

# Package ‘RLadyBug’

October 7, 2009

**Type** Package

**Version** 0.6-0

**Date** 2009-06-22

**Title** Analysis of Infectious Diseases using Stochastic Epidemic Models

**Author** Michael Hoehle, Ulrike Feldmann, Sebastian Meyer

**Maintainer** Michael Hoehle <hoehle@stat.uni-muenchen.de>

**Depends** R (>= 2.6.0), methods, rJava (>= 0.5), coda

**Imports** graphics, stats, grDevices, utils, MASS, boa, quadprog

**Description** Analysis of infectious disease data using stochastic Susceptible-Exposed-Infectious-Recovered (SEIR) models. Parts of the R package wrap functionality of a Java program, i.e. a java virtual machine has to be installed on your computer. Furthermore, infectious disease surveillance data can now be modeled based on a multivariate counting process with additive-multiplicative conditional intensities.

**License** file LICENSE

**URL** <http://www.stat.uni-muenchen.de/~hoehle/software/RLadyBug>

**LazyLoad** yes

**ZipData** no

**Repository** CRAN

**Date/Publication** 2009-10-07 07:48:05

**R topics documented:**

RLadyBug-package	2
abakaliki	4
csfv	4
hksars	5
ladybugExample	6
laevens	6
LBExperiment-class	7
LBInference-class	9
LBInferenceMCMC-class	10
LBInferenceML-class	11
LBInferenceMLK-class	12
LBLayout-class	13
LBOptions-class	14
LBOptionsMCMC-class	17
LBOptionsML-class	20
oneill	23
readSpecFile	23
seir	24
twinSIR	25
twinSIR_epidata	30
twinSIR_epidata_animate	33
twinSIR_epidata_intersperse	36
twinSIR_epidata_plot	37
twinSIR_epidata_summary	39
twinSIR_intensityPlot	41
twinSIR_methods	43
twinSIR_profile	45
twinSIR_simulation	47
<b>Index</b>	<b>53</b>

---

RLadyBug-package     *Analysis of infectious diseases using stochastic epidemic models*

---

**Description**

RLadyBug is an S4-package for the simulation, visualization and estimation of stochastic epidemic models in R. Utilizing the Susceptible-Infected-Recovered (SIR) and the S-Exposed-IR (SEIR) model as mathematical framework, maximum likelihood and Bayesian inference can be performed to estimate the parameters in data from a single outbreak of an infectious disease typical in e.g. disease transmission experiments. As of version 0.5 the package can also perform SIR modelling of infectious disease surveillance data containing several outbreaks.

## Details

Package: RLadyBug  
Type: Package  
Version: 0.6-0  
Date: 2009-06-22  
License: GPL version 2 (<http://www.gnu.org/licenses/gpl.html>)

- Likelihood and Bayesian inference for the SEIR-modelling of a single outbreak in a heterogeneous population as in Höhle et al. (2005) using an additive intensity model. Typical examples are animal disease transmission experiments. Event observations can be missing.
- Analysis of the same type of data structure based on approximate inference using Poisson regression as in Klinkenberg et al. (2002)
- Likelihood based analysis of infectious disease surveillance data using a model containing endemic and epidemic components in a counting process framework (Höhle, 2008). The endemic component can be modelled through covariates.

The aim of the package is to take a step towards statistical software supporting parameter estimation, the calculation of confidence intervals and hypothesis testing for stochastic epidemic models.

## Author(s)

Michael Höhle, Ulrike Feldmann and Sebastian Meyer

Maintainer: Michael Höhle (<[hoehle@stat.uni-muenchen.de](mailto:hoehle@stat.uni-muenchen.de)>)

## References

RLadyBug – An R package for working with stochastic epidemic models (2007), M. Höhle and U. Feldmann, *Computational Statistics and Data Analysis*, 52(2), pp. 680–686.

Höhle, M. (2008) Spatio-temporal epidemic modelling using additive-multiplicative intensity models. *Ludwig-Maximilians-Universität, Department of Statistics: Technical Reports*, No. 41. Available at <http://epub.ub.uni-muenchen.de/6366/>.

Höhle, M., Jørgensen, E. and O’Neill, P.D. (2005), *Journal of the Royal Statistical Society, Series C*, 54(2), pp. 349–366.

Klinkenberg, D., De Bree, J., Laevens, H. and De Jong, M. C. M. (2002), Within- and between-pen transmission of Classical Swine Fever Virus: a new method to estimate the basic reproduction ratio from transmission experiments, *Epidemiol. Infect.*, 128, 293-299.

## Examples

```
## Not run: demo(article-csda)
```

---

abakaliki

*Smallpox epidemic in Abakaliki, Nigeria*

---

### Description

Use MCMC to estimate parameters in the smallpox epidemic of Abakaliki, Nigeria also treated in the article by O'Neill and Roberts.

### Usage

```
data(abakaliki)
```

### Source

O'Neill, P. D. and Roberts, G. O. (1999). Bayesian inference for partially observed stochastic epidemics. *J. R. Statist. Soc. A* 162, 121–129.

### Examples

```
## Not run: data(abakaliki)
## Not run: seir(abakaliki, abakaliki.opts)
```

---

csfv

*CSFV Transmission Experiment*

---

### Description

Analysis of the transmission rates in the classical swine fever virus transmission experiment in the Dewulf et al. (2001) article.

### Usage

```
data(csfv)
```

### Details

The `csfvML` dataset is a version of `csfv`, where the exposure time is specified with an artificially assumed fixed incubation time of 6 days, except for the inoculated individual which has an incubation time of three days. This is rather ad hoc but allows us to calculate ML estimates.

The `csfvTDprior` dataset is a version of `csfv`, where rather strong priori distributions are assumed for the waiting time from exposure until diagnosis.

Note that respective `csfv.opts`, `csfvML.opts` and `csfvTD.opts` objects are loaded which provide an appropriate estimation method.

**Source**

An experimental infection with classical swine fever in E2 sub-unit marker-vaccine vaccinated and in non-vaccinated pigs, Vaccine 19, pages 475-482.

**Examples**

```
## Not run: data(csfv)
## Not run: seir(csfv, csfv.opts)
```

---

hksars

*Hong Kong SARS outbreak 2003*


---

**Description**

Severe Acute Respiratory Syndrome (SARS) data from the 2003 outbreak in Hong Kong. Data contain the daily reported number of cases among health care and others as given in Figure 2 of Anonymous (2003). A susceptible population of 6.7 mio is assumed.

**Usage**

```
data("hksars")
```

**Details**

A constant incubation time of 6.4 days and a constant recovery time of 34 days as in Donnelly et al. (2003) is assumed. Figure 2 in Anonymous (2003) provides the exposure time. A homogenous population is assumed.

**Source**

Anonymous (2003). Sars bulletin. Technical report, Health, Welfare and Food Bureau, Government of the Hong Kong Special Administrative Region, 10 June 2003. Available as <http://www.info.gov.hk/info/sars/bulletin/bulletin0610e.pdf>

Donnelly, C. A., Ghani, A. C., Leung, G. M., and 16 more authors (2003). Epidemiological determinants of spread of causal agent of severe acute respiratory syndrome in Hong Kong. *The Lancet*, 361:1761

**Examples**

```
data("hksars")

## Not run:
#Show how to plot using EpiTools
require("epitools")
require("chron")

E <- chron("2/15/2003") + hksars@data$E
curve <- epicurve.dates(E, axisnames = FALSE, before=0, after=0, legend.text = TRUE, col = color
```

```
axis(1, at = curve$xvals, labels = curve$cmday, tick = FALSE, line = 0)
axis(1, at = curve$xvals, labels = curve$month, tick = FALSE, line = 1)
## End(Not run)
```

---

`ladybugExample`      *Access files in the LadyBug directory examples directory*

---

### Description

A small helper function to access files in the LadyBug *examples/* directory.

### Usage

```
ladybugExample(exp.file)
```

### Arguments

`exp.file`      The filename relative to <LADYBUG>/examples/

### Value

The complete filename, where <LADYBUG> is replaced by options("ladybugPath").

### Author(s)

M. Höhle

### Examples

```
## Not run: ladybugExample( "/csfv/mcmc.sir" )
```

---

`laevens`      *CSFV Experiment by Laevens et. al*

---

### Description

In this experiment the spread of CSFV was investigated in a  $1 \times 3$  layout with  $S(0) = (5, 5, 6)$  and  $E(0) = (0, 1, 0)$  slaughter pigs. Every second day all pigs still alive were investigated using a virus isolation test based on blood plasma.

### Usage

```
data(laevens)
```

**Format**

The format is:

```
Formal class 'LBExperiment' [package "RLadyBug"] with 2 slots
..@ data :`data.frame':      15 obs. of  6 variables:
.. ..$ x: int [1:15] 1 1 1 1 1 1 1 1 1 1 ...
.. ..$ y: int [1:15] 2 1 1 1 1 2 2 2 2 3 ...
.. ..$ E: int [1:15] 0 18 14 20 20 10 10 10 6 20 ...
.. ..$ I: int [1:15] 6 24 20 26 26 16 16 16 12 26 ...
.. ..$ R: int [1:15] 12 34 28 34 34 32 34 28 30 28 ...
.. ..$ D: int [1:15] 12 34 28 34 34 32 34 28 30 28 ...
..@ layout:Formal class 'LLayout' [package "RLadyBug"] with 2 slots
.. .. ..@ S0: num [1, 1:3] 5 5 6
.. .. ..@ E0: num [1, 1:3] 0 1 0
```

**Details**

Together with an object `laevens` also an object `laevens.opts` is loaded which is an object of class `LBInferenceMCMC-class` suitable for MCMC inference

The data(`laevensML`) contains a version of the data, where a constant incubation time of `c=6` is assumed. Here `laevens.opts` contains the necessary object for maximum likelihood inference.

**Source**

H. Laevens, F. Koenen, H. Deluyker and A. de Kruif, Experimental infection of slaughter pigs with classical swine fever virus: transmission of the virus, course of the disease and antibody response, *Vet. Rec.*, 1999, 145:243-248.

**Examples**

```
data(laevens)
```

---

```
LBExperiment-class Class "LBExperiment"
```

---

**Description**

S4 class containing the data and the layout of the infectious disease data

**Slots**

**data:** Data Frame with six columns: x, y, E, I, R, D

**layout:** Object of class "LLayout"

**T:** A "numeric" specifying how long the epidemic was observed.

## Methods

**data2events** signature(object = "LBExperiment"): convert the data.frame of events for each individual to a time order data.frame of events for the entire Experiment. The information about each individual is lost. This function is used internally.

**show** signature(object = "LBExperiment"): shows all slots of the LBExperiment object.

**setLayout<-** signature(object = "LBExperiment"): sets the Layout.

**plot** signature(signature(x="LBExperiment", y="missing"), function(x, y, type=NULL, options=NULL, ...)) The type argument should be a formula specifying the desired type of plot. By providing an additional options list individual parameters for the plots are provided. Valid formulae are

- state ~ time | position** The number of susceptible, infectious and recovered as a function of time for each unit. Warning: in case there are many units this plot might be rather useless.
- state ~ time** shows the total number of susceptible, infectious and recovered (i.e. summed over all units) as a function of time. Individual options are
  - stacked** boolean whether stacked boxplots or just time-series are shown.
- state ~ 1|position** illustrates the three multivariate time series (susceptible, exposed, infected) as a "film" with `noOfPics` pictures. Individual options are
  - chart** Either "pie" or "bar", where the latter is default.
  - justInf** if FALSE pie charts with the number of S(t),E(t),I(t) are shown, otherwise only the number of infectious is shown, where the radius shows the proportion.
  - noOfPics** How many pictures in the animation. If not saved set the "History" attribute of the X11.
  - PDF** If TRUE the results are saved in PDF Files with the base name name.
  - name** Base name of the generated PDF Files. The actual files are then names name-addstr-number.pdf
  - addstr** This is added to the base name.
- individual ~ time** shows all events for each individual
- individual ~ time | position** show individual histories of each individual aligned to the same time axis

Additional parameters to the underlying plot routine, e.g. xlab, legend=FALSE, color, are passed using ...

## Examples

```
sim.layout <- new( "LLayout", S0=matrix( c( 13, rep( 14, 7 ) ), ncol=4 ),
                 E0=matrix( c( 1, rep( 0, 7 ) ), ncol=4 ) )
sim.opts <- new( "LBOptions", seed=2006,
                LBmodel=c( "gamma", "gamma", "gamma", FALSE ),
                ignoreData=c( FALSE, FALSE, FALSE ),
                initBeta =list( init=0.125,
                                gamma=0.001, delta=0.001 ),
                initBetaN=list( init=0.018,
                                gamma=0.001, delta=0.001 ),
                initIncu=list( asis=FALSE, const=FALSE,
```

```

                                g=6.697, g.gamma=0.001, g.delta=0.001,
                                d=0.84, d.gamma=0.001,d.delta=0.001 ),
                                initInf=list( 1.772, 0.001, 0.001, 0.123, 0.001, 0.001 ),
                                initDia=list( 149.126, 0.001, 0.001,
                                                8.737, 0.001, 0.001 ) )
exp <- simulate( sim.opts, layout=sim.layout )
plot(exp,type = state ~ time)
plot(exp,type = state ~ time, options=list(stacked=FALSE))

```

---

LBInference-class *Class "LBInference" – captures results for SEIR inference*

---

## Description

This class contains results from inference by the LadyBug program.

## Objects from the Class

Usually, there is no need to create objects of this class by hand.

