Package 'phylopomp'

April 24, 2024

R topics documented:

---

as.data.frame   *Coerce to a Data Frame*

Description

Functions to coerce an object to a data frame.

1

Usage

```
## S3 method for class 'gplin'
as.data.frame(x, ...)
```

Arguments

x	any R object.
...	additional arguments to be passed to or from methods.

Details

An object of class 'gplin' is coerced to a data frame by means of as.data.frame.

---

curtail	*Curtail a genealogy to the given time*

Description

Discards all nodes beyond the given time.

Usage

```
curtail(object, time = NA, prune = TRUE, obscure = TRUE)
```

Arguments

object	gpsim object.
time	logical; return the current time?
prune	logical; prune the genealogy?
obscure	logical; obscure the demes?

Value

A curtailed genealogy object.

Examples

```
library(ggplot2)

simulate("SIIR",time=5) -> x

plot_grid(
  x |>
    plot(prune=FALSE,points=TRUE),
  x |>
    curtail(time=3) |>
    plot(prune=FALSE,points=TRUE)+
```

diagram 3

```
      expand_limits(x=5),
    ncol=1,align="h",axis="tblr"
  )

  plot_grid(
    x |>
      plot(prune=TRUE,points=TRUE)+
      geom_vline(xintercept=3),
    x |> curtail(time=3) |>
      plot(prune=TRUE,points=TRUE)+
      geom_vline(xintercept=3)+
      expand_limits(x=5),
    ncol=1,align="h",axis="tblr"
  )
```

---

diagram   *Genealogy process diagram*

Description

Produces a diagram of the genealogy process state.

Usage

```
diagram(
  object,
  prune = TRUE,
  obscure = TRUE,
  m = NULL,
  n = NULL,
  ...,
  digits = 1,
  palette = scales::hue_pal(l = 80, c = 20, h = c(220, 580))
)

## S3 method for class 'gpdiag'
print(x, newpage = is.null(vp), vp = NULL, ...)
```

Arguments

object	gpsim object.
prune	logical; prune the genealogy?
obscure	logical; obscure the demes?
m	width of plotting window, in nodes. By default, the nodes will be adjusted in width to fit the window.
n	height of the pockets, in balls. By default, the balls will be adjusted in size to fit the space available.

`...`	other arguments, ignored.
`digits`	non-negative integer; number of decimal digits to print in the node time
`palette`	color palette for indicating demes. This can be furnished either as a function or a vector of colors. If this is a function, it should take a single integer argument, the number of colors required. If it is a vector, it should have at least as many elements as there are demes in the genealogy.
`x`	An R object.
`newpage`	draw new empty page first?
`vp`	viewport to draw plot in

Value

A **grid** graphics object (`grob`), invisibly.

Examples

```
runSIR(Beta=3,gamma=0.1,psi=0.2,S0=100,I0=5,R0=0,time=2,t0=0) -> x
plot(x,points=TRUE,prune=FALSE)
plot_grid(plotlist=list(plot(x,points=TRUE)[[1]],diagram(x)),
  ncol=1,rel_heights=c(4,1))
```

---

geneal	*Bare genealogy*

Description

Extracts the bare genealogy from a Markov genealogy process simulation

Usage

```
geneal(object)
```

Arguments

`object`	a 'gpgen' object.

Value

A bare genealogy object.

```
genealogy diagram internals
```
Diagramming internals

Description

Facilities to produce diagrammatic representations of genealogy process states.

Usage

```
genealogyGrob(object, m = NULL, n = NULL, vp = NULL, palette, ...)

nodeGrob(object, digits = 1, palette, n = NULL, vp = NULL)

pocketGrob(object, n, vp = NULL)

ballGrob(object, vp = NULL)

resizingTextGrob(..., vp = NULL)

## S3 method for class 'resizingTextGrob'
drawDetails(x, recording = TRUE)

## S3 method for class 'resizingTextGrob'
preDrawDetails(x)

## S3 method for class 'resizingTextGrob'
postDrawDetails(x)

## S3 method for class 'ballGrob'
drawDetails(x, recording = TRUE)

## S3 method for class 'ballGrob'
preDrawDetails(x)

## S3 method for class 'ballGrob'
postDrawDetails(x)

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

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

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

Arguments

`object`	list; pocket structure
`m`	width of plotting window, in nodes. By default, the nodes will be adjusted in width to fit the window.
`n`	length of longest genealogy
`vp`	viewport to draw plot in
`palette`	color palette for indicating demes. This can be furnished either as a function or a vector of colors. If this is a function, it should take a single integer argument, the number of colors required. If it is a vector, it should have at least as many elements as there are demes in the genealogy.
`...`	arguments to be passed to [`textGrob`].
`digits`	non-negative integer; number of decimal digits to print in the node time
`x`	An R object.
`recording`	A logical value indicating whether a grob is being added to the display list or redrawn from the display list.

Details

Code for the resizing text adapted from a blog post by Mark Heckmann (https://ryouready.wordpress.com/2012/08/01/creating-a-text-grob-that-automatically-adjusts-to-viewport-size/).

---

getInfo	*getInfo*

Description

Retrieve information from genealogy process simulation

Usage

