Package 'fastbeta'

March 24, 2025

Version 0.3.2

Date 2025-03-23

Title Fast Approximation of Time-Varying Infectious Disease Transmission Rates

Description A fast method for approximating time-varying infectious disease transmission rates from disease incidence time series and other data, based on a discrete time approximation of an SEIR model, as analyzed in Jagan et al. (2020) <doi:10.1371/journal.pcbi.1008124>.

License GPL (>= 2)

URL https://github.com/davidearn/fastbeta

BugReports https://github.com/davidearn/fastbeta/issues

Depends R (>= 4.3)

Imports grDevices, graphics, stats

Suggests adaptivetau, deSolve, tools, utils

BuildResaveData no

NeedsCompilation yes

Author Mikael Jagan [aut, cre] (<https://orcid.org/0000-0002-3542-2938>)

Maintainer Mikael Jagan <jaganmn@mcmaster.ca>

Repository CRAN

Date/Publication 2025-03-24 04:10:02 UTC

Contents

fastbeta-package 2
deconvolve 2
fastbeta 4
fastbeta.bootstrap 6
fastbeta.matrix 8
ptpi 9
seir 12

seir.auxiliary 15
seir.canonical 16
seir.library 19
smallpox 20

Index **22**

`fastbeta-package` R *Package* **fastbeta**

Description

An R package for approximating time-varying infectious disease transmission rates from disease incidence time series and other data.

Details

The "main" function is `fastbeta`.

To render a list of available help topics, use `help(package = "fastbeta")`.

To report a bug or request a change, use `bug.report(package = "fastbeta")`.

Author(s)

Mikael Jagan <jaganmn@mcmaster.ca>

`deconvolve` *Richardson-Lucy Deconvolution*

Description

Performs a modified Richardson-Lucy iteration for the purpose of estimating incidence from reported incidence or mortality, conditional on a reporting probability and on a distribution of the time to reporting.

Usage

```
deconvolve(x, prob = 1, delay = 1,
           start, tol = 1, iter.max = 32L, complete = FALSE)
```

Arguments

x	a numeric vector of length n giving the number of infections or deaths reported during n observation intervals of equal duration.
prob	a numeric vector of length d+n such that prob[d+i] is the probability that an infection during interval i is eventually reported. prob of length 1 is recycled.
delay	a numeric vector of length d+1 such that delay[j] is the probability that an infection during interval i is reported during interval i+j-1, given that it is eventually reported. delay need not sum to 1 but must not sum to 0.
start	a numeric vector of length d+n giving a starting value for the iteration. start[d+i] estimates the expected number of infections during interval i that are eventually reported. If missing, then a starting value is generated by padding x on the left and right with d-d0 and d0 zeros, choosing d0 = which.max(delay)-1.
tol	a tolerance indicating a stopping condition; see the reference.
iter.max	the maximum number of iterations.
complete	a logical flag indicating if the result should preserve successive updates to start.

Value

A list with elements:

value	the result of updating start iter times then dividing by prob. If complete = TRUE, then value is a (d+n)-by-(1+iter) matrix containing start and the iter successive updates, each divided by prob.
chisq	the chi-squared statistics corresponding to value.
iter	the number of iterations performed.

References

Goldstein, E., Dushoff, J., Ma, J., Plotkin, J. B., Earn, D. J. D., & Lipsitch, M. (2020). Reconstructing influenza incidence by deconvolution of daily mortality time series. *Proceedings of the National Academy of Sciences U. S. A.*, *106*(51), 21825-21829. doi:10.1073/pnas.0902958106

Examples

```
set.seed(2L)
n <- 200L
d <- 50L
p <- 0.1
prob <- plogis(rlogis(d + n, location = qlogis(p), scale = 0.1))
delay <- diff(pgamma(0L:(d + 1L), 12, 0.4))

h <- function (x, a = 1, b = 1, c = 0) a * exp(-b * (x - c)^2)
ans <- floor(h(seq(-60, 60, length.out = d + n), a = 1000, b = 0.001))

x0 <- rbinom(d + n, ans, prob)
x <- tabulate(rep(1L:(d + n), x0) +
              sample(0L:d, size = sum(x0), replace = TRUE, prob = delay),
              d + n)[-(1L:d)]
```

```
str(D0 <- deconvolve(x, prob, delay, complete = FALSE))
str(D1 <- deconvolve(x, prob, delay, complete =  TRUE))

matplot(-(d - 1L):n,
        cbind(x0, c(rep(NA, d), x), prob * D0[["value"]], p * ans),
        type = c("p", "p", "p", "l"),
        col = c(1L, 1L, 2L, 4L), pch = c(16L, 1L, 16L, NA),
        lty = c(0L, 0L, 0L, 1L), lwd = c(NA, NA, NA, 3L),
        xlab = "Time", ylab = "Count")
legend("topleft", NULL,
       c("actual", "actual+delay", "actual+delay+deconvolution", "p*h"),
       col = c(1L, 1L, 2L, 4L), pch = c(16L, 1L, 16L, NA),
       lty = c(0L, 0L, 0L, 1L), lwd = c(NA, NA, NA, 3L),
       bty = "n")

plot(0L:D1[["iter"]], D1[["chisq"]], xlab = "Iterations", ylab = quote(chi^2))
abline(h = 1, lty = 2L)
```