## Slots

**paramHat:** Object of class "numeric" A vector with point estimates for the model parameters.

**paramSe:** Object of class "numeric" Point estimates for the standard error.

**aic:** Object of class "numeric" Akaike's Information Criterion.

**loglik:** Object of class "numeric" Value of the log likelihood.

## Methods

**infValues** signature(object = "LBInference"): Fetches a list with all slots.

**infValues<-** signature(object = "LBInference"): ...

**show** signature(object = "LBInference"): ...

**summary** signature(object = "LBInference"): ...

## See Also

[LBInferenceML-class](#) and [LBInferenceMCMC-class](#)

## Examples

```

data(oneill)
mcmc <- seir(oneill,oneill.opts)
## Not run: infValues(mcmc)

```

---

 LBInferenceMCMC-class

*Class "LBInferenceMCMC" – results from MCMC inference in SEIR models*

---

## Description

This class holds the results from MCMC inference for SEIR models, i.e. sample paths and provides routines to calculate  $R_0$

## Objects from the Class

Objects can be created by calls of the form `new("LBInferenceMCMC", paramHat, paramSe, aic, loglik, samplePaths)`.

## Slots

**samplePaths:** Object of class "data.frame" A data frame containing the va

**paramHat:** Object of class "numeric" ~~

**paramSe:** Object of class "numeric" ~~

**aic:** Object of class "numeric" ~~

**loglik:** Object of class "numeric" ~~

## Extends

Class "LBInference", directly.

## Methods

**infValues** signature(object = "LBInferenceMCMC"): ...

**infValues<-** signature(object = "LBInferenceMCMC"): ...

**initialize** signature(.Object = "LBInferenceMCMC"): ...

**plot** signature(x = "LBInferenceMCMC", y = "missing"): Important is the which argument

"beta" CODA diagnostics for the  $\beta$  parameter

"betabetaN" Provides a diagnostic plot and HPD interval for the  $\frac{\beta}{\beta_n}$  ratio.

**R0** signature(object = "LBInferenceMCMC"): Compute the basic reproduction ratio for each sample. Mean, median, etc. are then computed.

**samplePaths** signature(object = "LBInferenceMCMC"): get the sample paths

**show** signature(object = "LBInferenceMCMC"): as usual

**summary** signature(object = "LBInferenceMCMC"): as usual

## See Also

[LBInference-class](#)

**Examples**

```

#Load Laevens (99) data
data(laevens)
inf.mcmc <- seir(laevens,laevens.opts)
#Algo part of the Options
algo(laevens.opts)

#Results
inf.mcmc

#Analysis through coda (library coda is called when starting RLadyBug)
samples <- mcmc(samplePaths(inf.mcmc))
plot(samples[, "beta"])

#Look at the \beta/\beta_n ratio
ratio <- plot(inf.mcmc,which = "betabetaN")
c(mean=ratio$mean,ratio$hpd)

#R0
quantile(R0(inf.mcmc,laevens),c(0.025,0.5,0.975))

```

---

LBInferenceML-class

*Class "LBInferenceML" – results from ML inference in SEIR models*

---

**Description**

Results from a maximum likelihood inferenceise SEIR models

**Objects from the Class**

Objects can be created by calls of the form `new("LBInferenceML", ...)`. `~~` describe objects here `~~`

**Slots**

**cov:** Object of class "matrix" giving the covariance matrix of all parameters, i.e. this is the inverse negative Hessian matrix evaluated at the MLE.

**corr:** Object of class "numeric" `~~`

**paramHat:** Object of class "numeric" containing the MLE of all parameters~

**paramSe:** Object of class "numeric" containing the standard error of all parameters

**aic:** Object of class "numeric" AIC of the fitted model

**loglik:** Object of class "numeric" containing the loglik at the MLE

**Extends**

Class "LBInference", directly.

**Methods**

**infValues** signature(object = "LBInferenceML"): get all slots  
**infValues<-** signature(object = "LBInferenceML"): set a list of slots  
**show** signature(object = "LBInferenceML"): as usual  
**summary** signature(object = "LBInferenceML"): as usual  
**R0** signature(object = "LBInferenceMCMC"): Compute the basic reproduction ratio based on the largest eigenvalue of the transmission matrix.

**See Also**

[LBInference-class](#)

**Examples**

```
data(laevensML)
seir(laevensML, laevensML.opts)
```

---

LBInferenceMLK-class

*Class "LBInferenceMLK" – results from MLK inference in SEIR models*

---

**Description**

Results from the Klinkenberg inference method

**Objects from the Class**

Objects can be created by calls of the form `new("LBInferenceMLK", ...)`. `~~` describe objects here `~~`

**Slots**

**r0**: Object of class "numeric"  
**r0.ci**: Object of class "numeric" containing the confidence interval of r0  
**cov**: Object of class "matrix" giving the covariance matrix of all parameters, i.e. this is the inverse negative Hessian matrix evaluated at the MLE.  
**corr**: Object of class "numeric" `~~`  
**paramHat**: Object of class "numeric" containing the MLE of all parameters  
**paramSe**: Object of class "numeric" containing the standard error of all parameters  
**aic**: Object of class "numeric" AIC of the fitted model  
**loglik**: Object of class "numeric" containing the loglik at the MLE

**Extends**

Class "LBInferenceML", directly.

**Methods**

**infValues** signature(object = "LBInferenceMLK"): get all slots

**infValues<-** signature(object = "LBInferenceMLK"): set a list of slots

**show** signature(object = "LBInferenceMLK"): as usual

**summary** signature(object = "LBInferenceMLK"): as usual

**See Also**

[LBInferenceML-class](#)

---

LLayout-class      *Class "LLayout" – grid layout structure*

---

**Description**

This class is used to specify the spatial (or structural) arrangement of the populations. Currently only a grid layout is handled.

**Objects from the Class**

Objects can be created by calls of the form `new("LLayout", ...)`.

**Slots**

**S0:** Object of class "matrix" A matrix specifying the number of initially susceptible in each unit.

**E0:** Object of class "matrix" A matrix specifying the number of initially exposed in each unit.

**Methods**

**layoutAsDataFrame** signature(object = "LLayout"): Returns a data frame containing the columns "u", "x", "y", "S" and "E"

**layoutMatrixes** signature(object = "LLayout"): provides a list with S0 and E0 in matrix form

**show** signature(object = "LLayout"): as usual

**summary** signature(object = "LLayout"): as usual

**Note**

Currently, LadyBug is not able to handle more than one initially exposed. This will change in the near future.

**See Also**

See also [LBExperiment-class](#).

**Examples**

```
data(csfv)
```

---

LBOptions-class      *Class "LBOptions"*

---

**Description**

Specification of LadyBug SEIR models

**Objects from the Class**

Objects can be created by calls of the form `new("LBOptions", seed, LBmodel, ignoreData, initBeta, initBetaN, initIncu, initInf, initDia, algo, randomWalk)`.

**Slots**

**seed:** Object of class "numeric" The seed value to use when calling the Java program

**LBmodel:** Object of class "vector" Contains a specification of the SEIR model, i.e. a vector with names

<code>incuTimePDF</code>	distribution of incubation time
<code>infTimePDF</code>	distribution of the infectious time
<code>diagTimePDF</code>	distribution of the seroconversion time
<code>meanVar</code>	mean variance representation of periods (TRUE/FALSE)

**ignoreData:** Object of class "vector" Booleans

<code>ignoreE</code>	Ignore the specified exposure (E) event times
<code>ignoreI</code>	Ignore the specified infective (I) event times
<code>ignoreD</code>	Ignore the specified diagnose (D) event time

**initBeta:** Object of class "list" Inital values:

<code>init</code>	for $\beta$
<code>gamma</code>	for the priori parameter $\gamma$
<code>delta</code>	for the priori parameter $\delta$

**initBetaN:** Object of class "list" Inital values:

<code>init</code>	for $\beta_n$
<code>gamma</code>	for the priori parameter $\gamma$
<code>delta</code>	for the priori parameter $\delta$

**initIncu:** Object of class "list" Initial values:

g	for parameter $\gamma$ of the gamma distribution of the incubation time
g.gamma	for the parameter <i>gamma</i> of the distribution of g
g.delta	for the parameter <i>delta</i> of the distribution of g
d	for parameter $\delta$ of the gamma distribution of the incubation time
d.gamma	for the parameter <i>gamma</i> of the distribution of d
d.delta	for the parameter <i>delta</i> of the distribution of d

or choose asis or constant:

asis	TRUE/FALSE
const	TRUE/FALSE
const.val	value of constant if const == TRUE

**initInf:** Object of class "list" Initial values:

g	for parameter $\gamma$ of the gamma distribution of the infectious time
g.gamma	for the parameter <i>gamma</i> of the distribution of g
g.delta	for the parameter <i>delta</i> of the distribution of g
d	for parameter $\delta$ of the gamma distribution of the infectious time
d.gamma	for the parameter <i>gamma</i> of the distribution of d
d.delta	for the parameter <i>delta</i> of the distribution of d

**initDia:** Object of class "list" Initial values:

g	for parameter $\gamma$ of the gamma distribution of the seroconversion time
g.gamma	for the parameter <i>gamma</i> of the distribution of g
g.delta	for the parameter <i>delta</i> of the distribution of g
d	for parameter $\delta$ of the gamma distribution of the seroconversion time
d.gamma	for the parameter <i>gamma</i> of the distribution of d
d.delta	for the parameter <i>delta</i> of the distribution of d

## Methods

**ignoreData** signature(object = "LBOptions"):  
returns value of slot ignoreData

**ignoreData<-** signature(object = "LBOptions", value = "vector"):  
assigns value to slot ignoreData

**initBeta** signature(object = "LBOptions"):  
returns value of slot initBeta

**initBeta<-** signature(object = "LBOptions", value = "list"):  
assigns value to slot initBeta

**initBetaN** signature(object = "LBOptions"):  
returns value of slot initBetaN

**initBetaN<-** signature(object = "LBOptions", value = "list"):  
assigns value to slot initBetaN

```

initDia signature(object = "LBOptions"):
  returns value of slot initDia
initDia<- signature(object = "LBOptions", value = "list"):
  assigns value to slot initDia
initialize signature(.Object = "LBOptions"):
  does initializing of the slots when new( "LBOptions", ... ) is called
initIncu signature(object = "LBOptions"):
  returns value of slot initIncu
initIncu<- signature(object = "LBOptions", value = "list"):
  assigns value to slot initIncu
initInf signature(object = "LBOptions"):
  returns value of slot initInf
initInf<- signature(object = "LBOptions", value = "list"):
  assigns value to slot initInf
initsAsDataFrame signature(object = "LBOptions"):
  returns initial values in a dataframe format
LBInits signature(object = "LBOptions"):
  returns all initial values (as there are initBeta, initBetaN, initIncu, initInf,
  initDia)
LBInits<- signature(object = "LBOptions", value = "list"):
  assigns value to all initial value slots (as there are initBeta, initBetaN, initIncu,
  initInf, initDia)
LBModel signature(object = "LBOptions"):
  returns value of slot LBModel
LBModel<- signature(object = "LBOptions", value = "vector"):
  assigns value to slot LBModel
LBOptions signature(object = "LBOptions"):
  returns values of real option slots (as there are seed, LBModel, ignoreData)
LBOptions<- signature(object = "LBOptions", value = "list"):
  assigns value to real option slots (as there are seed, LBModel, ignoreData)
optionsAsDataFrame signature(object = "LBOptions"):
  returns real option values in a dataframe format
seed signature(object = "LBOptions"):
  returns value of slot seed
seed<- signature(object = "LBOptions", value = "numeric"):
  assigns value to slot seed
show signature(object = "LBOptions"):
  shows the object
simulate signature(object = "LBOptions", layout = "Layout"):
  simulates data according to the specified model and init values and the given layout structure
summary signature(object = "LBOptions"):
  gives a summary of the object (at the moment no difference to show)

```

**Author(s)**

M. Hoehle and U. Feldmann

**See Also**See also [LBOptionsMCMC-class](#) and [LBOptionsML-class](#)**Examples**

```

opts <- new( "LBOptions", seed=2003,
            LBmodel=c( "gamma", "gamma", "gamma", FALSE ),
            ignoreData=c( TRUE, FALSE, FALSE ),
            initBeta=list( 0.125, 0.001, 0.001 ),
            initBetaN=list( init=0.018, gamma=0.001, delta=0.001 ),
            initIncu=list( g=6.697, g.gamma=0.001, g.delta=0.001,
                          d=0.840, d.gamma=0.001, d.delta=0.001 ),
            initInf=list( 1.772, 0.001, 0.001, 0.123, 0.001, 0.001 ),
            initDia=list( 149.126, 0.001, 0.001, 8.737, 0.001, 0.001 ) )
layout <- new( "LBLayout", S0=matrix( c( 14, 14 ), ncol=2 ),
              E0=matrix( c( 0, 1 ), ncol=2 ) )

exp <- simulate( opts, layout=layout )

```

---

LBOptionsMCMC-class

*Class "LBOptionsMCMC" – Specification of MCMC estimation in SEIR models.*

---

**Description**

Specification of MCMC estimation in SEIR models.

**Objects from the Class**

Objects can be created by calls of the form `new("LBOptionsMCMC", seed, LBmodel, ignoreData, initBeta, initBetaN, initIncu, initInf, initDia, algo, randomWalk)`.

