

# Package ‘phylopomp’

May 29, 2023

## R topics documented:

as.data.frame . . . . .	1
diagram . . . . .	2
genealogy diagram internals . . . . .	3
getInfo . . . . .	4
lbdp . . . . .	6
lineages . . . . .	7
moran . . . . .	8
newick2df . . . . .	9
parse_newick . . . . .	10
phylopomp . . . . .	11
reexports . . . . .	11
seir . . . . .	11
si2r . . . . .	13
siir . . . . .	15
simulate . . . . .	17
sir . . . . .	18
treeplot . . . . .	19
yaml . . . . .	21
<b>Index</b>	<b>22</b>

---

as.data.frame	<i>Coerce to a Data Frame</i>
---------------	-------------------------------

---

## Description

Functions to coerce an object to a data frame.

## 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`.

---

diagram	<i>Genealogy process diagram</i>
---------	----------------------------------

---

**Description**

Produces a diagram of the genealogy process state.

**Usage**

```
diagram(
  object,
  prune = TRUE,
  obscure = TRUE,
  m = NULL,
  n = NULL,
  ...,
  digits = 1
)

## 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

`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))
```

---

genealogy diagram internals

*Diagramming internals*

---

**Description**

Facilities to produce diagrammatic representations of genealogy process states.

**Usage**

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

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

```
pocketGrob(object, n = NULL, 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, ...)
```

### 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
...	arguments to be passed to <a href="#">textGrob</a> .
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,
  lineages = FALSE,
  tree = FALSE
)
```

**Arguments**

<code>object</code>	gpsim object.
<code>prune</code>	logical; prune the genealogy?
<code>obscure</code>	logical; obscure the demes?
<code>t0</code>	logical; return the zero-time?
<code>time</code>	logical; return the current time?
<code>description</code>	logical; return the description?
<code>structure</code>	logical; return the structure in R list format?
<code>yaml</code>	logical; return the structure in YAML format?
<code>lineages</code>	logical; return the lineage-count function?
<code>tree</code>	logical; return the tree?

**Value**

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

**t0** the initial time

**time** the current time

**tree** 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

**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	<i>Linear birth-death-sampling model</i>
------	--

---

### 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(data, lambda, mu, psi, n0 = 1)

lbdp_pomp(data, 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 <i>or</i> a previously computed ‘gpsim’ object
data	data frame containing the genealogy event times.

### 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	<i>Lineage-count function</i>
----------	-------------------------------

---

**Description**

The number of lineages through time

**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 'gpsim'
...	passed to <a href="#">theme</a> .
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, ....

**Value**

A [tibble](#) containing the lineage count function. If the genealogy has been obscured (the default), the number in the lineages column is the total number of lineages present at the times in the time column. If the genealogy has not been obscured (obscure = FALSE), the deme-specific lineage counts are returned.

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(obscurse=FALSE, palette=pal),
  x |> lineages(obscurse=FALSE) |>
    plot(palette=pal, legend.position=c(0.8, 0.9)),
  align="v", axis="b",
  ncol=2, byrow=FALSE
)
```

---

moran	<i>The classical Moran model</i>
-------	----------------------------------

---

## 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(data, 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 <i>or</i> a previously computed ‘gpsim’ object
data	data frame containing the genealogy event times.

## 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](#)

---

newick2df

---

*Convert a tree in Newick format to data frame*


---

**Description**

Convert a genealogical tree in Newick format to a data frame suitable for use with **pomp**.

**Usage**

```
newick2df(tree, time = NA, t0 = 0)
```

**Arguments**

<code>tree</code>	tree data in Newick format.
<code>time</code>	time of the genealogy.
<code>t0</code>	time of the root.

**Details**

If `time` is furnished, it is assumed that the absence of samples between the latest leaf and `time` is informative.

**Value**

A data frame suitable for use as **pomp** input, containing three columns:

**time** numeric; time of the genealogy event.

**lineages** integer; the value of the lineage-count function at the specified time. Note that this function is right-continuous with left limits, and constant on the inter-event intervals.