`fastbeta` *Estimate a Time-Varying Infectious Disease Transmission Rate*

Description

Generates a discrete approximation of a time-varying infectious disease transmission rate from an equally spaced disease incidence time series and other data.

Usage

```
fastbeta(series, sigma = 1, gamma = 1, delta = 0,
         m = 1L, n = 1L, init, ...)
```

Arguments

`series`	a "multiple time series" object, inheriting from class `mts`, with three columns storing ("parallel", equally spaced) time series of incidence, births, and the per capita natural mortality rate, in that order.
`sigma`, `gamma`, `delta`	non-negative numbers. `m*sigma`, `n*gamma`, and `delta` are the rates of removal from each latent, infectious, and recovered compartment.
`m`	a non-negative integer indicating a number of latent stages.
`n`	a positive integer indicating a number of infectious stages.
`init`	a numeric vector of length `1+m+n+1` giving an initial state with compartments ordered as (S, E, I, R).
`...`	optional arguments passed to `deconvolve`, if the first column of `series` represents *observed* incidence rather than actual or estimated incidence.

Details

The algorithm implemented by `fastbeta` is based on an SEIR model with

- m latent stages (E^i, $i = 1, \ldots, m$);
- n infectious stages (I^j, $j = 1, \ldots, n$);
- time-varying rates β, ν, and μ of transmission, birth, and natural death; and
- constant rates $m\sigma$, $n\gamma$, and δ of removal from each latent, infectious, and recovered compartment, where removal from the recovered compartment implies return to the susceptible compartment (loss of immunity).

It is derived by linearizing of the system of ordinary differential equations

$$\begin{aligned}
\mathrm{d}S \quad /\mathrm{d}t &= \quad \delta R \quad - (\lambda(t) + \mu(t))S \quad + \nu(t) \\
\mathrm{d}E^1 \quad /\mathrm{d}t &= \lambda(t)S \quad - (\ m\sigma + \mu(t))E^1 \\
\mathrm{d}E^{i+1}/\mathrm{d}t &= \ m\sigma E^i \quad - (\ m\sigma + \mu(t))E^{i+1} \\
\mathrm{d}I^1 \quad /\mathrm{d}t &= \ m\sigma E^m - (\ \ n\gamma + \mu(t))I^1 \\
\mathrm{d}I^{j+1}/\mathrm{d}t &= \ \ n\gamma I^j \quad - (\ \ n\gamma + \mu(t))I^{j+1} \\
\mathrm{d}R \quad /\mathrm{d}t &= \ \ n\gamma I^n \quad - (\quad \delta + \mu(t))R
\end{aligned} \qquad \lambda(t) = \beta(t) \sum_j I^j$$

and substituting actual or estimated incidence and births for definite integrals of λS and ν. This procedure yields a system of linear difference equations from which one recovers a discrete approximation of β:

$$\begin{aligned}
E^1_{t+1} &= [(1 - \tfrac{1}{2}(m\sigma + \mu_t))E^1_t \qquad\qquad\qquad + Z_{t+1} \qquad\quad]/[1 + \tfrac{1}{2}(m\sigma + \mu_{t+1})] \\
E^{i+1}_{t+1} &= [(1 - \tfrac{1}{2}(m\sigma + \mu_t))E^{i+1}_t + \tfrac{1}{2}m\sigma(E^i_t \ + E^i_{t+1}) \qquad\quad]/[1 + \tfrac{1}{2}(m\sigma + \mu_{t+1})] \\
I^1_{t+1} &= [(1 - \tfrac{1}{2}(\ n\gamma + \mu_t))I^1_t \quad + \tfrac{1}{2}m\sigma(E^m_t + E^m_{t+1}) \qquad\quad]/[1 + \tfrac{1}{2}(\ n\gamma + \mu_{t+1})] \\
I^{j+1}_{t+1} &= [(1 - \tfrac{1}{2}(\ n\gamma + \mu_t))I^{j+1}_t + \tfrac{1}{2}\ n\gamma(I^j_t \ + I^j_{t+1}\ ) \qquad\quad]/[1 + \tfrac{1}{2}(\ n\gamma + \mu_{t+1})] \\
R_{t+1} &= [(1 - \tfrac{1}{2}(\quad \delta + \mu_t))R_t \quad + \tfrac{1}{2}\ n\gamma(I^n_t \ + I^n_{t+1}\ ) \qquad\quad]/[1 + \tfrac{1}{2}(\quad \delta + \mu_{t+1})] \\
S_{t+1} &= [(1 - \tfrac{1}{2}(\qquad \mu_t))S_t \quad + \tfrac{1}{2}\quad \delta(R_t \ + R_{t+1}) - Z_{t+1} + B_{t+1}]/[1 + \tfrac{1}{2}(\qquad \mu_{t+1})]
\end{aligned} \qquad \beta_t = (Z_t + Z_{t+1})/(2S_t \sum_j I$$

