Package: spaero (via r-universe)

November 1, 2024

Title Software for Project AERO

Version 0.6.0

Description Implements methods for anticipating the emergence and eradication of infectious diseases from surveillance time series. Also provides support for computational experiments testing the performance of such methods.

Depends R (>= 3.2.1)

Imports stats, utils

License GPL (>= 2) | file LICENSE

LazyData true

Suggests covr (>= 3.0.0), earlywarnings (>= 1.0.59), knitr (>= 1.11), moments (>= 0.14), np (>= 0.60.2), pomp (>= 2.1), rmarkdown (>= 0.9.2), testthat (>= 0.11.0)

BugReports `https://github.com/e3bo/spaero/issues/`

RoxygenNote 7.1.1

VignetteBuilder knitr

Encoding UTF-8

Repository https://e3bo.r-universe.dev

RemoteUrl https://github.com/e3bo/spaero

RemoteRef HEAD

RemoteSha 37099fab35d0399363164c21bc217fbd1080204a

Contents

create_simulator *Create surveillance data simulator.*

Description

`create_simulator` creates a pomp object that will run simulations of an SIR or SIS model according to Gillespie's direct method and generate simulated observations of the process.

Usage

```
create_simulator(
  times = seq(0, 9),
  t0 = min(times),
  process_model = c("SIR", "SIS"),
  transmission = c("density-dependent", "frequency-dependent"),
 params = c(gamma = 24, mu = 1/70, d = 1/70, eta = 1e-05, beta_par = 1e-04, rho = 0.1,
    S_0 = 1, I_0 = 0, R_0 = 0, N_0 = 1e+05, p = 0),
 covar = data.frame(gamma_t = c(0, 0), mu_t = c(0, 0), d_t = c(0, 0), eta_t = c(0, 0),
    beta_par_t = c(0, 0), p_t = c(0, 0), time = c(0, 1e+06))
)
```

Arguments

`times`	A numeric vector of increasing times at which the state of the simulation will be sampled.
`t0`	The time at which the simulation is started with state variable set to the initial conditions specified via params.
`process_model`	Character string giving the process model. Allowed values are '"SIR"' and '"SIS"'.
`transmission`	Character string describing the transmission model. Allowed values are '"density-dependent"' and '"frequency-dependent"'.
`params`	A named numeric vector of parameter values and initial conditions.
`covar`	A data frame containing values of the time-dependent components of the parameters.

Details

See the vignette "Getting Started with spaero" for a description of the model. The "params" argument must include all model parameters. These will become the default parameters for the model object. They can be overridden when the simulation is run via the "params" argument of `pomp::simulate`. The case is the same for the "times" argument. The "covar" argument should be a data frame with a column named for each of the time-dependent parameters and a column named time. This data frame describes the time series of each of the time-dependent parameters. In the simulation, interpolation based on this data frame determines the value of these parameters at specific instants in time. The user must ensure that these values result in the parameters remaining non-negative for the course of the simulation.

Value

A pomp object with which simulations can be run via `pomp::simulate`.

See Also

`pomp` for documentation of pomp objects

Examples

```
if (requireNamespace("pomp", quietly = TRUE)) {
  foo <- create_simulator()
  out <- pomp::simulate(foo, times = seq(0, 20, by = 1/26))
  out <- as(out, "data.frame")
  head(out)

  opar <- par(mfrow = c(2, 1))
  plot((S/N)~time, data = out, type = "l")
  plot(cases~time, data = out, type = "l")
  par(opar)
}
```

`get_stats` *Get estimates of time-dependent properties of models.*

Description

`get_stats` estimates time-dependent properties of models (e.g., variance) from ensemble time series.