**code** integer; a code describing the nature of the event. 1 indicates a coalescence; 0 indicates a dead sample; -1 indicates a live sample; 2 indicates a root.

**Examples**

```
runSIR(Beta=2,gamma=1,psi=2,S0=100,I0=2,R0=0,time=10,t0=0) |>
  getInfo(tree=TRUE) |>
  getElement("tree") |>
  newick2df(time=5)
```

---

parse_newick	<i>parse_newick</i>
--------------	---------------------

---

## Description

Parse a Newick description and extract various equivalent representations.

## Usage

```
parse_newick(
  x,
  prune = TRUE,
  obscure = TRUE,
  t0 = 0,
  time = FALSE,
  description = FALSE,
  structure = FALSE,
  yaml = FALSE,
  lineages = TRUE,
  tree = FALSE
)
```

## Arguments

<code>x</code>	character; the Newick description. See Details for specifics.
<code>prune</code>	logical; prune the genealogy?
<code>obscure</code>	logical; obscure the demes?
<code>t0</code>	numeric; the root time.
<code>time</code>	logical; return the current time?
<code>description</code>	logical; return the description?
<code>structure</code>	logical; return the structure in R list format?
<code>yaml</code>	logical; return the structure in YAML format?
<code>lineages</code>	logical; return the lineage-count function?
<code>tree</code>	logical; return the tree?

## Value

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

**time** the current time

**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

**tree** the genealogical tree, in Newick format

---

phylopomp

*Phylogenetics 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,
  delta = 0,
  S0 = 100,
```

```

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

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

```

### Arguments

time	final time
t0	initial time
Beta	transmission rate
sigma	progression rate
gamma	recovery rate
psi	per capita sampling rate
delta	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 <i>or</i> a previously computed ‘gpsim’ object

### 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,delta=1,time=20,I0=4) |> plot(obscure=FALSE)

simulate("SEIR",sigma=1,delta=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,
  delta = 0,
  psi1 = 1,
  psi2 = 0,
  sigma12 = 1,
  sigma21 = 3,
  S0 = 500,
  I0 = 10,
  R0 = 0
)

continueSI2R(
  object,
  time,
  Beta = NA,
  mu = NA,
  gamma = NA,
  delta = 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
delta	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 <i>or</i> 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

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

Thus, assuming susceptibles are not limiting, the mean number of infections resulting from a superspreading event is  $\mu$  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,delta=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,
  delta = 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,

```

```

    delta = 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
delta	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 <i>or</i> 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

<code>object</code>	either the name of the model to simulate <i>or</i> a previously computed ‘gpsim’ object
<code>...</code>	additional arguments to the model-specific simulation functions
<code>time</code>	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,
  delta = 0,
  S0 = 100,
  I0 = 2,
  R0 = 0
)
```

```
continueSIR(object, time, Beta = NA, gamma = NA, psi = NA, delta = NA)
```

```
sir_pomp(data, Beta, gamma, psi, delta = 0, S0, I0, R0, t0 = 0)
```

**Arguments**

time	final time
t0	initial time
Beta	transmission rate.
gamma	recovery rate.
psi	sampling rate.
delta	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 <i>or</i> a previously computed ‘gpsim’ object
data	data frame containing the lineage count function

**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("SIR",delta=1,time=20,I0=4) |> plot()
simulate("SIR",delta=1,time=20,I0=4) |> lineages() |> plot()
```

---

treeplot

*Fancy tree plotter*


---

**Description**

Plots a genealogical tree.

**Usage**

```
## S3 method for class 'gpsim'
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

<code>x</code>	object of class 'gpsim'
<code>...</code>	plot passes extra arguments to <code>treeplot</code> . <code>treeplot</code> passes extra arguments to <code>theme</code> .
<code>time</code>	numeric; time of the genealogy.
<code>t0</code>	numeric; time of the root.
<code>prune</code>	logical; prune the genealogy?
<code>obscure</code>	logical; obscure the demes?
<code>tree</code>	character; tree representation in Newick format.
<code>ladderize</code>	Ladderize?
<code>points</code>	Show nodes and tips?
<code>palette</code>	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, ....