where we use the notation

$$X_t \sim X(t) : X = S, E^i, I^j, R, Z, B, \mu, \beta \qquad \begin{aligned} Z(t) &= \int_{t-1}^{t} \lambda(s)S(s)\,\mathrm{d}s \\ B(t) &= \int_{t-1}^{t} \nu(s)\,\mathrm{d}s \end{aligned}$$

and it is understood that the independent variable t is a unitless measure of time relative to the spacing of the substituted time series of incidence and births.

Value

A "multiple time series" object, inheriting from class `mts`, with `1+m+n+1+1` columns (named `S`, `E`, `I`, `R`, and `beta`) storing the result of the iteration described in 'Details'. It is completely parallel to argument `series`, having the same `tsp` attribute.

References

Jagan, M., deJonge, M. S., Krylova, O., & Earn, D. J. D. (2020). Fast estimation of time-varying infectious disease transmission rates. *PLOS Computational Biology*, *16*(9), Article e1008124, 1-39. doi:10.1371/journal.pcbi.1008124

Examples

```
if (requireNamespace("adaptivetau")) withAutoprint({

data(seir.ts02, package = "fastbeta")
a <- attributes(seir.ts02)
str(seir.ts02)
plot(seir.ts02)

## We suppose that we have perfect knowledge of incidence,
## births, and the data-generating parameters
series <- cbind(seir.ts02[, c("Z", "B")], mu = a[["mu"]](0))
colnames(series) <- c("Z", "B", "mu") # FIXME: stats:::cbind.ts mangles dimnames

args <- c(list(series = series),
          a[c("sigma", "gamma", "delta", "m", "n", "init")])
str(args)

X <- do.call(fastbeta, args)
str(X)
plot(X)

plot(X[, "beta"], ylab = "Transmission rate")
lines(a[["beta"]](time(X)), col = "red") # the "truth"

})
```

---

fastbeta.bootstrap *Parametric Bootstrapping*

Description

A simple wrapper around `fastbeta` using it to generate a "primary" estimate of a time-varying transmission rate and `r` bootstrap estimates. Bootstrap estimates are computed for incidence time series simulated using `seir`, with transmission rate defined as the linear interpolant of the primary estimate.

Usage

```
fastbeta.bootstrap(r,
                   series, sigma = 1, gamma = 1, delta = 0,
                   m = 1L, n = 1L, init, ...)
```

Arguments

r	a non-negative integer indicating a number of replications.
series	a "multiple time series" object, inheriting from class mts, with three columns storing ("parallel", equally spaced) time series of incidence, births, and the per capita natural mortality rate, in that order.
sigma, gamma, delta	non-negative numbers. m*sigma, n*gamma, and delta are the rates of removal from each latent, infectious, and recovered compartment.
m	a non-negative integer indicating a number of latent stages.
n	a positive integer indicating a number of infectious stages.
init	a numeric vector of length 1+m+n+1 giving an initial state with compartments ordered as (S, E, I, R).
...	optional arguments passed to seir and/or deconvolve. Both take optional arguments prob and delay. When prob is supplied but not delay, seir and deconvolve receive prob as is. When both are supplied, seir receives prob as is, whereas deconvolve receives prob augmented with length(delay)-1 ones.

Value

A "multiple time series" object, inheriting from class mts, with 1+r columns storing the one primary and r bootstrap estimates. It is completely parallel to argument series, having the same tsp attribute.

