  p1 = "R + 1e-5",
  p2 = NA,
  stringsAsFactors = FALSE
)

## set up model (no need to specify tspan
## argument as it is set in PMCMC())
transitions <- c(
  "S -> beta * S * I / (S + I + R + R1) -> I",
  "I -> gamma * I -> R",
  "R -> gamma1 * R -> R1"
)

compartments <- c("S", "I", "R", "R1")
pars <- c("beta", "gamma", "gamma1")
model <- mparseRcpp(
  transitions = transitions,
```

```
    compartments = compartments,
    pars = pars,
    obsProcess = obs
  )

  ## set priors
  priors <- data.frame(
    parnames = c("beta", "gamma", "gamma1"),
    dist = rep("unif", 3),
    stringsAsFactors = FALSE)
  priors$p1 <- c(0, 0, 0)
  priors$p2 <- c(5, 5, 5)

  ## define initial states
  iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

  set.seed(50)

  ## run PMCMC algorithm
  post <- PMCMC(
    x = flu_dat,
    priors = priors,
    func = model,
    u = iniStates,
    npart = 25,
    niter = 5000,
    nprintsum = 1000
  )

  ## plot MCMC traces
  plot(post, "trace")

  ## continue for some more iterations
  post <- PMCMC(post, niter = 5000, nprintsum = 1000)

  ## plot traces and posteriors
  plot(post, "trace")
  plot(post)

  ## remove burn-in
  post <- window(post, start = 5000)

  ## summarise posteriors
  summary(post)
```

**Description**

Predict method for PMCMC objects.

**Usage**

```
## S3 method for class 'PMCMC'
predict(object, tspan, npart = 50, ...)
```

**Arguments**

object	A PMCMC object.
tspan	A vector of times over which to output predictions.
npart	The number of particles to use in the bootstrap filter.
...	Not used here.

**Value**

A SimBIID\_runs object.

**See Also**

[PMCMC](#), [print.PMCMC](#), [plot.PMCMC](#), [summary.PMCMC](#) [window.PMCMC](#)

**Examples**

```
## set up data to pass to PMCMC
flu_dat <- data.frame(
  t = 1:14,
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)
)

## set up observation process
obs <- data.frame(
  dataNames = "Robs",
  dist = "pois",
  p1 = "R + 1e-5",
  p2 = NA,
  stringsAsFactors = FALSE
)

## set up model (no need to specify tspan
## argument as it is set in PMCMC())
transitions <- c(
  "S -> beta * S * I / (S + I + R + R1) -> I",
  "I -> gamma * I -> R",
  "R -> gamma1 * R -> R1"
)

compartments <- c("S", "I", "R", "R1")
pars <- c("beta", "gamma", "gamma1")
```

```

model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  obsProcess = obs
)

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma", "gamma1"),
  dist = rep("unif", 3),
  stringsAsFactors = FALSE)
priors$p1 <- c(0, 0, 0)
priors$p2 <- c(5, 5, 5)

## define initial states
iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

## run PMCMC algorithm for first three days of data
post <- PMCMC(
  x = flu_dat[1:3, ],
  priors = priors,
  func = model,
  u = iniStates,
  npart = 75,
  niter = 10000,
  nprintsum = 1000
)

## plot traces
plot(post, "trace")

## run predictions forward in time
post_pred <- predict(
  window(post, start = 2000, thin = 8),
  tspan = 4:14
)

## plot predictions
plot(post_pred, quant = c(0.6, 0.75, 0.95))

```

---

```
print.ABCSMC
```

```
Prints ABCSMC objects
```

---

### **Description**

Print method for ABCSMC objects.



**Usage**

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

**Arguments**

```
x          An ABCSMC object.  
...        Not used here.
```

**Value**

Summary outputs printed to the screen.

**See Also**

[ABCSMC](#), [plot.ABCSMC](#), [summary.ABCSMC](#)

**Examples**

```
## set up SIR simulationmodel  
transitions <- c(  
  "S -> beta * S * I -> I",  
  "I -> gamma * I -> R"  
)  
compartments <- c("S", "I", "R")  
pars <- c("beta", "gamma")  
model <- mparseRcpp(  
  transitions = transitions,  
  compartments = compartments,  
  pars = pars  
)  
model <- compileRcpp(model)  
  