**Slots**

**algo:** Object of class "vector". Contains a specification of the MCMC algorithm, i.e. a vector with names

samples	how many? (without burnin)
thin	how to thin the random numbers
burnin	the first x random numbers will be ignored

**randomWalk:** Object of class "vector". Contains a specification of the random walk, i.e. a

vector with names

betaRWsigma	sigma concerning parameter $\beta$
betaNRWsigma	sigma concerning parameter $\beta_n$
gammaERWsigma	sigma concerning parameter $\gamma$ of the gamma distribution of the incubation time
deltaERWsigma	sigma concerning parameter $\delta$ of the gamma distribution of the incubation time
gammaIRWsigma	sigma concerning parameter $\gamma$ of the gamma distribution of the infectious time
deltaIRWsigma	sigma concerning parameter $\delta$ of the gamma distribution of the infectious time
gammaDRWsigma	sigma concerning parameter $\gamma$ of the gamma distribution of the seroconversion time
deltaDRWsigma	sigma concerning parameter $\delta$ of the gamma distribution of the seroconversion time
ERWsigma	sigma concerning unknown exposure times

**seed:** Object of class "numeric". The seed value to use when calling the Java program

**LBmodel:** Object of class "vector". Contains a specification of the SEIR model, i.e. a vector with names

incuTimePDF	distribution of incubation time
infTimePDF	distribution of the infectious time
diagTimePDF	distribution of the seroconversion time
meanVar	mean variance representation of periods (TRUE/FALSE)

**ignoreData:** Object of class "vector". Booleans

ignoreE	Ignore the specified exposure (E) event times
ignoreI	Ignore the specified infective (I) event times
ignoreD	Ignore the specified diagnose (D) event time

**initBeta:** Object of class "list". Initial values:

init	for $\beta$
gamma	for the priori parameter $\gamma$
delta	for the priori parameter $\delta$

**initBetaN:** Object of class "list". Initial values:

init	for $\beta_n$
gamma	for the priori parameter $\gamma$
delta	for the priori parameter $\delta$

**initIncu:** Object of class "list". Initial values:

g	for parameter $\gamma$ of the gamma distribution of the incubation time
g.gamma	for the parameter <i>gamma</i> of the distribution of g
g.delta	for the parameter <i>delta</i> of the distribution of g
d	for parameter $\delta$ of the gamma distribution of the incubation time
d.gamma	for the parameter <i>gamma</i> of the distribution of d
d.delta	for the parameter <i>delta</i> of the distribution of d

or choose asis or constant:

```

      asis      TRUE/FALSE
      const     TRUE/FALSE
      const.val value of constant if const == TRUE

```

**initInf:** Object of class "list". Initial values:

```

      g          for parameter  $\gamma$  of the gamma distribution of the infectious time
      g.gamma    for the parameter gamma of the distribution of g
      g.delta    for the parameter delta of the distribution of g
      d          for parameter  $\delta$  of the gamma distribution of the infectious time
      d.gamma    for the parameter gamma of the distribution of d
      d.delta    for the parameter delta of the distribution of d

```

**initDia:** Object of class "list". Initial values:

```

      g          for parameter  $\gamma$  of the gamma distribution of the seroconversion time
      g.gamma    for the parameter gamma of the distribution of g
      g.delta    for the parameter delta of the distribution of g
      d          for parameter  $\delta$  of the gamma distribution of the seroconversion time
      d.gamma    for the parameter gamma of the distribution of d
      d.delta    for the parameter delta of the distribution of d

```

## Extends

Class "LBOptions", directly.

## Methods

**algo** signature(object = "LBOptionsMCMC"): returns value of slot algo

**algo<-** signature(object = "LBOptionsMCMC", value = "vector"): assigns value to slot algo

**LBOptions** signature(object = "LBOptionsMCMC"): returns values of real option slots (as there are seed, LBModel, ignoreData, algo, randomWalk)

**LBOptions<-** signature(object = "LBOptionsMCMC", value = "list"): assigns value to real option slots (as there are seed, LBModel, ignoreData, algo, randomWalk)

**optionsAsDataFrame** signature(object = "LBOptionsMCMC"): returns real option values in a dataframe format

**randomWalk** signature(object = "LBOptionsMCMC"): returns value of slot randomWalk

**randomWalk<-** signature(object = "LBOptionsMCMC", value = "vector"): assigns value to slot randomWalk

**show** signature(object = "LBOptionsMCMC"): shows the object

**summary** signature(object = "LBOptionsMCMC"): gives a summary of the object (at the moment no difference to show)

**writeOptionFile** signature(object = "LBOptionsMCMC", filename="vector"): writes a file containing all options as input for java

**Author(s)**

M. Hoehle and U. Feldmann

**See Also**

See also [LBOptions-class](#) and [LBOptionsML-class](#)

**Examples**

```
opts <- new( "LBOptionsMCMC", algo=c( samples=2500, thin=25, burnin=50000 ),
            randomWalk=c( "betaRWsigma"= 0.1,
                          "betaNRWsigma"=0.1,
                          "gammaERWsigma"=3,
                          "deltaERWsigma"=1,
                          "gammaIRWsigma"=1,
                          "deltaIRWsigma"=1,
                          "gammaDRWsigma"=3,
                          "deltaDRWsigma"=1,
                          "ERWsigma"=6 ),
            seed=2003,
            LBmodel=c( "gamma", "gamma", "gamma", FALSE ),
            ignoreData=c( TRUE, FALSE, TRUE ),
            initBeta=list( 0.4, 0.001, 0.001 ),
            initBetaN=list( init=0.005, gamma=0.001, delta=0.001 ),
            initIncu=list( g=1, g.gamma=0.001, g.delta=0.001,
                          d=0.11, d.gamma=0.001, d.delta=0.001 ),
            initInf=list( 1, 0.001, 0.001, 0.11, 0.001, 0.001 ),
            initDia=list( 8, 0.001, 0.001, 0.8, 0.001, 0.001 ) )
```

---

LBOptionsML-class *Class "LBOptionsML" – maximum likelihood inference in SEIR models*

---

**Description**

Specification of LadyBug SEIR models using maximum likelihood inference

**Objects from the Class**

Objects can be created by calls of the form `new("LBOptionsML", seed, LBmodel, ignoreData, initBeta, initBetaN, initIncu, initInf, initDia, algo, randomWalk)`.

**Slots**

**seed:** Object of class "numeric" The seed value to use when calling the Java program

**LBmodel:** Object of class "vector" Contains a specification of the SEIR model, i.e. a vector with names

incuTimePDF	distribution of incubation time
infTimePDF	distribution of the infectious time
diagTimePDF	distribution of the seroconversion time
meanVar	mean variance representation of periods (TRUE/FALSE)

**ignoreData:** Object of class "vector" Booleans

ignoreE	Ignore the specified exposure (E) event times
ignoreI	Ignore the specified infective (I) event times
ignoreD	Ignore the specified diagnose (D) event time

**initBeta:** Object of class "list" Initial values:

init	for $\beta$
gamma	for the priori parameter $\gamma$
delta	for the priori parameter $\delta$

**initBetaN:** Object of class "list" Initial values:

init	for $\beta_n$
gamma	for the priori parameter $\gamma$
delta	for the priori parameter $\delta$

**initIncu:** Object of class "list" Initial values:

g	for parameter $\gamma$ of the gamma distribution of the incubation time
g.gamma	for the parameter <i>gamma</i> of the distribution of g
g.delta	for the parameter <i>delta</i> of the distribution of g
d	for parameter $\delta$ of the gamma distribution of the incubation time
d.gamma	for the parameter <i>gamma</i> of the distribution of d
d.delta	for the parameter <i>delta</i> of the distribution of d

or choose asis or constant:

asis	TRUE/FALSE
const	TRUE/FALSE
const.val	value of constant if const == TRUE

**initInf:** Object of class "list" Initial values:

g	for parameter $\gamma$ of the gamma distribution of the infectious time
g.gamma	for the parameter <i>gamma</i> of the distribution of g

g.delta for the parameter *delta* of the distribution of g  
       d for parameter  $\delta$  of the gamma distribution of the infectious time  
 d.gamma for the parameter *gamma* of the distribution of d  
 d.delta for the parameter *delta* of the distribution of d

**initDia:** Object of class "list" Initial values:

      g for parameter  $\gamma$  of the gamma distribution of the seroconversion time  
 g.gamma for the parameter *gamma* of the distribution of g  
 g.delta for the parameter *delta* of the distribution of g  
       d for parameter  $\delta$  of the gamma distribution of the seroconversion time  
 d.gamma for the parameter *gamma* of the distribution of d  
 d.delta for the parameter *delta* of the distribution of d

### Extends

Class "LBOptions", directly.

### Methods

**show** signature(object = "LBOptionsML"):  
shows the object

**summary** signature(object = "LBOptionsML"):  
gives a summary of the object (at the moment no difference to show)

**writeOptionFile** signature(object = "LBOptionsML", filename = "vector"):  
writes a file containing all options as input for java

### Author(s)

M. Hoehle and U. Feldmann

### See Also

See also [LBOptions-class](#) and [LBOptionsMCMC-class](#)

### Examples

```

opts <- new( "LBOptionsML", seed=2003,
            LBmodel=c( "constant", "gamma", "none", FALSE ),
            ignoreData=c( FALSE, FALSE, FALSE ),
            initBeta=list( 0.4, 0.001, 0.001 ),
            initBetaN=list( init=0.005, gamma=0.001, delta=0.001 ),
            initIncu=list( asis=TRUE ),
            initInf=list( 1, 0.001, 0.001, 0.11, 0.001, 0.001 ),
            initDia=list( 8, 0.001, 0.001, 0.8, 0.001, 0.001 ) )

```

---

`oneill`*Simulated data of 5 recovery times from O'Neill et. al*

---

**Description**

Use MCMC to estimate parameters in the 5 point epidemic in the 5 removal times epidemic (p.126) of the article by O'Neill and Roberts.

**Usage**

```
data(oneill)
```

**Details**

n.a.

**Source**

O'Neill, P. D. and Roberts, G. O. (1999). Bayesian inference for partially observed stochastic epidemics. *J. R. Statist. Soc. A* 162, 121–129.

**Examples**

```
## Not run: data(oneill)
## Not run: seir(oneill,oneill.opts)
```

---

`readSpecFile`*Read LadyBug files and create corresponding S4 RLadyBug objects*

---

**Description**

Data and specification files in the LadyBug files are read and converted to S4 RLadyBug objects.

**Usage**

```
readSpecFile(options, data)
```

**Arguments**

<code>options</code>	Filename of the LadyBug options (i.e. the .sir) file
<code>data</code>	Filename of the Data in LadyBug format

**Details**

n.a.

**Value**

A list containing

options      The information relevant to the Options  
 experiment    An object of class Object containing the layout and the event times of the data

**Author(s)**

U. Feldmann and M. Höhle

**Examples**

```
## Not run: csfv <- readSpecFile( ladybugExample( "/csfv/mcmc.sir"), ladybugExample( "/csfv/c
#Show the MCMC options
## Not run: csfv$options
#Show the layout and initial configuration
## Not run: csfv$experiment
```

---

seir

*Parameter estimation in SEIR-Models based on ML or MCMC*

---

**Description**

Inference is perform for the parameters in an SEIR-model based on the data in experiment. The actual class of options (OptionsML or OptionsMCMC) decides what type of inference is performed.

**Usage**

```
seir(experiment, options, debug = FALSE)
```

**Arguments**

experiment    Data corresponding to an Experiment  
 options      An object of class Options. The specific action (ML or MCMC estimation) is determined by the subclass of options.  
 debug        Boolean (default FALSE) specifying whether to yield additional debug information in case of problems. For example the ladybug.system.out and ladybug.system.err are not removed after the call and can be found in the current working directory.

**Details**

Estimation is performed by calling LadyBug using a `.jcall` to the appropriate method in the Java class `sir.estimate.LadyBug`. Output is read from file and converted into an appropriate object of class `LBInference-class`.

Currently the method branches on the appropriate method using an `if`. Should become a generic method as some point.

Note that the `system.out` and `system.err` from the java call are saved in the currented working directory (the directory has to be writable). After a successfull call the files are deleted unless one uses the `debug` option.

**Value**

An object of class `Inference`

**Author(s)**

U. Feldmann and M. Höhle

**See Also**

`LBOptions-class`, `LBOptionsML-class`, `LBOptionsMCMC-class`

**Examples**

```
data(csfvML)
ml <- seir(csfvML, csfvML.opts)
ml

#MCMC Inference for the data from the Laevens experiment
data(laevens)
inf.mcmc <- seir(laevens, laevens.opts)
#Show some results
inf.mcmc

#Analysis through coda (library coda is called when starting RLadyBug)
samples <- mcmc(samplePaths(inf.mcmc))
plot(samples[, "beta"])
```

**Description**

twinSIR is used to fit additive-multiplicative intensity models for epidemics as described in Höhle (2008). Estimation is driven by (penalized) maximum likelihood in the point process framework. Optimization (maximization) of the (penalized) likelihood function is performed by means of `optim`.