Examples

```
if (requireNamespace("adaptivetau")) withAutoprint({

data(seir.ts02, package = "fastbeta")
a <- attributes(seir.ts02)
str(seir.ts02)
plot(seir.ts02)

## We suppose that we have perfect knowledge of incidence,
## births, and the data-generating parameters
series <- cbind(seir.ts02[, c("Z", "B")], mu = a[["mu"]](0))
colnames(series) <- c("Z", "B", "mu") # FIXME: stats:::cbind.ts mangles dimnames

args <- c(list(r = 100L, series = series),
          a[c("sigma", "gamma", "delta", "m", "n", "init")])
str(args)

R <- do.call(fastbeta.bootstrap, args)
str(R)
plot(R)
plot(R, level = 0.95)

})
```

`fastbeta.matrix` *Calculate Coefficient Matrix for Iteration Step*

Description

Calculates the coefficient matrix corresponding to one step of the iteration carried out by `fastbeta`:

```
y <- c(1, E, I, R, S)
for (pos in seq_len(nrow(series) - 1L)) {
    L <- fastbeta.matrix(pos, series, ...)
    y <- L %*% y
}
```

Usage

```
fastbeta.matrix(pos,
                series, sigma = 1, gamma = 1, delta = 0,
                m = 1L, n = 1L)
```

Arguments

`pos`	an integer indexing a row (but not the last row) of `series`.
`series`	a "multiple time series" object, inheriting from class `mts`, with three columns storing ("parallel", equally spaced) time series of incidence, births, and the per capita natural mortality rate, in that order.
`sigma`, `gamma`, `delta`	non-negative numbers. `m*sigma`, `n*gamma`, and `delta` are the rates of removal from each latent, infectious, and recovered compartment.
`m`	a non-negative integer indicating a number of latent stages.
`n`	a positive integer indicating a number of infectious stages.

Value

A lower triangular matrix of size `1+m+n+1+1`.

Examples

```
if (requireNamespace("adaptivetau")) withAutoprint({

data(seir.ts02, package = "fastbeta")
a <- attributes(seir.ts02); p <- length(a[["init"]])
str(seir.ts02)
plot(seir.ts02)

## We suppose that we have perfect knowledge of incidence,
## births, and the data-generating parameters
series <- cbind(seir.ts02[, c("Z", "B")], mu = a[["mu"]](0))
```

```
colnames(series) <- c("Z", "B", "mu") # FIXME: stats:::cbind.ts mangles dimnames

args <- c(list(series = series),
          a[c("sigma", "gamma", "delta", "init", "m", "n")])
str(args)

X <- unclass(do.call(fastbeta, args))[, seq_len(p)]
colnames(X)
Y <- Y. <- cbind(1, X[, c(2L:p, 1L)], deparse.level = 2L)
colnames(Y)

args <- c(list(pos = 1L, series = series),
          a[c("sigma", "gamma", "delta", "m", "n")])
str(args)

L <- do.call(fastbeta.matrix, args)
str(L)
symnum(L != 0)

for (pos in seq_len(nrow(series) - 1L)) {
    args[["pos"]] <- pos
    L. <- do.call(fastbeta.matrix, args)
    Y.[pos + 1L, ] <- L. %*% Y.[pos, ]
}
stopifnot(all.equal(Y, Y.))

})
```

---

ptpi *Peak to Peak Iteration*

Description

Approximates the state of an SEIR model at a reference time from an equally spaced, T-periodic incidence time series and other data. The algorithm relies on a strong assumption: that the incidence time series was generated by the asymptotic dynamics of an SEIR model admitting a locally stable, T-periodic attractor. Hence do interpret with care.

Usage

```
ptpi(series, sigma = 1, gamma = 1, delta = 0,
     m = 1L, n = 1L, init,
     start = tsp(series)[1L], end = tsp(series)[2L],
     tol = 1e-03, iter.max = 32L,
     backcalc = FALSE, complete = FALSE, ...)
```

Arguments

series	a "multiple time series" object, inheriting from class mts, with three columns storing ("parallel", equally spaced) time series of incidence, births, and the per capita natural mortality rate, in that order.
sigma, gamma, delta	non-negative numbers. m*sigma, n*gamma, and delta are the rates of removal from each latent, infectious, and recovered compartment.
m	a non-negative integer indicating a number of latent stages.
n	a positive integer indicating a number of infectious stages.
init	a numeric vector of length 1+m+n+1 giving an initial guess for the state at time start.
start, end	start and end times for the iteration, whose difference should be approximately equal to an integer number of periods. One often chooses the time of the first peak in the incidence time series and the time of the last peak in phase with the first.
tol	a tolerance indicating a stopping condition; see 'Details'.
iter.max	the maximum number of iterations.
backcalc	a logical indicating if the state at time tsp(series)[1] should be back-calculated from the state at time start if that is later.
complete	a logical indicating if intermediate states should be recorded in an array. Useful mainly for didactic or diagnostic purposes.
...	optional arguments passed to deconvolve, if the first column of series represents *observed* incidence rather than actual or estimated incidence.

Details

ptpi can be understood as an iterative application of fastbeta to a subset of series. The basic algorithm can be expressed in R code as:

```
w <- window(series, start, end); i <- nrow(s); j <- seq_along(init)
diff <- Inf; iter <- 0L
while (diff > tol && iter < iter.max) {
    init. <- init
    init <- fastbeta(w, sigma, gamma, delta, m, n, init)[i, j]
    diff <- sqrt(sum((init - init.)^2) / sum(init.^2))
    iter <- iter + 1L
}
value <- init
```

Back-calculation involves solving a linear system of equations; the back-calculated result can mislead if the system is ill-conditioned.