## generate function to run simulators  
## and return summary statistics  
simSIR <- function(pars, data, tols, u, model) {  
  
  ## run model  
  sims <- model(pars, 0, data[2] + tols[2], u)  
  
  ## this returns a vector of the form:  
  ## completed (1/0), t, S, I, R (here)  
  if(sims[1] == 0) {  
    ## if simulation rejected  
    return(NA)  
  } else {  
    ## extract finaltime and finalsize  
    finaltime <- sims[2]  
    finalsize <- sims[5]  
  }  
}
```

```

    ## return vector if match, else return NA
    if(all(abs(c(finalsize, finaltime) - data) <= tols)){
      return(c(finalsize, finaltime))
    } else {
      return(NA)
    }
  }

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma"),
  dist = rep("gamma", 2),
  stringsAsFactors = FALSE
)
priors$p1 <- c(10, 10)
priors$p2 <- c(10^4, 10^2)

## define the targeted summary statistics
data <- c(
  finalsize = 30,
  finaltime = 76
)

## set initial states (1 initial infection
## in population of 120)
iniStates <- c(S = 119, I = 1, R = 0)

## set initial tolerances
tols <- c(
  finalsize = 50,
  finaltime = 50
)

## run 2 generations of ABC-SMC
## setting tolerance to be 50th
## percentile of the accepted
## tolerances at each generation
post <- ABCSMC(
  x = data,
  priors = priors,
  func = simSIR,
  u = iniStates,
  tols = tols,
  ptol = 0.2,
  ngen = 2,
  npart = 50,
  model = model
)
post

## run one further generation
post <- ABCSMC(post, ptols = 0.5, ngen = 1)

```

```
post
summary(post)

## plot posteriors
plot(post)

## plot outputs
plot(post, "output")
```

---

print.PMCMC	<i>Prints PMCMC objects</i>
-------------	-----------------------------

---

## Description

Print method for PMCMC objects.

## Usage

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

## Arguments

x	A PMCMC object.
...	Not used here.

## Value

Summary outputs printed to the screen.

## See Also

[PMCMC](#), [plot.PMCMC](#), [predict.PMCMC](#), [summary.PMCMC](#) [window.PMCMC](#)

## Examples

```
## set up data to pass to PMCMC
flu_dat <- data.frame(
  t = 1:14,
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)
)

## set up observation process
obs <- data.frame(
  dataNames = "Robs",
  dist = "pois",
```

```

    p1 = "R + 1e-5",
    p2 = NA,
    stringsAsFactors = FALSE
  )

  ## set up model (no need to specify tspan
  ## argument as it is set in PMCMC())
  transitions <- c(
    "S -> beta * S * I / (S + I + R + R1) -> I",
    "I -> gamma * I -> R",
    "R -> gamma1 * R -> R1"
  )
  compartments <- c("S", "I", "R", "R1")
  pars <- c("beta", "gamma", "gamma1")
  model <- mparseRcpp(
    transitions = transitions,
    compartments = compartments,
    pars = pars,
    obsProcess = obs
  )

  ## set priors
  priors <- data.frame(
    parnames = c("beta", "gamma", "gamma1"),
    dist = rep("unif", 3),
    stringsAsFactors = FALSE)
  priors$p1 <- c(0, 0, 0)
  priors$p2 <- c(5, 5, 5)

  ## define initial states
  iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

  set.seed(50)

  ## run PMCMC algorithm
  post <- PMCMC(
    x = flu_dat,
    priors = priors,
    func = model,
    u = iniStates,
    npart = 25,
    niter = 5000,
    nprintsum = 1000
  )

  ## plot MCMC traces
  plot(post, "trace")

  ## continue for some more iterations
  post <- PMCMC(post, niter = 5000, nprintsum = 1000)

  ## plot traces and posteriors
  plot(post, "trace")

```

```
plot(post)