```
getInfo(
  object,
  prune = TRUE,
  obscure = TRUE,
  t0 = FALSE,
  time = FALSE,
  description = FALSE,
  structure = FALSE,
  yaml = FALSE,
  ndeme = FALSE,
  lineages = FALSE,
  newick = FALSE,
  nsample = FALSE,
  genealogy = FALSE
)
```

Arguments

`object`	gpsim object.
`prune`	logical; prune the genealogy?
`obscure`	logical; obscure the demes?
`t0`	logical; return the zero-time?
`time`	logical; return the current time?
`description`	logical; return the description?
`structure`	logical; return the structure in R list format?
`yaml`	logical; return the structure in YAML format?
`ndeme`	logical; return the number of demes?
`lineages`	logical; return the lineage-count function?
`newick`	logical; return a Newick-format description of the tree?
`nsample`	logical; return the number of samples?
`genealogy`	logical; return the lineage-traced genealogy?

Value

A list containing the requested elements, including any or all of:

t0 the initial time (a numeric scalar)

time the final time (a numeric scalar)

ndeme the number of demes (an integer)

nsample the number of samples (an integer)

newick the genealogical tree, in Newick format

description a human readable description of the state of the genealogy process

yaml the state of the genealogy process in YAML format

structure the state of the genealogy process in R list format

lineages a [tibble](#) containing the lineage count function through time

genealogy the lineage-traced genealogy (as a raw vector)

Examples

```
simulate("SIIR",time=3,psi1=1,psi2=0) |>
  simulate(Beta1=2,gamma=2,time=10,psi1=10,psi2=1) |>
  plot()

runSIIR(Beta1=10,Beta2=8,
  S0=200,I1_0=10,I2_0=8,R0=0,time=0,t0=-1) |>
  simulate(psi1=10,time=2) |>
  plot(points=TRUE,obscure=FALSE)

simulate("SIIR",Beta1=2,Beta2=50,gamma=1,psi1=2,
  S0=300,I1_0=20,I2_0=2,time=5) |>
  lineages() |>
  plot()
```

lbdp	*Linear birth-death-sampling model*

Description

The genealogy process induced by a simple linear birth-death process with constant-rate sampling.

Usage

```
runLBDP(time, t0 = 0, lambda = 2, mu = 1, psi = 1, n0 = 5)

continueLBDP(object, time, lambda = NA, mu = NA, psi = NA)

lbdp_exact(x, lambda, mu, psi, n0 = 1)

lbdp_pomp(x, lambda, mu, psi, n0 = 1, t0 = 0)
```

Arguments

time	final time
t0	initial time
lambda	per capita birth rate
mu	per capita recovery rate.
psi	per capita sampling rate.
n0	initial population size
object	either the name of the model to simulate *or* a previously computed 'gpsim' object
x	genealogy in **phylopomp** format (i.e., an object that inherits from 'gpgen').

Details

`lbdp_exact` gives the exact log likelihood of a linear birth-death process, conditioned on $n_0 = 0$ (Stadler, 2010, Thm 3.5). The derivation is also given in comments in the code.

`lbdp_pomp` constructs a **pomp** object containing a given set of data and a linear birth-death-sampling process.

Value

`runLBDP` and `continueLBDP` return objects of class 'gpsim' with 'model' attribute "LBDP".

`lbdp_exact` returns the log likelihood of the genealogy. Note that the time since the most recent sample is informative.

References

T. Stadler. Sampling-through-time in birth-death trees. *Journal of Theoretical Biology* **267**, 396–404, 2010.

See Also

More example genealogy processes: moran, seir, si2r, siir, simulate(), sir

Examples