## 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, delta=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, prune = TRUE, obscure = TRUE)
```

### Arguments

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

### Value

A string in yaml format.

### Examples

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

# Index

- \* **Genealogy processes**
  - lbdp, 6
  - moran, 8
  - seir, 11
  - si2r, 13
  - siir, 15
  - simulate, 17
  - sir, 18
- \* **internals**
  - genealogy diagram internals, 3
- \* **internal**
  - as.data.frame, 1
  - genealogy diagram internals, 3
  - reexports, 11
- %dopar% (reexports), 11
- %dopar%, 11
- as.data.frame, 1
- as.yaml, 11
- as.yaml (reexports), 11
- bake, 11
- bake (reexports), 11
- ballGrob (genealogy diagram internals), 3
- continueLBDP (lbdp), 6
- continueMoran (moran), 8
- continueSEIR (seir), 11
- continueSI2R (si2r), 13
- continueSIIR (siir), 15
- continueSIR (sir), 18
- diagram, 2
- drawDetails.ballGrob (genealogy diagram internals), 3
- drawDetails.resizingTextGrob (genealogy diagram internals), 3
- foreach, 11
- foreach (reexports), 11
- freeze, 11
- freeze (reexports), 11
- genealogy diagram internals, 3
- genealogyGrob (genealogy diagram internals), 3
- getInfo, 4
- LBDP (lbdp), 6
- lbdp, 6, 9, 12, 14, 16, 17, 19
- lbdp\_exact (lbdp), 6
- lbdp\_pomp (lbdp), 6
- lineages, 7
- Moran (moran), 8
- moran, 7, 8, 12, 14, 16, 17, 19
- moran\_exact (moran), 8
- newick2df, 9
- nodeGrob (genealogy diagram internals), 3
- parse\_newick, 10
- phylopomp, 11
- phylopomp-package (phylopomp), 11
- plot, 7
- plot.gplin (lineages), 7
- plot.gpsim (treeplot), 19
- plot\_grid, 11
- plot\_grid (reexports), 11
- pocketGrob (genealogy diagram internals), 3
- postDrawDetails.ballGrob (genealogy diagram internals), 3
- postDrawDetails.resizingTextGrob (genealogy diagram internals), 3
- preDrawDetails.ballGrob (genealogy diagram internals), 3

preDrawDetails.resizingTextGrob  
    (genealogy diagram internals),  
    3  
print.gpdiag(diagram), 2  
print.gpsim(genealogy diagram  
    internals), 3  
  
read\_yaml, 11  
read\_yaml (reexports), 11  
reexports, 11  
registerDoSEQ, 11  
registerDoSEQ (reexports), 11  
resizingTextGrob (genealogy diagram  
    internals), 3  
runLBDP (lbdp), 6  
runMoran (moran), 8  
runSEIR (seir), 11  
runSI2R (si2r), 13  
runSIIR (siir), 15  
runSIR (sir), 18  
  
SEIR (seir), 11  
seir, 7, 9, 11, 14, 16, 17, 19  
SI2R (si2r), 13  
si2r, 7, 9, 12, 13, 16, 17, 19  
SIIR (siir), 15  
siir, 7, 9, 12, 14, 15, 17, 19  
simulate, 7, 9, 12, 14, 16, 17, 19  
SIR (sir), 18  
sir, 7, 9, 12, 14, 16, 17, 18  
sir\_pomp (sir), 18  
stew, 11  
stew (reexports), 11  
  
textGrob, 4  
theme, 7, 20  
tibble, 5, 7, 10  
treeplot, 19, 20  
  
viewport, 11  
viewport (reexports), 11  
  
yaml, 21