## remove burn-in
post <- window(post, start = 5000)

## summarise posteriors
summary(post)
```

---

`print.SimBIID_model`     *Prints SimBIID\_model objects*

---

### **Description**

Print method for `SimBIID_model` objects.

### **Usage**

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

### **Arguments**

`x`                    A `SimBIID_model` object.  
`...`                 Not used here.

### **Value**

Prints parsed Rcpp code to the screen.

---

`print.SimBIID_runs`     *Prints SimBIID\_runs objects*

---

### **Description**

Print method for `SimBIID_runs` objects.

### **Usage**

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

### **Arguments**

`x`                    A `SimBIID_runs` object.  
`...`                 Not used here.

**Value**

Summary outputs printed to the screen.

**See Also**

[mparseRcpp](#), [plot.SimBIID\\_runs](#), [run](#)

**Examples**

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  tspan = TRUE
)

## run 100 replicate simulations and
## plot outputs
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0),
  tspan = seq(1, 100, length.out = 10),
  nrep = 100
)
sims
```

---

run

*Runs SimBIID\_model object*

---

**Description**

Wrapper function that compiles (if necessary) and runs a `SimBIID_model` object. Returns results in a user-friendly manner as a `SimBIID_run` object, for which `print()` and `plot()` generics are provided.

**Usage**

```
run(model, pars, tstart, tstop, u, tspan, nrep = 1, parallel = FALSE,
     mc.cores = NA)
```

**Arguments**

model	An object of class <code>SimBIID_model</code> .
pars	A named vector of parameters.
tstart	The time at which to start the simulation.
tstop	The time at which to stop the simulation.
u	A named vector of initial states.
tspan	A numeric vector containing the times at which to save the states of the system.
nrep	Specifies the number of simulations to run.
parallel	A logical determining whether to use parallel processing or not.
mc.cores	Number of cores to use if using parallel processing.

**Value**

An object of class `SimBIID_run`, essentially a list containing elements:

- `sums`: a `data.frame()` with summaries of the model runs. This includes columns `run`, `completed`, `t`, `u*` (see help file for `SimBIID_model` for more details);
- `runs`: a `data.frame()` object, containing columns: `run`, `t`, `u*` (see help file for `SimBIID_model` for more details). These contain time series counts for the simulations. Note that this will only be returned if `tspan = TRUE` in the original `SimBIID_model` object.
- `bootEnd`: a time point denoting when bootstrapped estimates end and predictions begin (for `predict.PMCMC()` method).

**See Also**

[mparseRcpp](#), [print.SimBIID\\_runs](#), [plot.SimBIID\\_runs](#)

**Examples**

```
## set up SIR simulation model
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)
```

```
## compile and run model
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0)
)
sims

## add tspan option to return
## time series counts at different
## time points
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  tspan = TRUE
)
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0),
  tspan = seq(1, 100, length.out = 10)
)
sims

## run 100 replicate simulations and
## plot outputs
sims <- run(
  model = model,
  pars = c(beta = 0.001, gamma = 0.1),
  tstart = 0,
  tstop = 100,
  u = c(S = 119, I = 1, R = 0),
  tspan = seq(1, 100, length.out = 10),
  nrep = 100
)
sims
plot(sims)
```



**Description**

A dataset containing time series counts for the number of new removals for the 1967 Abakaliki smallpox outbreak.

**Usage**

```
smallpox
```

**Format**

A data frame with 23 rows and 2 variables:

**time** days from initial observed removal

**removals** number of new removals in (time - 1, time)

**Source**

Thompson D and Foege W (1968) <[https://apps.who.int/iris/bitstream/handle/10665/67462/WHO\\_SE\\_68.3.pdf](https://apps.who.int/iris/bitstream/handle/10665/67462/WHO_SE_68.3.pdf)>

---

summary.ABCSMC	<i>Summarises ABCSMC objects</i>
----------------	----------------------------------

---

**Description**

Summary method for ABCSMC objects.

**Usage**

```
## S3 method for class 'ABCSMC'
summary(object, gen = NA, transfunc = NA, ...)
```

**Arguments**

object	An ABCSMC object.
gen	The generation of ABC that you want to extract. If left missing then defaults to final generation.
transfunc	Is a function object where the arguments to the function must match all or a subset of the parameters in the model. This function needs to return a <code>data.frame</code> object with columns containing the transformed parameters.
...	Not used here.