**Usage**

```
twinSIR(formula, data, weights, subset,
        knots = NULL, nIntervals = 1, lambda.smooth = 0, penalty = 1,
        optim.args = list(), model = TRUE, keep.data = FALSE)
```

**Arguments**

- |               |   |
|---------------|---|
| formula       | an object of class " <code>formula</code> " (or one that can be coerced to that class): a symbolic description of the intensity model to be estimated. The details of model specification are given under Details.  |
| data          | an object inheriting from class " <code>epidata</code> ".   |
| weights       | an optional vector of weights to be used in the fitting process. Should be <code>NULL</code> (the default, i.e. all observations have unit weight) or a numeric vector.   |
| subset        | an optional vector specifying a subset of observations to be used in the fitting process. The subset <code>atRiskY == 1</code> is automatically chosen, because the likelihood only depends on those observations.  |
| knots         | numeric vector or <code>NULL</code> (the default). Specification of the knots, where we suppose a step of the log-baseline. With the current implementation, these must be existing "stop" time points in <code>subset(data, atRiskY == 1)</code> . The intervals of constant log-baseline hazard rate then are $(minTime; knots_1]$ , $(knots_1; knots_2]$ , ..., $(knots_K; maxTime]$ . By default, the knots are automatically chosen at the quantiles of the infection time points such that <code>nIntervals</code> intervals result. Non- <code>NULL</code> knots take precedence over <code>nIntervals</code> .  |
| nIntervals    | the number of intervals of constant log-baseline hazard. Defaults to 1, which means an overall constant log-baseline hazard will be fitted.   |
| lambda.smooth | numeric, the smoothing parameter $\lambda$ . By default it is 0 which leads to unpenalized likelihood inference.  |
| penalty       | either a single number denoting the order of the difference used to penalize the log-baseline coefficients (defaults to 1), or a more specific penalty matrix $K$ for the parameter sub-vector $\beta$ .  |
| optim.args    | <p>a list with arguments passed to the <code>optim</code> function. Especially useful are the following ones:</p> <p><b>par:</b> to specify initial parameter values. Those must be in the order <code>c(alpha, h0, beta)</code>, i.e. first the coefficients of the epidemic covariates in the same order as they appear in the <code>formula</code>, then the log-baseline levels in chronological order and finally the coefficients of the endemic covariates in the same order as they appear in the <code>cox</code> terms of the <code>formula</code>. The default is to start with 1's for <code>alpha</code> and 0's for <code>h0</code> and <code>beta</code>.</p> <p><b>control:</b> for more detailed <code>trace</code>-ing (default: 1), another <code>REPORT</code>-ing frequency if <code>trace</code> is positive (default: 10), higher <code>maxit</code> (maximum number of iterations, default: 300) or another <code>factr</code> value (default: <code>1e7</code>, a lower value means higher precision).</p> <p><b>method:</b> the optimization algorithm defaults to "<code>L-BFGS-B</code>" (for box-constrained optimization), if there are any epidemic (non-<code>cox</code>) variables in the model, and to "<code>BFGS</code>" otherwise.</p> |

	<p><b>lower:</b> if <code>method = "L-BFGS-B"</code> this defines the lower bounds for the model coefficients. By default, all effects <math>\alpha</math> of epidemic variables are restricted to be non-negative. Normally, this is exactly what one would like to have, but there might be reasons for other lower bounds, see the Note below.</p> <p><b>hessian:</b> An estimation of the Expected Fisher Information matrix is always part of the return value of the function. It might be interesting to see the Observed Fisher Information (= negative Hessian at the maximum), too. This will be additionally returned if <code>hessian = TRUE</code>.</p>
<code>model</code>	logical indicating if the model frame, the <code>weights</code> , <code>lambda.smooth</code> , the penalty matrix $K$ and the list of used distance functions <code>f</code> (from <code>attributes(data)</code> ) should be returned for further computation. This defaults to <code>TRUE</code> as this information is necessary e.g. in the <code>profile</code> and <code>plot</code> methods.
<code>keep.data</code>	logical indicating if the "epidata" object 'data' should be part of the return value. This is only necessary for the use of the <code>simulate</code> method for "twinSIR" objects. The reason is that the <code>twinSIR</code> function only uses and stores the rows with <code>atRiskY == 1</code> in the model component, but for the simulation of new epidemic data one needs the whole data set with all individuals in every time block. The default value is <code>FALSE</code> , so if you intent to use <code>simulate.twinSIR</code> , you have to set this to <code>TRUE</code> .

## Details

A model is specified through the `formula`, which has the form `~ epidemicTerm1 + epidemicTerm2 + cox(endemicVar1) * cox(endemicVar2)`, i.e. the right hand side has the usual form as in `lm` with some variables marked as being endemic by the special function `cox`. The left hand side of the formula is empty and will be set internally to `cbind(start, stop, event)`, which is similar to `Surv(start, stop, event, type="counting")`.

Basically, the additive-multiplicative model for the infection intensity  $\lambda_i(t)$  for individual  $i$  is

$$\lambda_i(t) = Y_i(t) * (e_i(t) + h_i(t))$$

where

$Y_i(t)$  is the at-risk indicator, indicating if individual  $i$  is "at risk" of becoming infected at time point  $t$ . This variable is part of the event history `data`.

$e_i(t)$  is the epidemic component of the infection intensity, defined as

$$e_i(t) = \sum_{j \in I(t)} f(\|s_i - s_j\|)$$

where  $I(t)$  is the set of infectious individuals just before time point  $t$ ,  $s_i$  is the coordinate vector of individual  $i$  and the function  $f$  is defined as

$$f(u) = \sum_{m=1}^p \alpha_m B_m(u)$$

with unknown transmission parameters  $\alpha$  and known distance functions  $B_m$ . This set of distance functions results in the set of epidemic variables normally calculated by the converter

function `as.epidata`, considering the equality

$$e_i(t) = \sum_{m=1}^p \alpha_m x_{im}(t)$$

with  $x_{im}(t) = \sum_{j \in I(t)} B_m(\|s_i - s_j\|)$  being the  $m$ 'th epidemic variable for individual  $i$ .

**h\_i(t)** is the endemic (`cox`) component of the infection intensity, defined as

$$h_i(t) = \exp(h_0(t) + z_i(t)' \beta)$$

where  $h_0(t)$  is the log-baseline hazard function,  $z_i(t)$  is the vector of endemic covariates of individual  $i$  and  $\beta$  is the vector of unknown coefficients. To fit the model, the log-baseline hazard function is approximated by a piecewise constant function with known knots, but unknown levels, which will be estimated. The approximation is specified by the arguments `knots` or `nIntervals`.

If a big number of `knots` (or `nIntervals`) is chosen, the corresponding log-baseline parameters can be rendered identifiable by the use of penalized likelihood inference. At present, it is the job of the user to choose an adequate value of the smoothing parameter `lambda.smooth` or to do some cross validation in an exterior loop.

Note also that it is unwise to include endemic covariates with huge values, as they affect the intensities on the exponential scale after having been multiplied by the parameter vector  $\beta$ . With big covariates the `optim` method "L-BFGS-B" will likely terminate due to an infinite log-likelihood or score function in some iteration.

## Value

`twinSIR` returns an object of `class` "twinSIR". An object of this class is a list containing the following components:

<code>coefficients</code>	a named vector of coefficients.
<code>loglik</code>	the maximum of the (penalized) log-likelihood function.
<code>counts</code>	the number of log-likelihood and score function evaluations.
<code>converged</code>	logical indicating convergence of the optimization algorithm.
<code>fisherinfo.observed</code>	if requested, the negative Hessian from <code>optim</code> .
<code>fisherinfo</code>	an estimation of the Expected Fisher Information matrix.
<code>method</code>	the optimization algorithm used.
<code>intervals</code>	a numeric vector ( <code>c(minTime, knots, maxTime)</code> ) representing the consecutive intervals of constant log-baseline.
<code>nEvents</code>	a numeric vector containing the number of infections in each of the above <code>intervals</code> .
<code>model</code>	if requested, the model information used. This is a list with components "survs" (data.frame with the id, start, stop and event columns), "X" (matrix of the epidemic variables), "Z" (matrix of the endemic variables), "weights" (the specified weights), "lambda.smooth" (the specified <code>lambda.smooth</code> ), "K" (the penalty matrix used) and "f" (the distance functions used). Be aware that the model only contains those rows with <code>atRiskY == 1!</code>

data	if requested, the supplied "epidata" data.
call	the matched call.
formula	the specified formula.
terms	the terms object used.

### Note

There are some restrictions to modelling the infection intensity without a baseline hazard rate, i.e. without an intercept in the `formula`. Reason: At some point, the optimization algorithm L-BFGS-B tries to set all transmission parameters  $\alpha$  to the boundary value 0 and to calculate the (penalized) score function with this set of parameters (all 0). The problem then is that the values of the infection intensities  $\lambda_i(t)$  are 0 for all  $i$  and  $t$  and especially at observed event times, which is impossible. Without a baseline, it is not allowed to have all alpha's set to 0, because then we would not observe any infections. Unfortunately, L-BFGS-B can not consider this restriction. Thus, if one wants to fit a model without baseline hazard, the control parameter `lower` must be specified in `optim.args` so that some alpha is strictly positive, e.g. `optim.args = list(lower = c(0, 0.001, 0.001, 0))` and the initial parameter vector `par` must not be the zero vector.

### Author(s)

Michael Höhle and Sebastian Meyer

### References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Accepted for publication in the Biometrical Journal.

Höhle, M. (2008) Spatio-temporal epidemic modelling using additive-multiplicative intensity models. *Ludwig-Maximilians-Universität, Department of Statistics: Technical Reports*, No. 41. Available at <http://epub.ub.uni-muenchen.de/6366/>.

### See Also

`as.epidata` for the necessary data input structure, `plot.twinSIR` for plotting the path of the infection intensity, `profile.twinSIR` for profile likelihood estimation. and `simulate.twinSIR` for the simulation of epidemics following the fitted model.

Furthermore, the standard extraction methods `coef`, `vcov`, `logLik`, `AIC` and `extractAIC` are implemented for objects of class "twinSIR".

### Examples

```
data("fooepidata")
summary(fooepidata)

# fit an overall constant baseline hazard rate
fit1 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata, keep.data = TRUE)
fit1
summary(fit1)

# fit a piecewise constant baseline hazard rate with 3 intervals using
```

```

# _un_penalized ML and estimated coefs from fit1 as starting values
fit2 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata, nIntervals = 3,
  optim.args = list(par=c(coef(fit1)[1:2], rep(coef(fit1)[3],3), coef(fit1)[4])))
fit2
summary(fit2)

# fit a piecewise constant baseline hazard rate with 9 intervals
# using _penalized_ ML and estimated coefs from fit1 as starting values
fit3 <- twinSIR(~ B1 + B2 + cox(z2), data = fooepidata, nIntervals = 9,
  lambda.smooth = 0.1, penalty = 1, optim.args = list(
  par=c(coef(fit1)[1:2], rep(coef(fit1)[3],9), coef(fit1)[4])))
fit3
summary(fit3)
# plot of the 9 log-baseline levels
plot(x=fit3$intervals, y=coef(fit3)[c(3,3:11)], type="S")

### -> for more sophisticated intensity plots, see 'plot.twinSIR'
plot(fit3)

```

---

twinSIR\_epidata      *Class for Epidemic Data*

---

## Description

The function `as.epidata` converts a matrix or a data frame into an object of `class` "epidata". Objects of this class are specific data frames containing the event history of an epidemic together with some additional attributes. These objects are the basis for fitting spatio-temporal epidemic intensity models with the function `twinSIR`. Note that the spatial information itself, i.e. the positions of the individuals, is assumed to be constant over time. Besides epidemics following the SIR compartmental model, also data from SI, SIRS and SIS epidemics may be supplied. Inference for the infectious process works as usual and simulation of such epidemics is also possible.

## Usage

```

as.epidata(data, id.col, start.col, stop.col, atRiskY.col,
  event.col, Revent.col, coords.cols, f = list())

## S3 method for class 'epidata':
print(x, ...)
## S3 method for class 'epidata':
x[i, j, drop]

```

## Arguments

`data`            a `matrix` or a `data.frame`. It contains the observed event history in a form similar to `Surv(, type="counting")` with additional information (variables) along the process. It must not be sorted in any specific order; this will be done automatically during conversion. The observation period is splitted up into *consecutive* intervals of constant state - thus constant infection intensities.

The data frame consists of a block of  $N$  (number of individuals) rows for each of those time intervals (all rows in a block share the same start and stop values... therefore the name “block”), where there is one row per individual in the block. Each row describes the (fixed) state of the individual during the interval given by the start and stop columns `start.col` and `stop.col`.

Note that there may not be more than one event (infection or removal) in a single block. Thus, in a single block, only one entry in the `event.col` and `Revent.col` may be 1, all others are 0. This rule follows the assumption that there are no concurrent events (infections or removals).

<code>id.col</code>	single index of the <code>id</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>id</code> column identifies the individuals in the data frame. It will be converted to a factor variable.
<code>start.col</code>	single index of the <code>start</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>start</code> column contains the (numeric) time points of the beginnings of the consecutive time intervals of the event history. The minimum value in this column, i.e. the start of the observation period should be 0.
<code>stop.col</code>	single index of the <code>stop</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>stop</code> column contains the (numeric) time points of the ends of the consecutive time intervals of the event history. The stop value must always be greater than the start value of a row.
<code>atRiskY.col</code>	single index of the <code>atRiskY</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>atRiskY</code> column indicates if the individual was “at-risk” of becoming infected during the time interval ( <code>start</code> ; <code>stop</code> ]. This variable must be logical or in 0/1-coding. Individuals with <code>atRiskY == 0</code> in the first time interval (normally the rows with <code>start == 0</code> ) are taken as <i>initially infectious</i> .
<code>event.col</code>	single index of the <code>event</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>event</code> column indicates if the individual became <i>infected</i> at the <code>stop</code> time of the interval. This variable must be logical or in 0/1-coding.
<code>Revent.col</code>	single index of the <code>Revent</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>Revent</code> column indicates if the individual was <i>recovered</i> at the <code>stop</code> time of the interval. This variable must be logical or in 0/1-coding.
<code>coords.cols</code>	<i>indexes</i> of the <code>coords</code> columns in <code>data</code> . Can be a numeric (by column number) vector, a character (by column name) vector or <code>NULL</code> (in which case epidemic covariates are not calculateable). These columns contain the coordinates of the individuals. It must be emphasized that the functions in this package currently assume <i>fixed positions</i> of the individuals during the whole epidemic. Thus, an individual has the same coordinates in every block. For simplicity, the coordinates are derived from the first time block only (normally the rows with <code>start == 0</code> ). The epidemic covariates are calculated based on the euclidian distances between the individuals, see <code>f</code> .
<code>f</code>	a <i>named</i> list of distance functions or <code>list()</code> (the default), if calculation of epidemic covariates is not requested. The functions must interact elementwise

on a (distance) matrix so that - for a matrix  $D$  -  $f[[m]](D)$  results in a matrix. A simple example is `function(u) {u <= 1}`, which indicates if the euclidian distance between the individuals is smaller than or equal to 1. To ensure that an individual does not influence itself, the distance to itself is defined as `Inf`. Consequently, all of the distance functions must have the property  $f[[m]](\text{Inf}) = 0$ . The names of the functions will be the names of the epidemic variables in the resulting data frame. The value of such a variable is computed as follows:  $I(t)$  denotes the set of infectious individuals just before time  $t$  and  $s_i$  the coordinate vector of individual  $i$ . For individual  $i$  at time  $t$  the epidemic component  $m$  has the value  $\sum_{j \in I(t)} f_m(\|s_i - s_j\|)$

`x` an object of class "epidata".  
`...` arguments passed to `print.data.frame`.  
`i, j, drop` arguments passed to `[.data.frame]`.