Value

A list with elements:

value	an approximation of the state at time start or at time tsp(series)[1L], depending on backcalc.
diff	the relative difference between the last two approximations.
iter	the number of iterations performed.
x	if complete = TRUE, then a "multiple time series" object, inheriting from class mts, with dimensions c(nrow(w), length(value), iter), where w = window(series, start, end). x[, , k] contains the state at each time(w) in iteration k.

References

Jagan, M., deJonge, M. S., Krylova, O., & Earn, D. J. D. (2020). Fast estimation of time-varying infectious disease transmission rates. *PLOS Computational Biology*, *16*(9), Article e1008124, 1-39. doi:10.1371/journal.pcbi.1008124

Examples

```
if (requireNamespace("deSolve")) withAutoprint({

data(seir.ts01, package = "fastbeta")
a <- attributes(seir.ts01); p <- length(a[["init"]])
str(seir.ts01)
plot(seir.ts01)

## We suppose that we have perfect knowledge of incidence,
## births, and the data-generating parameters, except for
## the initial state, which we "guess"
series <- cbind(seir.ts01[, c("Z", "B")], mu = a[["mu"]](0))
colnames(series) <- c("Z", "B", "mu") # FIXME: stats:::cbind.ts mangles dimnames

plot(series[, "Z"])
start <- 23; end <- 231
abline(v = c(start, end), lty = 2)

set.seed(0L)
args <- c(list(series = series),
          a[c("sigma", "gamma", "delta", "m", "n", "init")],
          list(start = start, end = end, complete = TRUE))
init <- seir.ts01[which.min(abs(time(seir.ts01) - start)), seq_len(p)]
args[["init"]] <- init * rlnorm(p, 0, 0.1)
str(args)

L <- do.call(ptpi, args)
str(L)

S <- L[["x"]][, "S", ]
plot(S, plot.type = "single")
lines(seir.ts01[, "S"], col = "red", lwd = 4) # the "truth"
abline(h = L[["value"]]["S"], v = start, col = "blue", lwd = 4, lty = 2)

## Relative error
L[["value"]] / init - 1
```

```
})
```

`seir` *Simulate Infectious Disease Time Series*

Description

Simulates incidence time series based on an SEIR model with user-defined forcing and a simple model for observation error.

Note that simulation code depends on availability of suggested packages **adaptivetau** and **deSolve**. If the dependency cannot be loaded then an error is signaled.

Usage

```
seir(length.out = 1L,
     beta, nu = function(t) 0, mu = function(t) 0,
     sigma = 1, gamma = 1, delta = 0,
     m = 1L, n = 1L, init,
     stochastic = TRUE, prob = 1, delay = 1,
     aggregate = FALSE, useCompiled = TRUE, ...)

## A basic wrapper for the m=0L case:

 sir(length.out = 1L,
     beta, nu = function(t) 0, mu = function(t) 0,
     gamma = 1, delta = 0,
     n = 1L, init,
     stochastic = TRUE, prob = 1, delay = 1,
     aggregate = FALSE, useCompiled = TRUE, ...)
```

Arguments

`length.out`	a non-negative integer indicating the time series length.
`beta`, `nu`, `mu`	functions of one or more arguments returning transmission, birth, and natural death rates at the time point indicated by the first argument. Arguments after the first must be strictly optional. The functions need not be vectorized.
`sigma`, `gamma`, `delta`	non-negative numbers. `m*sigma`, `n*gamma`, and `delta` are the rates of removal from each latent, infectious, and recovered compartment.
`m`	a non-negative integer indicating a number of latent stages.
`n`	a positive integer indicating a number of infectious stages.
`init`	a numeric vector of length `1+m+n+1` giving an initial state with compartments ordered as (S, E, I, R).
`stochastic`	a logical indicating if the simulation should be stochastic; see 'Details'.

prob	a numeric vector of length n such that prob[i] is the probability that an infection during interval i is eventually observed. prob of length 1 is recycled.
delay	a numeric vector of positive length such that delay[i] is the probability that an infection during interval j is observed during interval j+i-1, given that it is eventually observed. delay need not sum to 1 but must not sum to 0.
aggregate	a logical indicating if latent and infectious compartments should be aggregated.
useCompiled	a logical indicating if derivatives should be computed by compiled C functions rather than by R functions (which *may* be *byte*-compiled). Set to FALSE only if TRUE seems to cause problems, and in that case please report the problems with bug.report(package = "fastbeta").
...	optional arguments passed to lsoda (directly) or ssa.adaptivetau (via its list argument tl.params), depending on stochastic.