```
simulate("LBDP",time=4) |> plot(points=TRUE)

simulate("LBDP",lambda=2,mu=1,psi=3,n0=1,time=1) |>
  simulate(time=10,lambda=1) |>
  plot()

simulate("LBDP",time=4) |>
  lineages() |>
  plot()
```

---

lineages   *Lineage-count function*

Description

Lineage-counts, saturations, and event-codes.

Usage

```
lineages(object, prune = TRUE, obscure = TRUE)

## S3 method for class 'gplin'
plot(x, ..., palette = scales::hue_pal(l = 30, h = c(220, 580)))
```

Arguments

object	gpsim object.
prune	logical; prune the genealogy?
obscure	logical; obscure the demes?
x	object of class 'gpgen'
...	passed to theme.
palette	color palette for branches. This can be furnished either as a function or a vector of colors. If this is a function, it should take a single integer argument, the number of colors required. If it is a vector, it should have at least as many elements as there are demes in the genealogy.

Details

This function extracts from the specified genealogy several important time-varying quantities. These include:

lineages number of lineages through time

saturation the number of lineages emerging from the event

event_type an integer coding the type of event

If the genealogy has been obscured (the default), the number in the `lineages` returned is the total number of lineages present at the specified time and the saturation is the total saturation. If the genealogy has not been obscured (`obscure = FALSE`), the deme-specific data are returned. In this case, the deme column specifies the pertinent deme.

The event types are:

0 no event,

-1 a root,

1 a sample event,

2 a non-sample event,

3 the end of the time interval, which may or may not coincide with the latest tip of the genealogy.

Value

A [tibble](#) containing information about the genealogy. See Details for specifics. The [tibble](#) returned by `lineages` has a [plot](#) method.

Examples

```
library(tidyverse)

pal <- c("#00274c","#ffcb05")

simulate("SIIR",time=3) -> x
plot_grid(
  x |> plot(),
  x |> lineages() |> plot(),
  x |> plot(obscure=FALSE,palette=pal),
  x |> lineages(obscure=FALSE) |>
    plot(palette=pal,legend.position=c(0.8,0.9)),
  align="v",axis="b",
  ncol=2,byrow=FALSE
)
```

---

moran | *The classical Moran model*

Description

The Markov genealogy process induced by the classical Moran process, in which birth/death events occur at a constant rate and the population size remains constant.

Usage

```
runMoran(time, t0 = 0, n = 100, mu = 1, psi = 1)

continueMoran(object, time, mu = NA, psi = NA)

moran_exact(x, n = 100, mu = 1, psi = 1)
```

Arguments

time	final time
t0	initial time
n	population size
mu	event rate
psi	sampling rate.
object	either the name of the model to simulate *or* a previously computed 'gpsim' object
x	genealogy in **phylopomp** format (i.e., an object that inherits from 'gpgen').

Details

moran_exact gives the exact log likelihood of a genealogy under the uniformly-sampled Moran process.

Value

runMoran and continueMoran return objects of class 'gpsim' with 'model' attribute "Moran".

moran_exact returns the log likelihood of the genealogy.

See Also

More example genealogy processes: [lbdp](), [seir](), [si2r](), [siir](), [simulate](), [sir]()

---

newick *Newick output*

Description

Extract a Newick-format description of a genealogy.

Usage

```
newick(object, prune = TRUE, obscure = TRUE)
```

Arguments

object	gpsim object.
prune	logical; prune the genealogy?
obscure	logical; obscure the demes?

Value

A string in Newick format.

Examples

```
simulate("SIIR",time=1) |> newick()
```

---

parse_newick *parse a Newick string*

Description

Parses a Newick description and returns a binary version of the genealogy.

Usage

```
parse_newick(x, t0 = 0, tf = NA)
```

Arguments

x	character; the Newick description. See Details for specifics.
t0	numeric; the root time.
tf	numeric; the current or final time.

Value

An object of class "gpgen".

---

phylopomp *Phylodynamics for POMP models*

Description

Simulation and inference of Markov genealogy processes.

Author(s)

Aaron A. King, Qianying Lin

---

reexports *Objects exported from other packages*

Description

These objects are imported from other packages. Follow the links below to see their documentation.

cowplot [plot_grid](#)

foreach [%dopar%](#), [foreach](#), [registerDoSEQ](#)

grid [viewport](#)

pomp [bake](#), [freeze](#), [stew](#)

yaml [as.yaml](#), [read_yaml](#)

---

seir *Classical susceptible-exposed-infected-recovered model*

Description

The population is structured by infection progression.

Usage