### Details

The `print` method for objects of class "epidata" simply prints the data frame with a small header containing the time range of the observed epidemic and the number of infected individuals. Usually, the data frames are quite long, so the summary method `summary.epidata` might be useful. Also, indexing/subsetting "epidata" works exactly as for `data.frames`, but there is an own method, which assures consistency of the resulting "epidata" or drops this class, if necessary.

SIS epidemics are implemented as SIRS epidemics where the length of the removal period equals 0. This means that an individual, which has an R-event will be at risk immediately afterwards, i.e. in the following time block. Therefore, data of SIS epidemics have to be provided in that form containing "pseudo-R-events".

### Value

a `data.frame` with the columns "BLOCK", "id", "start", "stop", "atRiskY", "event", "Revent" and the coordinate columns (with the original names from data), which are all obligatory. These columns are followed by any remaining columns of the input data. Last but not least, the newly generated columns with epidemic variables corresponding to the functions in the list `f` are appended, if `length(f) > 0`.

The `data.frame` is given the additional *attributes*

"eventTimes" numeric vector of infection time points (sorted chronologically).  
"timeRange" numeric vector of length 2: `c(min(start), max(stop))`.  
"coords.cols" numeric vector containing the column indices of the coordinate columns in the resulting data frame.  
"f" this equals the argument `f`.

### Note

The column name "BLOCK" is a reserved name. This column will be added automatically at conversion and the resulting data frame will be sorted by this column and by id. Also the names

"id", "start", "stop", "atRiskY", "event" and "Revent" are reserved for the respective columns only.

### Author(s)

Sebastian Meyer

### See Also

The [plot](#) and the [summary](#) method for class "epidata". Furthermore, the function [animate](#) for the animation of epidemics.

Function [twinSIR](#) for fitting spatio-temporal epidemic intensity models to epidemic data.

Function [simEpidata](#) for the simulation of epidemic data.

### Examples

```
# an artificial example of an event history from the package
data("foodata")
str(foodata)

# convert the data to an object of class "epidata",
# also generating some epidemic covariates
myEpidata <- as.epidata(foodata, id.col = 1, start.col = "start",
  stop.col = "stop", atRiskY.col = "atrisk", event.col = "infected",
  Revent.col = "removed", coords.cols = c("x", "y"),
  f = list(B1 = function(u) u<=1,
    B2 = function(u) u>1 & is.finite(u))
)
# note the is.finite restriction in B2 to ensure that f[[i]](Inf) = 0, for all i

str(myEpidata)
subset(myEpidata, BLOCK == 1)

summary(myEpidata)      # see 'summary.epidata'
plot(myEpidata)        # see 'plot.epidata' and also 'animate'
stateplot(myEpidata, "15") # see 'stateplot'

## Not run:
# works in interactive mode, but not in R CMD check
data("fooepidata")
stopifnot(identical(myEpidata, fooepidata))
## End(Not run)
```

## Description

Function for the animation of epidemic data, i.e. objects inheriting from class "epidata". This only works with 1- or 2-dimensional coordinates and is not useful if some individuals share the same coordinates (overlapping). There are two types of animation, see argument `time.spacing`. Besides the direct plotting in the R session, it is also possible to generate a sequence of graphics files to create animations outside R.

## Usage

```
animate(object, main = "An animation of the epidemic", pch = 19,
        col = c(3, 2, gray(0.6)), time.spacing = NULL,
        sleep = quote(5/.nTimes), legend.opts = list(), timer.opts = list(),
        end = NULL, generate.snapshots = NULL, ...)
```

## Arguments

- |              |   |
|--------------|---|
| object       | an object inheriting from class "epidata" or "summary.epidata". In the former case, its summary is calculated and the function continues as in the latter case.   |
| main         | a main title for the plot, see also <a href="#">title</a> .   |
| pch, col     | vectors of length 3 specifying the point symbols and colors for susceptible, infectious and removed individuals (in this order). The vectors are recycled if necessary. By default, susceptible individuals are marked as filled green circles, infectious individuals as filled red circles and removed individuals as filled gray circles. Note that the symbols are iteratively drawn (overlaid) in the same plotting region as time proceeds. For information about the possible values of <code>pch</code> and <code>col</code> , see the help pages of <a href="#">points</a> and <a href="#">par</a> , respectively.   |
| time.spacing | time interval for the animation steps. If <code>NULL</code> (the default), the events are plotted one by one with pauses of <code>sleep</code> seconds. Thus, it is just the <i>ordering</i> of the events, which is shown. To plot the appearance of events proportionally to the exact time line, <code>time.spacing</code> can be set to a numeric value indicating the period of time between consecutive plots. Then, for each time point in <code>seq(0, end, by = time.spacing)</code> the current state of the epidemic can be seen and an additional timer indicates the current time (see <code>timer.opts</code> below). The argument <code>sleep</code> will be the artificial pause in seconds between two of those time points. |
| sleep        | time in seconds to <code>Sys.sleep</code> before the next plotting event. By default, each artificial pause is of length <code>5/.nTimes</code> seconds, where <code>.nTimes</code> is the number of events (infections and removals) of the epidemic, which is evaluated in the function body. Thus, for <code>time.spacing = NULL</code> the animation has a duration of approximately 5 seconds. In the other case, <code>sleep</code> is the duration of the artificial pause between two time points.  |
| legend.opts  | either a list of arguments passed to the <a href="#">legend</a> function or <code>NULL</code> (or <code>NA</code> ), in which case no legend will be plotted. All necessary arguments have sensible defaults and need not be specified, i.e.<br><br><b>x:</b> "topright"  |

```

legend: c("susceptible", "infectious", "removed")
pch: same as argument pch of the main function
col: same as argument col of the main function
timer.opts either a list of arguments passed to the legend function or NULL (or NA), in
            which case no timer will be plotted. All necessary arguments have sensible
            defaults and need not be specified, i.e.
x: "bottomright"
title: "time"
box.lty: 0
adj: c(0.5, 0.5)
inset: 0.01
bg: "white"
            Note that the argument legend, which is the current time of the animation, can
            not be modified.
end ending time of the animation in case of time.spacing not being NULL. By
    default (NULL), time stops after the last event.
generate.snapshots NULL (the default) or a list of arguments passed to function dev.print, which
    then is executed at each time point of the grid defined by time.spacing.
    Note that this only works with time.spacing not being NULL. Essentially,
    this is used for saving the produced snapshots to files, e.g. generate.snapshots
    = list(device=pdf, file=quote(paste("epidemic_", sprintf(form,t0), ".pdf",
    sep=""))) will store the animation steps in pdf-files in the current working
    directory, where the file names each end with the time point represented by the
    corresponding plot. Because the variables t0 and form are evaluated inside the
    function the file argument is quoted.
... further graphical parameters passed to the basic call of plot, e.g. las, cex.axis
    (etc.) and mgp.

```

**Author(s)**

Sebastian Meyer

**See Also**

[summary.epidata](#) for the data, on which the plot is based. [plot.epidata](#) for plotting the evolution of an epidemic by the numbers of susceptible, infectious and removed individuals.

**Examples**

```

data("fooepidata")
s <- summary(fooepidata)

# plot the ordering of the events only
animate(s) # or animate(fooepidata)

# with timer
animate(s, time.spacing = 0.1)

```

---

`twinSIR_epidata_intersperse`*Impute Blocks for Extra Stops in "epidata" Objects*

---

### Description

This function modifies an object inheriting from class "epidata" such that it features the specified stop time points. For this purpose, the time interval in the event history into which the new stop falls will be splitted up into two parts, one block for the time period until the new stop – where no infection or removal occurs – and the other block for the time period from the new stop to the end of the original interval.

Main application is to enable the use of knots in twinSIR, which are not existing stop time points in the "epidata" object.

### Usage

```
intersperse(epidata, stoptimes)
```

### Arguments

`epidata` an object inheriting from class "epidata".  
`stoptimes` a numeric vector of time points inside the observation period of the `epidata`.

### Value

an object of the same class as `epidata` with additional time blocks for any new `stoptimes`.

### Author(s)

Sebastian Meyer

### Examples

```
data("fooepidata")
subset(fooepidata, start < 25 & stop > 25, select = 1:7)
nrow(fooepidata)
moreStopsEpi <- intersperse(fooepidata, c(25,75))
nrow(moreStopsEpi)
subset(moreStopsEpi, stop == 25 | start == 25, select = 1:7)
```

---

twinSIR\_epidata\_plot

*Plotting the Evolution of an Epidemic*


---

## Description

Functions for plotting the evolution of epidemics. The `plot` methods for classes `"epidata"` and `"summary.epidata"` plots the numbers of susceptible, infectious and recovered (= removed) individuals by step functions along the time axis. The function `stateplot` shows individual state changes along the time axis.

## Usage

```
## S3 method for class 'summary.epidata':
plot(x, lty = c(2, 1, 3), lwd = 1, col = 1, col.hor = col,
      col.vert = col, xlab = "Time", ylab = "Number of individuals",
      xlim = NULL, ylim = NULL, legend.opts = list(), do.axis4 = NULL,
      panel.first = grid(), rug.opts = list(),
      which.rug = c("infections", "removals", "susceptibility", "all"), ...)
## S3 method for class 'epidata':
plot(x, ...)

stateplot(x, id, ...)
```

## Arguments

<code>x</code>	an object inheriting from class <code>"epidata"</code> or <code>"summary.epidata"</code> . In the former case, its summary is calculated and the function continues as in the latter case. The <code>plot</code> method for class <code>"epidata"</code> is a simple wrapper for <code>plot.summary.epidata</code> implemented as <code>plot(summary(x, ...))</code> .
<code>lty</code> , <code>lwd</code>	vectors of length 3 containing the line types and widths, respectively, for the numbers of susceptible, infectious and removed individuals (in this order). By default, all lines have width 1 and the line types are dashed (susceptible), solid (infectious) and dotted (removed), respectively. To omit the drawing of a specific line, just set the corresponding entry in <code>lty</code> to 0. The vectors are recycled if necessary. For information about the different <code>lty</code> and <code>lwd</code> codes, see the help pages of <a href="#">par</a> .
<code>col</code> , <code>col.hor</code> , <code>col.vert</code>	vectors of length 3 containing the line colors for the numbers of susceptible, infectious and removed individuals (in this order). <code>col.hor</code> defines the color for the horizontal parts of the step function, whilst <code>col.vert</code> defines the color for its vertical parts. The argument <code>col</code> is just short for <code>col.hor = col</code> and <code>col.vert = col</code> . By default, all lines are completely drawn in black. The vectors are recycled if necessary. For information about the possible values of <code>col</code> , see the help pages of <a href="#">par</a> .
<code>xlab</code> , <code>ylab</code>	axis labels, default to <code>"Time"</code> and <code>"Number of individuals"</code> , respectively.

<code>xlim, ylim</code>	the x and y limits of the plot in the form <code>c(xmin, xmax)</code> and <code>c(ymin, ymax)</code> , respectively. By default, these are chosen adequately to fit the time range of the epidemic and the number of individuals.
<code>legend.opts</code>	either a list of arguments passed to the <code>legend</code> function or <code>NULL</code> (or <code>NA</code> ), in which case no legend will be plotted. All necessary arguments have sensible defaults and need not be specified, i.e. <b>x:</b> "topright" <b>legend:</b> <code>c("susceptible", "infectious", "removed")</code> <b>lty:</b> same as argument <code>lty</code> of the main function <b>lwd:</b> same as argument <code>lwd</code> of the main function <b>col:</b> same as argument <code>col.hor</code> of the main function <b>bty:</b> "n"
<code>do.axis4</code>	logical indicating if the final numbers of susceptible and removed individuals should be indicated on the right axis. The default <code>NULL</code> means <code>TRUE</code> , if <code>x</code> represents a SIR epidemic and <code>FALSE</code> otherwise, i.e. if the epidemic is SI, SIS or SIRS.
<code>panel.first</code>	an expression to be evaluated after the plot axes are set up but before any plotting takes place. By default, a standard grid is drawn.
<code>rug.opts</code>	either a list of arguments passed to the function <code>rug</code> or <code>NULL</code> (or <code>NA</code> ), in which case no rug will be plotted. By default, the argument <code>ticksize</code> is set to 0.02 and <code>quiet</code> is set to <code>TRUE</code> . Note that the argument <code>x</code> , which contains the locations for the rug is fixed internally and can not be modified. The argument <code>which.rug</code> (see below) determines the locations to mark.
<code>which.rug</code>	By default, tick marks are drawn at the time points of infections. Alternatively, one can choose to mark only "removals", "susceptibilities" (i.e. state change from R to S) or "all" events.
<code>id</code>	single character string or factor of length 1 specifying the individual for which the stateplot should be established.
<code>...</code>	For <code>plot.summary.epidata</code> : further graphical parameters passed to <code>plot</code> , <code>lines</code> and <code>axis</code> , e.g. <code>main</code> , <code>las</code> , <code>cex.axis</code> (etc.) and <code>mgp</code> . For <code>plot.epidata</code> : arguments passed to <code>plot.summary.epidata</code> . For <code>stateplot</code> : arguments passed to <code>plot.stepfun</code> or <code>plot.function</code> (if <code>id</code> had no events during the observation period). By default, <code>xlab="time"</code> , <code>ylab="state"</code> , <code>xlim=attr(x, "timeRange")</code> , <code>xaxs="i"</code> and <code>do.points=FALSE</code> .