Details

Simulations are based on an SEIR model with

- m latent stages (E^i, $i = 1, \ldots, m$);
- n infectious stages (I^j, $j = 1, \ldots, n$);
- time-varying rates β, ν, and μ of transmission, birth, and natural death; and
- constant rates $m\sigma$, $n\gamma$, and δ of removal from each latent, infectious, and recovered compartment, where removal from the recovered compartment implies return to the susceptible compartment (loss of immunity).

seir(stochastic = FALSE) works by numerically integrating the system of ordinary differential equations

$$
\begin{aligned}
\mathrm{d}S \;/\mathrm{d}t &= \quad \delta R \quad - (\lambda(t) + \mu(t))S \quad + \nu(t) \\
\mathrm{d}E^1 \;/\mathrm{d}t &= \lambda(t)S \quad - (\, m\sigma + \mu(t))E^1 \\
\mathrm{d}E^{i+1}/\mathrm{d}t &= m\sigma E^i \quad - (\, m\sigma + \mu(t))E^{i+1} \\
\mathrm{d}I^1 \;/\mathrm{d}t &= m\sigma E^m - (\; n\gamma + \mu(t))I^1 \\
\mathrm{d}I^{j+1}/\mathrm{d}t &= \quad n\gamma I^j \quad - (\; n\gamma + \mu(t))I^{j+1} \\
\mathrm{d}R \;/\mathrm{d}t &= \quad n\gamma I^n \quad - (\quad \delta + \mu(t))R
\end{aligned}
\qquad \lambda(t) = \beta(t) \sum_j I^j
$$

where it is understood that the independent variable t is a unitless measure of time relative to an observation interval. To get time series of incidence and births, the system is augmented with two equations describing *cumulative* incidence and births

$$
\begin{aligned}
\mathrm{d}Z/\mathrm{d}t &= \lambda(t)S \\
\mathrm{d}B/\mathrm{d}t &= \nu(t)
\end{aligned}
$$

and the *augmented* system is numerically integrated. Observed incidence is simulated from incidence by scaling the latter by prob and convolving the result with delay.

seir(stochastic = TRUE) works by simulating a Markov process corresponding to the augmented system, as described in the reference. Observed incidence is simulated from incidence by binning binomial samples taken with probabilities prob over future observation intervals according to multinomial samples taken with probabilities delay.

Value

A "multiple time series" object, inheriting from class `mts`. Beneath the class, it is a `length.out`-by-`(1+m+n+1+2)` numeric matrix with columns `S`, `E`, `I`, `R`, `Z`, and `B`, where `Z` and `B` specify incidence and births as the number of infections and births since the previous time point.

If `prob` or `delay` is not missing, then there is an additional column `Z.obs` specifying *observed* incidence as the number of infections observed since the previous time point. The first `length(delay)` elements of this column contain partial counts.

References

Cao, Y., Gillespie, D. T., & Petzold, L. R. (2007). Adaptive explicit-implicit tau-leaping method with automatic tau selection. *Journal of Chemical Physics*, *126*(22), Article 224101, 1-9. doi:10.1063/1.2745299

See Also

`seir.auxiliary`, `seir.canonical`, `seir.library`.

Examples

```
if (requireNamespace("adaptivetau")) withAutoprint({

beta <- function (t, a = 1e-01, b = 1e-05) b * (1 + a * sinpi(t / 26))
nu   <- function (t) 1e+03
mu   <- function (t) 1e-03

sigma <- 0.5
gamma <- 0.5
delta <- 0

init <- c(S = 50200, E = 1895, I = 1892, R = 946011)

length.out <- 250L
prob <- 0.1
delay <- diff(pgamma(0:8, 2.5))

set.seed(0L)
X <- seir(length.out, beta, nu, mu, sigma, gamma, delta, init = init,
          prob = prob, delay = delay, epsilon = 0.002)
##                                                  ^^^^^
## default epsilon = 0.05 allows too big leaps => spurious noise
##
str(X)
plot(X)

r <- 10L
Y <- do.call(cbind, replicate(r, simplify = FALSE,
seir(length.out, beta, nu, mu, sigma, gamma, delta, init = init,
     prob = prob, delay = delay, epsilon = 0.002)[, "Z.obs"]))
str(Y) # FIXME: stats:::cbind.ts mangles dimnames
plot(window(Y, start = tsp(Y)[1L] + length(delay) / tsp(Y)[3L]),
```

```
##        ^^^^^
## discards points showing edge effects due to 'delay'
##
plot.type = "single", col = seq_len(r), ylab = "Case reports")

})
```

`seir.auxiliary` *Auxiliary Functions for the SEIR Model without Forcing*

Description

Calculate the basic reproduction number, endemic equilibrium, and Jacobian matrix of the SEIR model without forcing.