**Value**

A `data.frame` with weighted posterior means and variances.

**See Also**

[ABCSCM](#), [print.ABCSCM](#), [plot.ABCSCM](#)

**Examples**

```
## set up SIR simulationmodel
transitions <- c(
  "S -> beta * S * I -> I",
  "I -> gamma * I -> R"
)
compartments <- c("S", "I", "R")
pars <- c("beta", "gamma")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars
)
model <- compileRcpp(model)

## generate function to run simulators
## and return summary statistics
simSIR <- function(pars, data, tols, u, model) {

  ## run model
  sims <- model(pars, 0, data[2] + tols[2], u)

  ## this returns a vector of the form:
  ## completed (1/0), t, S, I, R (here)
  if(sims[1] == 0) {
    ## if simulation rejected
    return(NA)
  } else {
    ## extract finaltime and finalsize
    finaltime <- sims[2]
    finalsize <- sims[5]
  }

  ## return vector if match, else return NA
  if(all(abs(c(finalsize, finaltime) - data) <= tols)){
    return(c(finalsize, finaltime))
  } else {
    return(NA)
  }
}

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma"),
  dist = rep("gamma", 2),
  stringsAsFactors = FALSE
)
```

```
priors$p1 <- c(10, 10)
priors$p2 <- c(10^4, 10^2)

## define the targeted summary statistics
data <- c(
  finalsize = 30,
  finaltime = 76
)

## set initial states (1 initial infection
## in population of 120)
iniStates <- c(S = 119, I = 1, R = 0)

## set initial tolerances
tols <- c(
  finalsize = 50,
  finaltime = 50
)

## run 2 generations of ABC-SMC
## setting tolerance to be 50th
## percentile of the accepted
## tolerances at each generation
post <- ABCSMC(
  x = data,
  priors = priors,
  func = simSIR,
  u = iniStates,
  tols = tols,
  ptol = 0.2,
  ngen = 2,
  npart = 50,
  model = model
)
post

## run one further generation
post <- ABCSMC(post, ptols = 0.5, ngen = 1)
post
summary(post)

## plot posteriors
plot(post)

## plot outputs
plot(post, "output")
```

**Description**

Summary method for PMCMC objects.

**Usage**

```
## S3 method for class 'PMCMC'
summary(object, transfunc = NA, ...)
```

**Arguments**

object	A PMCMC object.
transfunc	Is a function object where the arguments to the function must match all or a subset of the parameters in the model. This function needs to return a <code>data.frame</code> object with columns containing the transformed parameters.
...	Not used here.

**Value**

A `summary.mcmc` object.

**See Also**

[PMCMC](#), [print.PMCMC](#), [predict.PMCMC](#), [plot.PMCMC](#) [window.PMCMC](#)

**Examples**

```
## set up data to pass to PMCMC
flu_dat <- data.frame(
  t = 1:14,
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)
)

## set up observation process
obs <- data.frame(
  dataNames = "Robs",
  dist = "pois",
  p1 = "R + 1e-5",
  p2 = NA,
  stringsAsFactors = FALSE
)

## set up model (no need to specify tspan
## argument as it is set in PMCMC())
transitions <- c(
  "S -> beta * S * I / (S + I + R + R1) -> I",
  "I -> gamma * I -> R",
  "R -> gamma1 * R -> R1"
)
compartments <- c("S", "I", "R", "R1")
```

```
pars <- c("beta", "gamma", "gamma1")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  obsProcess = obs
)

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma", "gamma1"),
  dist = rep("unif", 3),
  stringsAsFactors = FALSE)
priors$p1 <- c(0, 0, 0)
priors$p2 <- c(5, 5, 5)

## define initial states
iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

set.seed(50)

## run PMCMC algorithm
post <- PMCMC(
  x = flu_dat,
  priors = priors,
  func = model,
  u = iniStates,
  npart = 25,
  niter = 5000,
  nprintsum = 1000
)

## plot MCMC traces
plot(post, "trace")

## continue for some more iterations
post <- PMCMC(post, niter = 5000, nprintsum = 1000)

## plot traces and posteriors
plot(post, "trace")
plot(post)

## remove burn-in
post <- window(post, start = 5000)

## summarise posteriors
summary(post)
```

**Description**

window method for class PMCMC.