### Value

`plot.summary.epidata` (and `plot.epidata`) invisibly returns the matrix used for plotting, which contains the evolution of the three counters.

`stateplot` invisibly returns the function, which was plotted, typically of class "stepfun", but maybe of class "function", if no events have been observed for the individual in question (then the function always returns the initial state). The vertical axis of `stateplot` can range from 1 to 3, where 1 corresponds to Susceptible, 2 to *Infectious* and 3 to *Removed*.

### Author(s)

Sebastian Meyer

**See Also**

[summary.epidata](#) for the data, on which the plots are based. [animate](#) for the animation of epidemics.

**Examples**

```
data("fooepidata")
s <- summary(fooepidata)

# evolution of the epidemic
par(las = 1)
plot(s)

# stateplot
stateplot(s, id = "15", main = "Some individual event paths")
stateplot(s, id = "1", add = TRUE, col = 2)
stateplot(s, id = "20", add = TRUE, col = 3)
legend("topright", legend = c(15, 1, 20), title = "id", lty = 1, col = 1:3,
      inset = 0.1)
```

---

```
twinSIR_epidata_summary
      Summarizing an Epidemic
```

---

**Description**

The [summary](#) method for [class "epidata"](#) gives an overview of the epidemic. Its [print](#) method shows the type of the epidemic, the time range, the total number of individuals, the initially and never infected individuals and the size of the epidemic. An excerpt of the returned `counters` data frame is also printed (see the Value section below).

**Usage**

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

## S3 method for class 'summary.epidata':
print(x, ...)
```

**Arguments**

<code>object</code>	an object inheriting from class "epidata".
<code>x</code>	an object inheriting from class "summary.epidata", i.e. an object returned by the function <code>summary.epidata</code> .
<code>...</code>	unused (argument of the generic).

**Value**

A list with the following components:

<code>type</code>	character string. Compartmental type of the epidemic, i.e. one of "SIR", "SI", "SIS" or "SIRS".
<code>size</code>	integer. Size of the epidemic, i.e. the number of initially susceptible individuals, which became infected during the course of the epidemic.
<code>initiallyInfected</code>	factor (with the same levels as the <code>id</code> column in the "epidata" object). Set of initially infected individuals.
<code>neverInfected</code>	factor (with the same levels as the <code>id</code> column in the "epidata" object). Set of never infected individuals, i.e. individuals, which were neither initially infected nor infected during the course of the epidemic.
<code>coordinates</code>	numeric matrix of individual coordinates with as many rows as there are individuals and one column for each spatial dimension. The row names of the matrix are the <code>ids</code> of the individuals.
<code>byID</code>	data frame with time points of infection and optionally removal and re-susceptibility (depending on the <code>type</code> of the epidemic) ordered by <code>id</code> . If an event was not observed, the corresponding entry is missing.
<code>counters</code>	data frame containing all events (S, I and R) ordered by time. The columns are <code>time</code> , <code>type</code> (of event), corresponding <code>id</code> and the three counters <code>nSusceptible</code> , <code>nInfectious</code> and <code>nRemoved</code> . The first row additionally shows the counters at the beginning of the epidemic, where the <code>type</code> and <code>id</code> column contain missing values.

**Author(s)**

Sebastian Meyer

**See Also**

[as.epidata](#) for generating objects of class "epidata".

**Examples**

```
data("fooepidata")
s <- summary(fooepidata)
s          # uses the print method for summary.epidata
names(s)  # components of the list 's'

# positions of the individuals
plot(s$coordinates)

# events by id
head(s$byID)
```

---

twinSIR\_intensityPlot

*Plotting Paths of Infection Intensities*


---

## Description

Function to plot the values of the total infection intensity, its epidemic proportion and its endemic proportion along the evolution of the epidemic.

## Usage

```
intensityPlot(x, type = c("overall", "individual"),
              what = c("epidemic proportion", "endemic proportion",
                      "total intensity"),
              theta = NULL, plot = TRUE, add = FALSE, rug.opts = list(), ...)

## S3 method for class 'twinSIR':
plot(x, type = c("overall", "individual"),
     what = c("epidemic proportion", "endemic proportion", "total intensity"),
     theta = coef(x), plot = TRUE, add = FALSE, rug.opts = list(), ...)
```

## Arguments

x	an object of class "simEpidata" or "twinSIR".
type	single character: "overall" or "individual". Partial matching is applied. Determines whether lines for all individual infection intensities should be drawn or their sum only.
what	single character: "epidemic proportion", "endemic proportion" or "total intensity". Partial matching is applied. Determines whether to plot the path of the total intensity $\lambda(t)$ or its epidemic or endemic proportions $\frac{e(t)}{\lambda(t)}$ or $\frac{h(t)}{\lambda(t)}$ .
theta	numeric vector of model coefficients. If x is of class "twinSIR", then theta = c(alpha, beta), where beta consists of the coefficients of the piecewise constant log-baseline function and the coefficients of the endemic (cox) predictor. If x is of class "simEpidata", then theta = c(alpha, 1, betarest), where 1 refers to the (true) log-baseline used in the simulation and betarest is the vector of the remaining coefficients of the endemic (cox) predictor. The default (NULL) means that the fitted or true parameters, respectively, will be used.
plot	logical indicating if a plot is desired, defaults to TRUE. Otherwise, only the data of the plot will be returned. Especially with type = "individual" and many individuals one might e.g. consider to plot a subset of the individual intensity paths only or do some further calculations/analysis of the infection intensities.
add	logical. If TRUE, plots are added to current one, using lines.

`rug.opts` either a list of arguments passed to the function `rug` or NULL (or NA), in which case no `rug` will be plotted. By default, the argument `ticksize` is set to 0.02 and `quiet` is set to TRUE. Note that the argument `x`, which contains the locations for the `rug` is fixed internally and can not be modified. The locations of the `rug` are the time points of infections.

`...` further graphical parameters passed to the function `matplot`, e.g. `lty`, `lwd`, `col`, `xlab`, `ylab` and `main`. Note that the `matplot` arguments `x`, `y`, `type` and `add` are implicit and can not be specified here.

### Value

numeric matrix with the first column "stop" and as many rows as there are "stop" time points in the event history `x`. The other columns depend on the argument `type`: if `type = "overall"` there is only one other column named `what`, which contains the values of `what` at the respective "stop" time points. Otherwise, if `type = "individual"`, there is one column for each individual, each of them containing the individual `what` at the respective "stop" times points.

### Author(s)

Sebastian Meyer

### References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Accepted for publication in the Biometrical Journal.

Höhle, Michael (2008) Spatio-temporal epidemic modelling using additive-multiplicative intensity models. *Ludwig-Maximilians-Universität, Department of Statistics: Technical Reports*, No. 41. Available at <http://epub.ub.uni-muenchen.de/6366/>.

### See Also

`twinSIR` or Höhle (2008) for a more detailed description of the intensity model.

### Examples

```
data("fooepidata")
data("foofit")

# an overview of the evolution of the epidemic
plot(fooepidata)

# overall total intensity
plot(foofit, what="total")

# overall epidemic proportion
head(plot(foofit, what="epidemic"))

# add the inverse overall endemic proportion = 1 - epidemic proportion
head(plot(foofit, what="endemic", add=TRUE, col=2))
legend("right", legend="endemic proportion \n(= 1 - epidemic proportion)",
```

```

        lty=1, col=2, bty="n")

# individual intensities
tmp <- plot(foofit, type="individual", what="total",
           col=rgb(0,0,0,alpha=if(getRversion() < "2.7.0") 1 else 0.1),
           main=expression("Individual infection intensities" *
                           lambda[i](t) == Y[i](t) %.% (e[i](t) + h[i](t))))
str(tmp)

# and only for individuals 3 and 99
matplot(x= tmp[,1], y=tmp[,1+c(3,99)], type="S", ylab="infection intensity",
        xlab="time", main=expression("Paths of the infection intensities" *
                                     lambda[3](t) * " and " * lambda[99](t)))
legend("topright", legend=paste("Individual", c(3,99)), col=c(1,2), lty=c(1,2))

```

---

twinSIR\_methods      *Print, Summary and Extraction Methods for "twinSIR" Objects*

---

## Description

Besides `print` and `summary` methods there are also some standard extraction methods defined for objects of class "twinSIR": `coef`, `vcov`, `logLik` and especially `AIC` and `extractAIC`, which extract Akaike's Information Criterion. Note that special care is needed, when fitting models with parameter constraints such as the epidemic effects  $\alpha$  in `twinSIR` models. Parameter constraints reduce the average increase in the maximized loglikelihood - thus the penalty for constrained parameters should be smaller than the factor 2 used in the ordinary definition of AIC. To this end, these two methods offer the calculation of the so-called one-sided AIC (OSAIC).

## Usage

```

## S3 method for class 'twinSIR':
print(x, digits = max(3, getOption("digits") - 3), ...)
## S3 method for class 'twinSIR':
summary(object,
         correlation = FALSE, symbolic.cor = FALSE, ...)

## S3 method for class 'twinSIR':
AIC(object, ..., k = 2, one.sided = NULL, nsim = 1e3)
## S3 method for class 'twinSIR':
extractAIC(fit, scale = 0, k = 2, one.sided = NULL,
          nsim = 1e3, ...)

## S3 method for class 'twinSIR':
coef(object, ...)
## S3 method for class 'twinSIR':
vcov(object, ...)
## S3 method for class 'twinSIR':
logLik(object, ...)

```

**Arguments**

<code>x, object, fit</code>	an object of class "twinSIR".
<code>digits</code>	integer, used for number formatting with <code>signif()</code> . Minimum number of significant digits to be printed in values.
<code>correlation</code>	logical. if TRUE, the correlation matrix of the estimated parameters is returned and printed.
<code>symbolic.cor</code>	logical. If TRUE, print the correlations in a symbolic form (see <code>symnum</code> ) rather than as numbers.
<code>...</code>	For the <code>summary</code> method: arguments passed to <code>extractAIC.twinSIR</code> . For the <code>AIC</code> method, optionally more fitted model objects. For the <code>print</code> , <code>extractAIC</code> , <code>coef</code> , <code>vcov</code> and <code>logLik</code> methods: unused (argument of the generic).
<code>k</code>	numeric specifying the "weight" of the <i>penalty</i> to be used; in an unconstrained fit $k = 2$ is the classical AIC.
<code>one.sided</code>	logical or NULL (the default). Determines if the one-sided AIC should be calculated instead of using the classical penalty $k \cdot \text{edf}$ . The default value NULL chooses classical AIC in the case of an unconstrained fit and one-sided AIC in the case of constraints. The type of the fit can be seen in <code>object\$method</code> (or <code>fit\$method</code> respectively), where "L-BFGS" means constrained optimization.
<code>nsim</code>	number of simulations to use for determining the weights in the OSAIC formula when there are more than two epidemic covariates in the fit. Defaults to 1000 samples.
<code>scale</code>	unused (argument of the generic).

**Details**

The `print` and `summary` methods allow the compact or comprehensive representation of the fitting results, respectively. The former only prints the original function call, the estimated coefficients and the maximum log-likelihood value. The latter prints the whole coefficient matrix with standard errors, z- and p-values (see `printCoefmat`), and additionally the number of infections per log-baseline `interval`, the (one-sided) AIC and the number of log-likelihood evaluations. They both append a big "WARNING", if the optimization algorithm did not converge.

The two AIC functions differ only in that `AIC` can take more than one fitted model object and that `extractAIC` always returns the number of parameters in the model (`AIC` only does with more than one fitted model object).

Concerning the choice of one-sided AIC: parameter constraints – such as the non-negative constraints for the epidemic effects  $\alpha$  in `twinSIR` models – reduce the average increase in the maximized loglikelihood. Thus, the penalty for constrained parameters should be smaller than the factor 2 used in the ordinary definition of AIC. One-sided AIC (OSAIC) suggested by Hughes and King (2003) is such a proposal when  $p$  out of  $k = p + q$  parameters have non-negative constraints:

$$OSAIC = -2l(\theta, \tau) + 2 \sum_{g=0}^p w(p, g)(k - p + g)$$

where  $w(p, g)$  are  $p$ -specific weights. For more details see Section 5.2 in Höhle (2008).

**Value**

See the documentation for the generic functions `AIC` and `extractAIC`, respectively.

**Author(s)**

Michael Höhle and Sebastian Meyer

**References**

Hughes A, King M (2003) Model selection using AIC in the presence of one-sided information. *Journal of Statistical Planning and Inference* **115**, pp. 397–411.

Höhle, Michael (2008) Spatio-temporal epidemic modelling using additive-multiplicative intensity models. *Ludwig-Maximilians-Universität, Department of Statistics: Technical Reports*, No. 41. Available at <http://epub.ub.uni-muenchen.de/6366/>.

**Examples**

```
data("foofit")

foofit

coef(foofit)
vcov(foofit)
logLik(foofit)

summary(foofit, correlation = TRUE, symbolic.cor = TRUE)

# AIC or OSAIC
AIC(foofit)
AIC(foofit, one.sided = FALSE)
extractAIC(foofit)
extractAIC(foofit, one.sided = FALSE)

# with BIC-like penalty weight
AIC(foofit, k = log(nlevels(foofit$model$survs$id)))

# just as a stupid example for the use of AIC with multiple fits
foofit2 <- foofit
AIC(foofit, foofit2) # 2nd column should actually be named "OSAIC" here
```

**Description**

Function to compute estimated and profile likelihood based confidence intervals. Computations might be cumbersome!

**Usage**