Usage

```
seir.R0      (beta, nu = 0, mu = 0, sigma = 1, gamma = 1, delta = 0,
              m = 1L, n = 1L, N = 1)
seir.ee      (beta, nu = 0, mu = 0, sigma = 1, gamma = 1, delta = 0,
              m = 1L, n = 1L, N = 1)
seir.jacobian(beta, nu = 0, mu = 0, sigma = 1, gamma = 1, delta = 0,
              m = 1L, n = 1L)
```

Arguments

`beta`, `nu`, `mu`, `sigma`, `gamma`, `delta`	non-negative numbers. `beta`, `nu`, and `mu` are the rates of transmission, birth, and natural death. `m*sigma`, `n*gamma`, and `delta` are the rates of removal from each latent, infectious, and recovered compartment.
`m`	a non-negative integer indicating a number of latent stages.
`n`	a positive integer indicating a number of infectious stages.
`N`	a non-negative number indicating a population size for the (`nu == 0 && mu == 0`) case.

Details

If $\mu, \nu = 0$, then the basic reproduction number is computed as

$$\mathcal{R}_0 = N\beta/\gamma$$

and the endemic equilibrium is computed as

$$\begin{bmatrix} S \\ E^i \\ I^j \\ R \end{bmatrix} = \begin{bmatrix} \gamma/\beta \\ w\delta/(m\sigma) \\ w\delta/(n\gamma) \\ w \end{bmatrix}$$

where w is chosen so that the sum is N.

If $\mu, \nu > 0$, then the basic reproduction number is computed as

$$\mathcal{R}_0 = \nu\beta a^{-m}(1 - b^{-n})/\mu^2$$

and the endemic equilibrium is computed as

$$\begin{bmatrix} S \\ E^i \\ I^j \\ R \end{bmatrix} = \begin{bmatrix} \mu a^m/(\beta(1 - b^{-n})) \\ w a^{m-i} b^n (\delta + \mu)/(m\sigma) \\ w b^{n-j} (\delta + \mu)/(n\gamma) \\ w \end{bmatrix}$$

where w is chosen so that the sum is ν/μ, the population size at equilibrium, and $a = 1 + \mu/(m\sigma)$ and $b = 1 + \mu/(n\gamma)$.

Currently, none of the functions documented here are vectorized. Arguments must have length 1.

Value

`seir.R0` returns a numeric vector of length 1. `seir.ee` returns a numeric vector of length `1+m+n+1`. `seir.jacobian` returns a function of one argument `x` (which must be a numeric vector of length `1+m+n+1`) whose return value is a square numeric matrix of size `length(x)`.

See Also

`seir`, for the system of ordinary differential equations on which these computations are predicated.

`seir.canonical` *Solve the Canonical SEIR Equations*

Description

Numerically integrates the canonical SEIR equations, a special case of the more general SEIR equations handled by `seir`; see 'Details'.

Usage

```
seir.canonical(from = 0, to = from + 1, by = 1,
               R0, ell = (2 * n)/(3 * n + 1),
               m = 1L, n = 1L,
               init = c(1 - p, p), p = .Machine[["double.neg.eps"]],
               weights = rep(c(1, 0), c(1L, m + n - 1L)),
               root = c("none", "peak"), aggregate = FALSE, ...)

## S3 method for class 'seir.canonical'
summary(object, tol = 1e-6, ...)
```

Arguments

from, to, by	passed to seq.int in order to generate an increasing, equally spaced vector of time points in units of the generation interval.
R0	a positive number giving the basic reproduction number.
ell	a number in $(0, 1)$ giving the ratio of the mean latent period and mean generation interval when m is positive. The default value implies that the mean latent period and mean infectious period are equal.
m	a non-negative integer indicating a number of latent stages.
n	a positive integer indicating a number of infectious stages.
init	a numeric vector of length 2 giving initial susceptible and infected proportions.
p	a number in $(0, 1]$ used only to define init when init is unset.
weights	a numeric vector of length m+n containing non-negative weights, defining the initial distribution of infected individuals among the latent and infectious stages. By default, all infected individuals occupy the first stage.
root	a character string determining what is returned. "none": the numerical solution as a multiple time series object. "peak": information about the time and state when Y attains a local maximum.
aggregate	a logical indicating if latent and infectious compartments should be aggregated when root = "none".
...	optional arguments passed to lsoda.
object	an R object inheriting from class seir.canonical, typically the value of a call to seir.canonical.
tol	a positive number giving an upper bound on the relative change (from one time point to the next) in the slope of log prevalence, defining time windows in which growth or decay of prevalence is considered to be exponential.