```
runSEIR(
  time,
  t0 = 0,
  Beta = 4,
  sigma = 1,
  gamma = 1,
  psi = 1,
  omega = 0,
  S0 = 100,
```

```
  E0 = 5,
  I0 = 5,
  R0 = 0
)

continueSEIR(
  object,
  time,
  Beta = NA,
  sigma = NA,
  gamma = NA,
  psi = NA,
  omega = NA
)

seirs_pomp(x, Beta, sigma, gamma, psi, omega = 0, S0, E0, I0, R0)
```

Arguments

time	final time
t0	initial time
Beta	transmission rate
sigma	progression rate
gamma	recovery rate
psi	per capita sampling rate
omega	rate of waning of immunity
S0	initial size of susceptible population
E0	initial size of exposed population
I0	initial size of infected population
R0	initial size of immune population
object	either the name of the model to simulate *or* a previously computed 'gpsim' object
x	genealogy in **phylopomp** format.

Details

seirs_pomp constructs a **pomp** object containing a given set of data and an SEIRS model.

Value

runSEIR and continueSEIR return objects of class 'gpsim' with 'model' attribute "SEIR".

See Also

More example genealogy processes: [lbdp](), [moran](), [si2r](), [siir](), [simulate](), [sir]()

Examples

```
simulate("SEIR",Beta=2,sigma=2,gamma=1,psi=2,S0=1000,I0=5,time=5) |>
  simulate(Beta=5,gamma=2,time=10,psi=3) |>
  plot()

runSEIR(Beta=3,gamma=1,psi=2,S0=20,I0=5,R0=0,time=5,t0=-1) |>
  plot(points=TRUE,obscure=FALSE)

runSEIR(Beta=3,gamma=0.1,psi=0.2,S0=100,I0=5,R0=0,time=2,t0=0) -> x
plot_grid(plotlist=list(plot(x,points=TRUE),diagram(x)),
  ncol=1,rel_heights=c(4,1))

simulate("SEIR",sigma=1,omega=1,time=20,I0=4) |> plot(obscure=FALSE)

simulate("SEIR",sigma=1,omega=1,time=20,I0=4) |>
  lineages(obscure=FALSE) |>
  plot()
```

---

si2r   *Two-deme model of superspreading*

Description

Deme 2 consists of "superspreaders" who engender clusters of infection in "superspreading events".

Usage

```
runSI2R(
  time,
  t0 = 0,
  Beta = 5,
  mu = 5,
  gamma = 1,
  omega = 0,
  psi1 = 1,
  psi2 = 0,
  sigma12 = 1,
  sigma21 = 3,
  S0 = 500,
  I0 = 10,
  R0 = 0
)

continueSI2R(
  object,
  time,
  Beta = NA,
  mu = NA,
```

```
    gamma = NA,
    omega = NA,
    psi1 = NA,
    psi2 = NA,
    sigma12 = NA,
    sigma21 = NA
)
```

Arguments

`time`	final time
`t0`	initial time
`Beta`	transmission rate
`mu`	mean superspreading-event cluster size
`gamma`	recovery rate
`omega`	rate of waning of immunity
`psi1, psi2`	sampling rates for demes 1 and 2, respectively
`sigma12, sigma21`	
	movement rates from deme 1 to 2 and 2 to 1, respectively
`S0`	initial size of susceptible population
`I0`	initial size of I1 population (I2 = 0 at t = 0)
`R0`	initial size of recovered population
`object`	either the name of the model to simulate *or* a previously computed 'gpsim' object

Details

Superspreaders (deme 2) behave differently than ordinary infections: transmission events occur at the same rate (`Beta`), but at each event, a superspreader infects N individuals, where

$$N \sim 1 + \mathrm{Geometric}(1/\mu).$$

Thus, assuming susceptibles are not limiting, the mean number of infections resulting from a superspreading event is μ and the variance in this number is $\mu^2 - \mu$. If susceptibles are limiting, i.e., if the number of susceptibles is not greater than N, then all remaining susceptibles are infected.

Value

`runSI2R` and `continueSI2R` return objects of class 'gpsim' with 'model' attribute "SI2R".

See Also

More example genealogy processes: [lbdp](), [moran](), [seir](), [siir](), [simulate](), [sir]()

Examples