```
## S3 method for class 'twinSIR':
profile(fitted, profile, alpha = 0.05,
        control = list(fnscale = -1, factr = 10, maxit = 100), ...)
```

**Arguments**

fitted	an object of class "twinSIR".
profile	a list with elements being numeric vectors of length 4. These vectors must have the form <code>c(index, lower, upper, gridsize)</code> .  <b>index:</b> index of the parameter to be profiled in the vector <code>coef(fitted)</code> . <b>lower, upper:</b> lower/upper limit of the grid on which the profile log-likelihood is evaluated. Can also be NA in which case lower/upper equals the lower/upper bound of the respective 0.3 % Wald confidence interval ( $\pm 3 \cdot se$ ). <b>gridsize:</b> grid size of the equally spaced grid between lower and upper. Can also be 0 in which case the profile log-likelihood for this parameter is not evaluated on a grid.
alpha	$(1 - \alpha)\%$ profile likelihood based confidence intervals are computed. If $\alpha \leq 0$ , then no confidence intervals are computed.
control	control object to use in <code>optim</code> for the profile log-likelihood computations.
...	unused (argument of the generic).

**Value**

list with profile log-likelihood evaluations on the grid and highest likelihood and wald confidence intervals. The argument `profile` is also returned.

**Author(s)**

Michael Höhle

**Examples**

```
data("foofit")
# the following call takes a while
## Not run:
prof <- profile(foofit, list(c(1,0,0.05,5), c(3,NA,NA,0), c(4, NA, NA, 10)))
prof
## End(Not run)
```

---

twinSIR\_simulation *Simulation of Epidemic Data*

---

## Description

This function simulates the infection (and removal) times of an epidemic. Besides the classical SIR type of epidemic, also SI, SIRS and SIS epidemics are supported. Simulation works via the conditional intensity of infection of an individual, given some (time varying) endemic covariates and/or some distance functions (epidemic components) as well as the fixed positions of the individuals. The lengths of the infectious and removed periods are generated following a pre-specified function (can be deterministic).

The `simulate` method for objects of class "twinSIR" simulates new epidemic data using the model and the parameter estimates of the fitted object.

## Usage

```
simEpidata(formula, data, id.col, I0.col, coords.cols, subset,
           beta, h0, f = list(), alpha, infPeriod,
           remPeriod = function(ids) rep(Inf, length(ids)),
           end = Inf, trace = FALSE, .allocate = 500L)

## S3 method for class 'twinSIR':
simulate(object, nsim = 1, seed = 1,
         infPeriod = NULL, remPeriod = NULL,
         end = diff(range(object$intervals)), trace = FALSE, .allocate = 500L,
         data = object$data, ...)
```

## Arguments

formula	an object of class "formula" (or one that can be coerced to that class): a symbolic description of the intensity model to be estimated. The details of model specification are given under Details.
data	a data.frame containing the variables in formula and the variables specified by id.col, I0.col and coords.col (see below). It represents the "history" of the endemic covariates to use for the simulation. The form is similar to and can be an object of class "epidata". The simulation period is splitted up into <i>consecutive</i> intervals of constant endemic covariables. The data-frame consists of a block of N (number of individuals) rows for each of those time intervals (all rows in a block share the same start and stop values... therefore the name "block"), where there is one row per individual in the block. Each row describes the (fixed) state of the endemic covariates of the individual during the time interval given by the start and stop columns (specified through the lhs of formula).  For the simulate method of class "twinSIR" this should be the object of class "epidata" used for the fit. This is a part of the return value of the function twinSIR, if called with argument keep.data set to TRUE.

<code>id.col</code>	only if <code>data</code> does not inherit from <code>epidata</code> : single index of the <code>id</code> column in <code>data</code> . Can be numeric (by column number) or character (by column name). The <code>id</code> column identifies the individuals in the data-frame. It will be converted to a factor variable and its levels serve also to identify individuals as argument to the <code>infPeriod</code> function.
<code>I0.col</code>	only if <code>data</code> does not inherit from <code>epidata</code> : single index of the <code>I0</code> column in <code>data</code> . Can be numeric (by column number), character (by column name) or <code>NULL</code> . The <code>I0</code> column indicates if an individual is initially infectious, i.e. it is already infectious at the beginning of the first time block. Setting <code>I0.col = NULL</code> is short for “there are no initially infectious individuals”. Otherwise, the variable must be logical or in 0/1-coding. As this variable is constant over time the initially infectious individuals are derived from the first time block only.
<code>coords.cols</code>	only if <code>data</code> does not inherit from <code>epidata</code> : indexes of the <code>coords</code> columns in <code>data</code> . Can be a numeric (by column number), a character (by column name) vector or <code>NULL</code> . These columns contain the coordinates of the individuals. It must be emphasized that the functions in this package currently assume <i>fixed positions</i> of the individuals during the whole epidemic. Thus, an individual has the same coordinates in every block. For simplicity, the coordinates are derived from the first time block only. The epidemic covariates are calculated based on the euclidian distance between the individuals, see <code>f</code> .
<code>subset</code>	an optional vector specifying a subset of the covariate history to be used in the simulation.
<code>beta</code>	numeric vector of length equal the number of endemic ( <code>cox</code> ) terms on the rhs of <code>formula</code> . It contains the effects of the endemic predictor (excluding the log-baseline <code>h0</code> , see below) in the same order as in the formula.
<code>h0</code>	<i>either</i> a single number to specify a constant baseline hazard (equal to $\exp(h0)$ ) <i>or</i> a list of functions named <code>exact</code> and <code>upper</code> . In the latter case, <code>h0\$exact</code> is the true log-baseline hazard function and <code>h0\$upper</code> is a <i>piecewise constant upper bound</i> for <code>h0\$exact</code> . The function <code>h0\$upper</code> must inherit from <code>stepfun</code> with <code>right=FALSE</code> . Theoretically, the intensity function is left-continuous, thus <code>right=TRUE</code> would be adequate, but in the implementation, when we evaluate the intensity at the <i>knots</i> (change points) of <code>h0\$upper</code> we need its value for the subsequent interval.
<code>f</code>	a <i>named</i> list of distance functions or <code>list()</code> (the default), if no epidemic component is desired (if additionally <code>infPeriod</code> (see below) always returns <code>Inf</code> , then one simulates from the Cox model). The functions must interact elementwise on a (distance) matrix so that - for a matrix <code>D</code> - <code>f[[m]](D)</code> results in a matrix. A simple example is <code>function(u) {u &lt;= 1}</code> , which indicates if the euclidian distance between the individuals is smaller than or equal to 1. To ensure that an individual does not influence itself, the distance to itself is defined as <code>Inf</code> . Consequently, all of the distance functions must have the property <code>f[[m]](Inf) = 0</code> . The names of the functions will be the names of the epidemic variables in the resulting data-frame. So, the names should not coincident with names of endemic variables. The value of such an epidemic variable is computed as follows: $I(t)$ denotes the set of infectious individuals just before time

	$t$ and $s_i$ the coordinate vector of individual $i$ . For individual $i$ at time $t$ the epidemic component $m$ has the value $\sum_{j \in I(t)} f_m(\ s_i - s_j\ )$
alpha	numeric vector of length equal the number of epidemic terms, i.e. the number of distance functions in $\mathbf{f}$ . It contains the effects of the epidemic predictor in the same order as the distance functions in $\mathbf{f}$ or, if names are supplied, matching to the distance functions will be done by name.
infPeriod	<p>a function generating lengths of infectious periods. It should take one parameter (e.g. <code>ids</code>), which is a character vector of id's of individuals, and return appropriate infection periods for those individuals. Therefore, the value of the function should be of length <code>length(ids)</code>. For example, for independent and identically distributed infection periods following <math>Exp(1)</math>, the generating function is <code>function(ids) rexp(length(ids), rate=1)</code>. For a constant infectious period of length <code>c</code>, it is sufficient to set <code>function(x) {c}</code>.</p> <p>For the <code>simulate</code> method of class "twinSIR" only, this can also be <code>NULL</code> (the default), which means that the observed infectious periods of infected individuals are re-used when simulating a new epidemic and individuals with missing infectious periods (i.e. infection and recovery was not observed) are attributed to the mean observed infectious period.</p> <p>Note that it is even possible to simulate an SI-epidemic by setting <code>infPeriod = function(x) {Inf}</code>: once an individual became infected it spreads the disease forever, i.e. it will never be removed.</p>
remPeriod	a function generating lengths of removal periods. Per default, once an individual was removed it will stay in this state forever ( <code>Inf</code> ). Therefore, it will not become at-risk (S) again and re-infections are not possible. Alternatively, always returning 0 as length of the removal period corresponds to a SIS epidemic. Any other values correspond to SIRS.
end	<p>a single positive numeric value specifying the time point at which the simulation should be forced to end. By default, this is <code>Inf</code>, i.e. the simulation continues until there is no susceptible individual left.</p> <p>For the <code>simulate</code> method of class "twinSIR" the default is to have equal simulation and observation periods.</p>
trace	logical (or integer) indicating if (or how often) the sets of susceptible and infected individuals as well as the rejection indicator (of the rejection sampling step) should be <code>cated</code> . Defaults to <code>FALSE</code> .
.allocate	number of blocks to initially allocate for the event history; defaults to 500, i.e. $500 \times N$ rows. Each time the simulated epidemic exceeds the allocated space, the event history will be enlarged by <code>.allocate</code> blocks.
object	an object of class "twinSIR". This must contain the original data used for the fit (see <code>data</code> ).
nsim	number of epidemics to simulate. Defaults to 1.
seed	an integer that will be used in the call to <code>set.seed</code> before simulating the epidemics.
...	unused (argument of the generic).

## Details

A model is specified through the `formula`, which has the form `cbind(start, stop) ~ cox(endemicVar1) * cox(endemicVar2)`, i.e. the right hand side has the usual form as in `lm`, but all variables are marked as being endemic by the special function `cox`. The effects of those predictor terms are specified by `beta`. The left hand side of the formula denotes the start and stop columns in `data`. This can be omitted, if `data` inherits from class `"epidata"` in which case `cbind(start, stop)` will be used. The epidemic model component is specified by the parameter `f` (and by the epidemic effects `alpha`).

The simulation algorithm used is *Ogata's modified thinning*. For details, see Höhle (2008), Section 4.

## Value

An object of class `"simEpidata"`, which is a `data.frame` with the columns `"id"`, `"start"`, `"stop"`, `"atRiskY"`, `"event"`, `"Revent"` and the coordinate columns (with the original names from `data`), which are all obligatory. These columns are followed by all the variables appearing on the rhs of the `formula`. Last but not least, the generated columns with epidemic variables corresponding to the functions in the list `f` are appended, if `length(f) > 0`. Note that objects of class `"simEpidata"` also inherit from class `"epidata"`, thus all `"epidata"` methods can be applied.

The `data.frame` is given the additional *attributes*

`"eventTimes"` numeric vector of infection time points (sorted chronologically).  
`"timeRange"` numeric vector of length 2: `c(min(start), max(stop))`.  
`"coords.cols"`  
 numeric vector containing the column indices of the coordinate columns in the resulting data-frame.  
`"f"` this equals the argument `f`.  
`"config"` a list with elements `h0 = h0$exact`, `beta` and `alpha`.  
`call` the matched call.  
`terms` the `terms` object used.

## Author(s)

Sebastian Meyer and Michael Höhle

## References

Höhle, M. (2009), Additive-Multiplicative Regression Models for Spatio-Temporal Epidemics, Accepted for publication in the Biometrical Journal.

Höhle, Michael (2008) Spatio-temporal epidemic modelling using additive-multiplicative intensity models. *Ludwig-Maximilians-Universität, Department of Statistics: Technical Reports*, No. 41. Available at <http://epub.ub.uni-muenchen.de/6366/>.

**See Also**

The `plot` method and the function `animate` for plotting and animation of epidemic data, respectively. `intensityPlot` for plotting the path of the infection intensity.

Function `twinSIR` for fitting spatio-temporal epidemic intensity models to epidemic data.

**Examples**

```

set.seed(1234)
# Generate a data frame containing a hypothetical population with 100 individuals
n <- 100
pos <- matrix(rnorm(n*2), ncol=2, dimnames=list(NULL, c("x", "y")))
pop <- data.frame(id=1:n,x=pos[,1], y=pos[,2],
                 gender=sample(0:1, n, replace=TRUE),
                 I0col=rep(0,n),start=rep(0,n),stop=rep(Inf,n))

epi <- simEpidata(cbind(start,stop) ~ cox(gender),
                 data = pop,
                 id = "id", I0.col = "I0col", coords.cols = c("x","y"),
                 beta = c(-3), h0 = -2, alpha = c(B1 = 0.1),
                 f = list(B1 = function(u) u <= 1),
                 infPeriod = function(ids) rexp(length(ids), rate=1))

plot(epi)

# load data of an observed epidemic
data("fooePIData")
summary(fooePIData)

# simulate a new evolution of the epidemic
simepi <- simEpidata(cbind(start, stop) ~ cox(z1) + cox(z1):cox(z2),
                   data = fooePIData,
                   beta = c(1,0.5), h0 = -7, alpha = c(B2 = 0.1, B1 = 0.05),
                   f = list(B1 = function(u) u<=1, B2 = function(u) u>1 & is.finite(u)),
                   infPeriod = function(ids) rexp(length(ids), rate=1), trace = FALSE)
summary(simepi)
plot(simepi)
## Not run:
animate(simepi)
## End(Not run)
intensityPlot(simepi)

# load a fitted model object, which must contain the original data
# (use 'keep.data = TRUE' in the call of 'twinSIR')
data("foofit")
foofit

# plot original epidemic
plot(foofit$data)

## simulate a new epidemic using the model and parameter estimates of 'foofit'
## and set simulation period = observation period
# with observed infPeriods:
simfitepi1 <- simulate(foofit, nsim = 1)[[1]]

```