Usage

```
get_stats(
  x,
  center_trend = "grand_mean",
  center_kernel = c("gaussian", "uniform"),
  center_bandwidth = NULL,
  stat_trend = c("local_constant", "local_linear"),
  stat_kernel = c("uniform", "gaussian"),
  stat_bandwidth = NULL,
  lag = 1,
  backward_only = FALSE
)
```

Arguments

`x`	A univariate or multivariate numeric time series object or a numeric vector or matrix.
`center_trend`	Character string giving method of calculating the trend to subtract. Allowed values are '"assume_zero"', '"grand_mean"', '"ensemble_means"', '"local_constant"', and '"local_linear"'. Will be partially matched.
`center_kernel`	Character string giving the kernel for any local detrending. Allowed values are '"gaussian"' and '"uniform"'.
`center_bandwidth`	Bandwidth of kernel for any local detrending done. A numeric value >= 1.
`stat_trend`	Character string giving method of smoothing estimates. Allowed values are '"local_constant"', and '"local_linear"'. Will be partially matched.
`stat_kernel`	Character string giving the kernel for local smoothing of estimates. Allowed values are '"gaussian"' and '"uniform"'.
`stat_bandwidth`	Bandwidth of kernel for local smoothing of estimates. A numeric value >= 1.
`lag`	Integer lag at which to calculate the acf. This lag is in terms of the index of `x` and does not account for the frequency of `x` if `x` is a time series. It should be non-negative.
`backward_only`	Logical value (defaulting to 'FALSE') that determines whether any uniform smoothing kernels are restricted to using data before the index of the smoothed estimate.

Details

Any missing values in 'x' will cause an error.

Bandwidths affect weights in local smoothers as follows. To get the local estimate corresponding to index i, the distance to each other index j is calculated as (i - j) / h, where h is the bandwidth. Then that distance is plugged into the kernel function to obtain a weight. The weights are normalized to sum to one for each index.

The gaussian kernel is equivalent to a standard Gaussian density function. The uniform kernel is an indicator function of whether the distance is less than 1. Thus selecting a uniform kernel with a bandwidth of 2 is equivalent to a sliding window of length 3 that is centered on the focal index. In general, if n is the greatest integer that is less than the value of the bandwidth h, the window includes the n nearest values on each side of the focal index.

'"local_constant"' smoothers are local means computed with the kernel weights. '"local_linear"' smoothers are the fitted values of local linear regressions with the kernel weights. The linear smoothers avoid biases that the one-sided kernels at the ends of the time series can create for the local constant smoothers.

See the vignette "Getting Started with spaero" for the formulas used for each estimate.

Value

A list with elements '"stats"', '"taus"', '"centered"', '"stat_trend"', '"stat_kernel"', '"stat_bandwidth"', and '"lag"'. "stats" is a list containing vectors of the estimates. '"taus"' is a list containing Kendall's correlation coefficient of each element of '"stats"' with time. '"centered"' is a list of the detrended time series, the trend subtracted, and the bandwidth used in the detrending. The other elements record the parameters provided to this function for future reference.

See Also

`acf`, `var`, `kurtosis`, and `skewness` for estimation of properties that are not time-dependent. See `generic_ews` for another approach to estimation of time-dependent properties.

Examples

```
# A highly autocorrelated time series
x <- 1:10
get_stats(x, stat_bandwidth = 3)$stats

# Plot log of acf
plot(log(get_stats(x, stat_bandwidth = 3)$stats$autocor))

# Check estimates with AR1 simulations with lag-1 core 0.1
w <- rnorm(1000)
xnext <- function(xlast, w) 0.1 * xlast + w
x <- Reduce(xnext, x = w, init = 0, accumulate = TRUE)
acf(x, lag.max = 1, plot = FALSE)
head(get_stats(x, stat_bandwidth = length(x))$stats$autocor)

# Check detrending ability
x2 <- x + seq(1, 10, len = length(x))
ans <- get_stats(x2, center_trend = "local_linear",
                  center_bandwidth = length(x),
                   stat_bandwidth = length(x))$stats
head(ans$autocor)

# The simple acf estimate is inflated by the trend
acf(x2, lag.max = 1, plot = FALSE)

# Check ability to estimate time-dependent autocorrelation
xnext <- function(xlast, w) 0.8 * xlast + w
xhi <- Reduce(xnext, x = w, init = 0, accumulate = TRUE)
acf(xhi, lag.max = 1, plot = FALSE)
wt <- seq(0, 1, len = length(x))
xdynamic <- wt * xhi + (1 - wt) * x
get_stats(xdynamic, stat_bandwidth = 100)$stats$autocor
```

Index