```
simulate("SI2R",time=1) |>
  plot(obscure=FALSE)

runSI2R(Beta=10,S0=2000,time=1,psi1=0) |>
  simulate(time=2,psi1=1) |>
  plot(points=TRUE,obscure=FALSE)

simulate("SI2R",time=5) |>
  lineages() |>
  plot()

simulate("SI2R",time=2) |>
  diagram(m=30)

simulate("SI2R",time=20,omega=0.2,mu=20) -> x
plot_grid(
  x |> plot(obscure=FALSE),
  x |> lineages(obscure=FALSE) |> plot(),
  ncol=1,
  align="v",axis="b"
)
```

---

siir	*Two-strain SIR model.*

Description

Two distinct pathogen strains compete for susceptibles.

Usage

```
runSIIR(
  time,
  t0 = 0,
  Beta1 = 5,
  Beta2 = 5,
  gamma = 1,
  psi1 = 1,
  psi2 = 0,
  sigma12 = 0,
  sigma21 = 0,
  omega = 0,
  S0 = 500,
  I1_0 = 10,
  I2_0 = 10,
  R0 = 0
)
```

```
continueSIIR(
  object,
  time,
  Beta1 = NA,
  Beta2 = NA,
  gamma = NA,
  psi1 = NA,
  psi2 = NA,
  sigma12 = NA,
  sigma21 = NA,
  omega = NA
)
```

Arguments

`time`	final time
`t0`	initial time
`Beta1, Beta2`	transmission rates from each of the infectious classes.
`gamma`	recovery rate.
`psi1, psi2`	sampling rates.
`sigma12, sigma21`	
	movement rates from deme 1 to 2 and 2 to 1, respectively
`omega`	rate of loss of immunity
`S0`	initial size of susceptible population.
`I1_0`	initial size of I2 population.
`I2_0`	initial size of I2 population.
`R0`	initial size of recovered population.
`object`	either the name of the model to simulate *or* a previously computed 'gpsim' object

Value

`runSIIR` and `continueSIIR` return objects of class 'gpsim' with 'model' attribute "SIIR".

See Also

More example genealogy processes: [lbdp](#), [moran](#), [seir](#), [si2r](#), [simulate](#)(), [sir](#)

Examples

```
simulate("SIIR",time=3,psi1=1,psi2=0) |>
  simulate(Beta1=2,gamma=2,time=10,psi1=10,psi2=1) |>
  plot()

runSIIR(Beta1=10,Beta2=8,
  S0=200,I1_0=10,I2_0=8,R0=0,time=0,t0=-1) |>
```

```
  simulate(psi1=10,time=2) |>
  plot(points=TRUE,obscure=FALSE)

simulate("SIIR",Beta1=2,Beta2=50,gamma=1,psi1=2,
  S0=300,I1_0=20,I2_0=2,time=5) |>
  lineages() |>
  plot()
```

---

simulate	*simulate*

Description

Simulate Markov genealogy processes

Usage

```
simulate(object, ...)

## Default S3 method:
simulate(object, ...)

## S3 method for class 'character'
simulate(object, time, ...)

## S3 method for class 'gpsim'
simulate(object, time, ...)
```

Arguments

object	either the name of the model to simulate *or* a previously computed 'gpsim' object
...	additional arguments to the model-specific simulation functions
time	end timepoint of simulation

Details

When `object` is of class 'gpsim', i.e., the result of a genealogy-process simulation, `simulate` acts to continue the simulation to a later timepoint. Note that, one cannot change initial conditions or `t0` when continuing a simulation.

Value

An object of 'gpsim' class.

See Also

More example genealogy processes: [lbdp](), [moran](), [seir](), [si2r](), [siir](), [sir]()

---

sir   *Classical susceptible-infected-recovered model*

Description

A single, unstructured population of hosts.

Usage

```
runSIR(
  time,
  t0 = 0,
  Beta = 2,
  gamma = 1,
  psi = 1,
  omega = 0,
  S0 = 100,
  I0 = 2,
  R0 = 0
)

runSIRS(
  time,
  t0 = 0,
  Beta = 2,
  gamma = 1,
  psi = 1,
  omega = 0,
  S0 = 100,
  I0 = 2,
  R0 = 0
)

continueSIR(object, time, Beta = NA, gamma = NA, psi = NA, omega = NA)

sir_pomp(x, Beta, gamma, psi, omega = 0, S0, I0, R0, t0 = 0)

runSIRS(
  time,
  t0 = 0,
  Beta = 2,
  gamma = 1,
  psi = 1,
  omega = 0,
  S0 = 100,
  I0 = 2,
  R0 = 0
```