**Usage**

```
## S3 method for class 'PMCMC'  
window(x, ...)
```

**Arguments**

x                    a PMCMC object, usually as a result of a call to PMCMC.  
...                   arguments to pass to [window.mcmc](#)

**Details**

Acts as a wrapper function for [window.mcmc](#) from the coda package

**Value**

a PMCMC object

**See Also**

[PMCMC](#), [print.PMCMC](#), [predict.PMCMC](#), [summary.PMCMC](#) [plot.PMCMC](#)

**Examples**

```
## set up data to pass to PMCMC  
flu_dat <- data.frame(  
  t = 1:14,  
  Robs = c(3, 8, 26, 76, 225, 298, 258, 233, 189, 128, 68, 29, 14, 4)  
)  
  
## set up observation process  
obs <- data.frame(  
  dataNames = "Robs",  
  dist = "pois",  
  p1 = "R + 1e-5",  
  p2 = NA,  
  stringsAsFactors = FALSE  
)  
  
## set up model (no need to specify tspan  
## argument as it is set in PMCMC())  
transitions <- c(  
  "S -> beta * S * I / (S + I + R + R1) -> I",  
  "I -> gamma * I -> R",  
  "R -> gamma1 * R -> R1"  
)  
compartments <- c("S", "I", "R", "R1")
```

```
pars <- c("beta", "gamma", "gamma1")
model <- mparseRcpp(
  transitions = transitions,
  compartments = compartments,
  pars = pars,
  obsProcess = obs
)

## set priors
priors <- data.frame(
  parnames = c("beta", "gamma", "gamma1"),
  dist = rep("unif", 3),
  stringsAsFactors = FALSE)
priors$p1 <- c(0, 0, 0)
priors$p2 <- c(5, 5, 5)

## define initial states
iniStates <- c(S = 762, I = 1, R = 0, R1 = 0)

set.seed(50)

## run PMCMC algorithm
post <- PMCMC(
  x = flu_dat,
  priors = priors,
  func = model,
  u = iniStates,
  npart = 25,
  niter = 5000,
  nprintsum = 1000
)

## plot MCMC traces
plot(post, "trace")

## continue for some more iterations
post <- PMCMC(post, niter = 5000, nprintsum = 1000)

## plot traces and posteriors
plot(post, "trace")
plot(post)

## remove burn-in
post <- window(post, start = 5000)

## summarise posteriors
summary(post)
```

# Index

## \*Topic **datasets**

ebola, [10](#)

smallpox, [32](#)

ABCRef, [3](#)

ABCSCMC, [3](#), [5](#), [13](#), [25](#), [34](#)

compileRcpp, [8](#), [12](#)

ebola, [10](#)

mparseRcpp, [9](#), [10](#), [18](#), [30](#), [31](#)

plot.ABCSCMC, [7](#), [13](#), [25](#), [34](#)

plot.PMCMC, [15](#), [21](#), [23](#), [27](#), [36](#), [38](#)

plot.SimBIID\_runs, [17](#), [30](#), [31](#)

PMCMC, [16](#), [19](#), [23](#), [27](#), [36](#), [38](#)

predict.PMCMC, [16](#), [21](#), [22](#), [27](#), [36](#), [38](#)

print.ABCSCMC, [7](#), [13](#), [24](#), [34](#)

print.PMCMC, [16](#), [21](#), [23](#), [27](#), [36](#), [38](#)

print.SimBIID\_model, [12](#), [29](#)

print.SimBIID\_runs, [18](#), [29](#), [31](#)

run, [12](#), [18](#), [30](#), [30](#)

SimBIID-package, [2](#)

smallpox, [32](#)

summary.ABCSCMC, [7](#), [13](#), [25](#), [33](#)

summary.PMCMC, [16](#), [21](#), [23](#), [27](#), [35](#), [38](#)

window.mcmc, [38](#)

window.PMCMC, [16](#), [21](#), [23](#), [27](#), [36](#), [37](#)