```
plot(simfitepi1)
# with new infPeriods:
simfitepi2 <- simulate(foofit, nsim = 1,
                      infPeriod=function(ids) rexp(length(ids), rate=0.3))[[1]]
plot(simfitepi2)
```

# Index

- \*Topic **aplot**
  - twinSIR\_intensityPlot, 41
- \*Topic **classes**
  - LBExperiment-class, 7
  - LBInference-class, 8
  - LBInferenceMCMC-class, 9
  - LBInferenceML-class, 11
  - LBInferenceMLK-class, 12
  - LBLayouT-class, 13
  - LBOptions-class, 14
  - LBOptionsMCMC-class, 17
  - LBOptionsML-class, 20
  - twinSIR\_epidata, 30
- \*Topic **datagen**
  - twinSIR\_simulation, 47
- \*Topic **datasets**
  - abakaliki, 3
  - csfv, 3
  - hksars, 4
  - laevens, 6
  - oneill, 23
- \*Topic **dplot**
  - twinSIR\_intensityPlot, 41
  - twinSIR\_profile, 45
- \*Topic **dynamic**
  - twinSIR\_epidata\_animate, 33
- \*Topic **hplot**
  - twinSIR\_epidata\_animate, 33
  - twinSIR\_epidata\_plot, 37
  - twinSIR\_intensityPlot, 41
- \*Topic **htest**
  - seir, 24
  - twinSIR\_methods, 43
  - twinSIR\_profile, 45
- \*Topic **manip**
  - twinSIR\_epidata, 30
  - twinSIR\_epidata\_intersperse, 36
- \*Topic **methods**
  - twinSIR\_epidata\_plot, 37
  - twinSIR\_epidata\_summary, 39
  - twinSIR\_intensityPlot, 41
  - twinSIR\_methods, 43
  - twinSIR\_profile, 45
- \*Topic **models**
  - twinSIR, 25
  - twinSIR\_simulation, 47
- \*Topic **optimize**
  - twinSIR, 25
  - twinSIR\_profile, 45
- \*Topic **package**
  - RLadyBug-package, 2
- \*Topic **print**
  - twinSIR\_methods, 43
- \*Topic **spatial**
  - twinSIR\_epidata, 30
  - twinSIR\_epidata\_animate, 33
  - twinSIR\_epidata\_intersperse, 36
  - twinSIR\_epidata\_plot, 37
- \*Topic **utilities**
  - ladybugExample, 5
  - readSpecFile, 23
  - [.data.frame, 32
  - [.epidata (*twinSIR\_epidata*), 30
- abakaliki, 3
- AIC, 29, 45
- AIC.twinSIR (*twinSIR\_methods*), 43
- algo (*LBOptionsMCMC-class*), 17
- algo, LBOptionsMCMC-method (*LBOptionsMCMC-class*), 17
- algo<- (*LBOptionsMCMC-class*), 17
- algo<-, LBOptionsMCMC-method (*LBOptionsMCMC-class*), 17
- animate, 33, 39, 51
- animate (*twinSIR\_epidata\_animate*), 33

- as.epidata, 28, 29, 40
- as.epidata (*twinSIR\_epidata*), 30
- class, 28, 30, 37, 39
- coef, 29
- coef.twinSIR (*twinSIR\_methods*), 43
- cox, 27, 50
- csfv, 3
- csfvML (*csfv*), 3
- csfvTDprior (*csfv*), 3
- data.frame, 30, 32
- data2events (*LBExperiment-class*), 7
- data2events, *LBExperiment-method* (*LBExperiment-class*), 7
- dev.print, 35
- epidata, 26, 37, 39, 47, 50
- epidata (*twinSIR\_epidata*), 30
- extractAIC, 29, 45
- extractAIC.twinSIR, 44
- extractAIC.twinSIR (*twinSIR\_methods*), 43
- formula, 26, 47
- hksars, 4
- hksars.opts.ml (*hksars*), 4
- ignoreData (*LBOptions-class*), 14
- ignoreData, *LBOptions-method* (*LBOptions-class*), 14
- ignoreData<- (*LBOptions-class*), 14
- ignoreData<-, *LBOptions-method* (*LBOptions-class*), 14
- infValues (*LBInference-class*), 8
- infValues, *LBInference-method* (*LBInference-class*), 8
- infValues, *LBInferenceMCMC-method* (*LBInferenceMCMC-class*), 9
- infValues, *LBInferenceML-method* (*LBInferenceML-class*), 11
- infValues, *LBInferenceMLK-method* (*LBInferenceMLK-class*), 12
- infValues<- (*LBInference-class*), 8
- infValues<-, *LBInference-method* (*LBInference-class*), 8
- infValues<-, *LBInferenceMCMC-method* (*LBInferenceMCMC-class*), 9
- infValues<-, *LBInferenceML-method* (*LBInferenceML-class*), 11
- infValues<-, *LBInferenceMLK-method* (*LBInferenceMLK-class*), 12
- initBeta (*LBOptions-class*), 14
- initBeta, *LBOptions-method* (*LBOptions-class*), 14
- initBeta<- (*LBOptions-class*), 14
- initBeta<-, *LBOptions-method* (*LBOptions-class*), 14
- initBetaN (*LBOptions-class*), 14
- initBetaN, *LBOptions-method* (*LBOptions-class*), 14
- initBetaN<- (*LBOptions-class*), 14
- initBetaN<-, *LBOptions-method* (*LBOptions-class*), 14
- initDia (*LBOptions-class*), 14
- initDia, *LBOptions-method* (*LBOptions-class*), 14
- initDia<- (*LBOptions-class*), 14
- initDia<-, *LBOptions-method* (*LBOptions-class*), 14
- initialize, *LBInferenceMCMC-method* (*LBInferenceMCMC-class*), 9
- initialize, *LBOptions-method* (*LBOptions-class*), 14
- initIncu (*LBOptions-class*), 14
- initIncu, *LBOptions-method* (*LBOptions-class*), 14
- initIncu<- (*LBOptions-class*), 14
- initIncu<-, *LBOptions-method* (*LBOptions-class*), 14
- initInf (*LBOptions-class*), 14
- initInf, *LBOptions-method* (*LBOptions-class*), 14
- initInf<- (*LBOptions-class*), 14
- initInf<-, *LBOptions-method* (*LBOptions-class*), 14
- initsAsDataFrame (*LBOptions-class*), 14
- initsAsDataFrame, *LBOptions-method* (*LBOptions-class*), 14
- intensityPlot, 51
- intensityPlot (*twinSIR\_intensityPlot*), 41
- intersperse (*twinSIR\_epidata\_intersperse*), 36

- knots, 48
- ladybugExample, 5
- laevens, 6
- laevens.opts.mcmc (*laevens*), 6
- laevensML (*laevens*), 6
- layoutAsDataFrame
  - (*LBLayout-class*), 13
- layoutAsDataFrame, *LBLayout*-method
  - (*LBLayout-class*), 13
- layoutMatrixes (*LBLayout-class*), 13
- layoutMatrixes, *LBLayout*-method
  - (*LBLayout-class*), 13
- LBExperiment-class, 13
- LBExperiment-class, 7
- LBInference-class, 10, 11, 25
- LBInference-class, 8
- LBInferenceMCMC-class, 6, 9
- LBInferenceMCMC-class, 9
- LBInferenceML-class, 9, 12
- LBInferenceML-class, 11
- LBInferenceMLK-class, 12
- LBInits (*LBOptions-class*), 14
- LBInits, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBInits<- (*LBOptions-class*), 14
- LBInits<-, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBLayout-class, 13
- LBModel (*LBOptions-class*), 14
- LBModel, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBModel<- (*LBOptions-class*), 14
- LBModel<-, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBOptions (*LBOptions-class*), 14
- LBOptions, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBOptions, *LBOptionsMCMC*-method
  - (*LBOptionsMCMC-class*), 17
- LBOptions-class, 20, 22, 25
- LBOptions-class, 14
- LBOptions<- (*LBOptions-class*), 14
- LBOptions<-, *LBOptions*-method
  - (*LBOptions-class*), 14
- LBOptions<-, *LBOptionsMCMC*-method
  - (*LBOptionsMCMC-class*), 17
- LBOptionsMCMC-class, 17, 22, 25
- LBOptionsMCMC-class, 17
- LBOptionsML-class, 17, 20, 25
- LBOptionsML-class, 20
- legend, 34, 35, 38
- lm, 27, 50
- logLik, 29
- logLik.twinSIR (*twinSIR\_methods*), 43
- matplot, 42
- matrix, 30
- oneill, 23
- optim, 25, 26, 46
- optionsAsDataFrame
  - (*LBOptions-class*), 14
- optionsAsDataFrame, *LBOptions*-method
  - (*LBOptions-class*), 14
- optionsAsDataFrame, *LBOptionsMCMC*-method
  - (*LBOptionsMCMC-class*), 17
- par, 34, 37
- plot, 33, 37, 51
- plot, *LBExperiment*, missing-method
  - (*LBExperiment-class*), 7
- plot, *LBExperiment*-method
  - (*LBExperiment-class*), 7
- plot, *LBInferenceMCMC*, missing-method
  - (*LBInferenceMCMC-class*), 9
- plot.epidata, 35
- plot.epidata
  - (*twinSIR\_epidata\_plot*), 37
- plot.function, 38
- plot.stepfun, 38
- plot.summary.epidata
  - (*twinSIR\_epidata\_plot*), 37
- plot.twinSIR, 29
- plot.twinSIR
  - (*twinSIR\_intensityPlot*), 41
- points, 34
- print, 39
- print.data.frame, 32
- print.epidata (*twinSIR\_epidata*), 30
- print.summary.epidata
  - (*twinSIR\_epidata\_summary*), 39
- print.twinSIR (*twinSIR\_methods*), 43

- printCoefmat, 44  
 profile.twinSIR, 29  
 profile.twinSIR  
     (*twinSIR\_profile*), 45  
  
 R0 (*LBInferenceMCMC-class*), 9  
 R0, *LBInferenceMCMC*-method  
     (*LBInferenceMCMC-class*), 9  
 R0, *LBInferenceML*-method  
     (*LBInferenceML-class*), 11  
 randomWalk (*LBOptionsMCMC-class*),  
     17  
 randomWalk, *LBOptionsMCMC*-method  
     (*LBOptionsMCMC-class*), 17  
 randomWalk<-  
     (*LBOptionsMCMC-class*), 17  
 randomWalk<-, *LBOptionsMCMC*-method  
     (*LBOptionsMCMC-class*), 17  
 readSpecFile, 23  
 RLadyBug (*RLadyBug-package*), 2  
 RLadyBug-package, 2  
 rug, 38, 42  
  
 samplePaths  
     (*LBInferenceMCMC-class*), 9  
 samplePaths, *LBInferenceMCMC*-method  
     (*LBInferenceMCMC-class*), 9  
 seed (*LBOptions-class*), 14  
 seed, *LBOptions*-method  
     (*LBOptions-class*), 14  
 seed<- (*LBOptions-class*), 14  
 seed<-, *LBOptions*-method  
     (*LBOptions-class*), 14  
 seir, 24  
 set.seed, 49  
 show, *LBExperiment*-method  
     (*LBExperiment-class*), 7  
 show, *LBInference*-method  
     (*LBInference-class*), 8  
 show, *LBInferenceMCMC*-method  
     (*LBInferenceMCMC-class*), 9  
 show, *LBInferenceML*-method  
     (*LBInferenceML-class*), 11  
 show, *LBInferenceMLK*-method  
     (*LBInferenceMLK-class*), 12  
 show, *LBLayout*-method  
     (*LBLayout-class*), 13  
 show, *LBOptions*-method  
     (*LBOptions-class*), 14  
  
 show, *LBOptionsMCMC*-method  
     (*LBOptionsMCMC-class*), 17  
 show, *LBOptionsML*-method  
     (*LBOptionsML-class*), 20  
 simulate, 27, 47  
 simulate, *LBOptions*-method  
     (*LBOptions-class*), 14  
 simulate.twinSIR, 29  
 simulate.twinSIR  
     (*twinSIR\_simulation*), 47  
 stateplot (*twinSIR\_epidata\_plot*),  
     37  
 stepfun, 48  
 summary, 33, 39  
 summary, *LBExperiment*-method  
     (*LBExperiment-class*), 7  
 summary, *LBInference*-method  
     (*LBInference-class*), 8  
 summary, *LBInferenceMCMC*-method  
     (*LBInferenceMCMC-class*), 9  
 summary, *LBInferenceML*-method  
     (*LBInferenceML-class*), 11  
 summary, *LBInferenceMLK*-method  
     (*LBInferenceMLK-class*), 12  
 summary, *LBLayout*-method  
     (*LBLayout-class*), 13  
 summary, *LBOptions*-method  
     (*LBOptions-class*), 14  
 summary, *LBOptionsMCMC*-method  
     (*LBOptionsMCMC-class*), 17  
 summary, *LBOptionsML*-method  
     (*LBOptionsML-class*), 20  
 summary.epidata, 32, 35, 39  
 summary.epidata  
     (*twinSIR\_epidata\_summary*),  
     39  
 summary.twinSIR  
     (*twinSIR\_methods*), 43  
 Sys.sleep, 34  
  
 title, 34  
 twinSIR, 25, 30, 33, 41, 42, 47, 51  
 twinSIR\_epidata, 30  
 twinSIR\_epidata\_animate, 33  
 twinSIR\_epidata\_intersperse, 36  
 twinSIR\_epidata\_plot, 37

twinSIR\_epidata\_summary, [39](#)  
twinSIR\_intensityPlot, [41](#)  
twinSIR\_methods, [43](#)  
twinSIR\_profile, [45](#)  
twinSIR\_simulation, [47](#)

vcov, [29](#)  
vcov.twinSIR(*twinSIR\_methods*), [43](#)

writeOptionFile  
    (*LBOptionsMCMC-class*), [17](#)  
writeOptionFile, LBOptionsMCMC-method  
    (*LBOptionsMCMC-class*), [17](#)  
writeOptionFile, LBOptionsML-method  
    (*LBOptionsML-class*), [20](#)