```
)

continueSIRS(object, time, Beta = NA, gamma = NA, psi = NA, omega = NA)

sirs_pomp(x, Beta, gamma, psi, omega = 0, S0, I0, R0, t0 = 0)
```

Arguments

`time`	final time
`t0`	initial time
`Beta`	transmission rate.
`gamma`	recovery rate.
`psi`	sampling rate.
`omega`	immunity waning rate
`S0`	initial size of susceptible population.
`I0`	initial size of infected population.
`R0`	initial size of recovered population.
`object`	either the name of the model to simulate *or* a previously computed 'gpsim' object
`x`	genealogy in **phylopomp** format (i.e., an object that inherits from 'gpgen').

Details

sir_pomp constructs a **pomp** object containing a given set of data and a SIR model.

Value

runSIR and continueSIR return objects of class 'gpsim' with 'model' attribute "SIR".

See Also

More example genealogy processes: lbdp, moran, seir, si2r, siir, simulate()

Examples

```
simulate("SIR",Beta=2,gamma=1,psi=2,S0=1000,I0=5,time=5) |>
  simulate(Beta=5,gamma=2,time=10,psi=3) |>
  plot()

runSIR(Beta=3,gamma=1,psi=2,S0=20,I0=5,R0=0,time=5,t0=-1) |>
  plot(points=TRUE)

runSIR(Beta=3,gamma=0.1,psi=0.2,S0=100,I0=5,R0=0,time=2,t0=0) -> x
plot_grid(plotlist=list(plot(x,points=TRUE),diagram(x)),
  ncol=1,rel_heights=c(4,1))

simulate("SIRS",omega=1,time=20,I0=4) |> plot()
simulate("SIRS",omega=1,time=20,I0=4) |> lineages() |> plot()
```

---

treeplot   *Fancy tree plotter*

Description

Plots a genealogical tree.

Usage

```
## S3 method for class 'gpgen'
plot(x, ..., time, t0, prune = TRUE, obscure = TRUE)

treeplot(
  tree,
  time = NULL,
  t0 = 0,
  ladderize = TRUE,
  points = FALSE,
  ...,
  palette = scales::hue_pal(l = 30, h = c(220, 580))
)
```

Arguments

x	object of class 'gpgen'
...	plot passes extra arguments to treeplot. treeplot passes extra arguments to theme.
time	numeric; time of the genealogy.
t0	numeric; time of the root.
prune	logical; prune the genealogy?
obscure	logical; obscure the demes?
tree	character; tree representation in Newick format.
ladderize	Ladderize?
points	Show nodes and tips?
palette	color palette for branches. This can be furnished either as a function or a vector of colors. If this is a function, it should take a single integer argument, the number of colors required. If it is a vector, it should have at least as many elements as there are demes in the genealogy.

Value

A printable ggplot object.

Examples

```
## Not run:
  library(ggplot2)
  times <- seq(from=0,to=8,by=0.1)[-1]

  png_files <- sprintf(
    file.path(tempdir(),"frame%05d.png"),
    seq_along(times)
  )

  pb <- utils::txtProgressBar(0,length(times),0,style=3)
  x <- simulate("SIIR",time=0,Beta1=5,Beta2=10,gamma=1,omega=0.5,
    psi1=0.2,psi2=0.1,sigma12=1,sigma21=1,S0=200,I1_0=3,I2_0=2)
  for (k in seq_len(length(times))) {
    x <- simulate(x,time=times[k])
    ggsave(
      filename=png_files[k],
      plot=plot(
        x, t0=0, time=max(times),
        points=FALSE, prune=FALSE, obscure=FALSE,
        palette=c("#ffcb05","#dddddd"),
        axis.line=element_line(color="white"),
        axis.ticks=element_line(color="white"),
        axis.text=element_blank(),
        plot.background=element_rect(fill=NA,color=NA),
        panel.background=element_rect(fill=NA,color=NA)
      ),
      device="png",dpi=300,
      height=2,width=3,units="in"
    )
    setTxtProgressBar(pb,k)
  }

  library(gifski)
  gif_file <- "movie1.gif"
  gifski(png_files,gif_file,delay=0.02,loop=TRUE)
  unlink(png_files)

## End(Not run)
```

---

yaml   *YAML output*

Description

Human- and machine-readable description.

Usage

```
yaml(object)
```

Arguments

object   gpsim object.

Value

A string in YAML format, with class "gpyaml".

Examples

```
simulate("SIIR",time=1) |> yaml()
```

Index