Details

The general SEIR equations handled by seir are

$$
\begin{aligned}
\mathrm{d}S \quad /\mathrm{d}t &= \nu(t) + \delta R - (\beta(t) \textstyle\sum_j I^j + \mu(t))S \\
\mathrm{d}E^1 \quad /\mathrm{d}t &= \beta(t) S \textstyle\sum_j I^j - (m\sigma + \mu(t))E^1 \\
\mathrm{d}E^{i+1} /\mathrm{d}t &= m\sigma E^i - (m\sigma + \mu(t))E^{i+1} \\
\mathrm{d}I^1 \quad /\mathrm{d}t &= m\sigma E^m - (n\gamma + \mu(t))I^1 \\
\mathrm{d}I^{j+1} /\mathrm{d}t &= n\gamma I^j - (n\gamma + \mu(t))I^{j+1} \\
\mathrm{d}R \quad /\mathrm{d}t &= n\gamma I^n - (\delta + \mu(t))R
\end{aligned}
$$

The canonical SEIR equations are obtained by substituting $\beta(t) = \mathcal{R}_0\gamma$, $\nu(t) = \mu(t) = \delta = 0$, $1/\sigma = \ell g$, and $1/\gamma = (1 - \ell)g/h$, where $h = (n + 1)/(2n)$, then choosing $\tau = t/g$ as an

independent variable:

$$\begin{aligned}
\mathrm{d}S &/\mathrm{d}\tau = -(h\mathcal{R}_0/(1-\ell))S\textstyle\sum_j I^j \\
\mathrm{d}E^1 &/\mathrm{d}\tau = (h\mathcal{R}_0/(1-\ell))S\textstyle\sum_j I^j - (m/\ell)E^1 \\
\mathrm{d}E^{i+1} &/\mathrm{d}\tau = (m/\ell)E^i - (m/\ell)E^{i+1} \\
\mathrm{d}I^1 &/\mathrm{d}\tau = (m/\ell)E^m - (hn/(1-\ell))I^1 \\
\mathrm{d}I^{j+1} &/\mathrm{d}\tau = (hn/(1-\ell))I^j - (hn/(1-\ell))I^{j+1} \\
\mathrm{d}R &/\mathrm{d}\tau = (hn/(1-\ell))I^n
\end{aligned}$$

The constraint $\mu(t) = \nu(t)$ implies that the population size $N = S + \sum_i E^i + \sum_j I^j + R$ is constant, hence, without loss of generality, `seir.canonical` assumes $N = 1$ and drops the last equation. The resulting system of $1 + m + n$ equations is augmented with a final equation

$$\mathrm{d}Y/\mathrm{d}\tau = (h/(1-\ell))(\mathcal{R}_0 S - 1)\sum_j I^j$$

due to the usefulness of the solution Y in applications.

Value

If `root = "none"`, a "multiple time series" object, inheriting from class `seir.canonical` and transitively from class `mts`. Beneath the class, it is a `length(seq(from, to, by))`-by-`(1+m+n+1)` numeric matrix with columns `S`, `E`, `I`, and `Y`.

If `root = "peak"`, a numeric vector of length 5 of the form `c(tau, S, E, I, Y)` containing the time in units of the generation interval at which Y attains a local maximum and the values at that time of S, $\sum_i E^i$, $\sum_j I^j$, and Y. Attributes `E.full`, `I.full`, and `curvature` store the corresponding values of E^i, I^j, and Y''. If Y does not attain a local maximum between times `from` and `to`, then all of the elements are `NaN` and there are no attributes. If `m` is zero, then the third element is `NaN` and there is no attribute `E.full`.

The method for `summary` returns a numeric vector of length 2 containing the left and right "tail exponents", defined as the asymptotic values of the slope of log prevalence. `NaN` elements indicate that a tail exponent cannot be approximated from the prevalence time series represented by `object`, because the time window does not cover enough of the tail, where the meaning of "enough" is set by `tol`.

See Also

`seir`.

Examples

```
if (requireNamespace("deSolve")) withAutoprint({

to <- 100; by <- 0.01; R0 <- 16; ell <- 1/3

peak <- seir.canonical(to = to, by = by, R0 = R0, ell = ell,
                       root = "peak")
peak
```

```
to <- 4 * peak[["tau"]] # a more principled endpoint

soln <- seir.canonical(to = to, by = by, R0 = R0, ell = ell)
head(soln)

plot(soln) # dispatching stats:::plot.ts

plot(soln[, "Y"])
abline(v = peak[["tau"]], h = peak[["Y"]],
       lty = 2, lwd = 2, col = "red")

xoff <- function (x, k) x - x[k]
prev <- soln[, "E"] + soln[, "I"]
lamb <- summary(soln)
k <- c(16L, nrow(soln)) # c(1L, nrow(soln)) suffers due to transient
plot(prev, log = "y")
for (i in 1:2)
lines(prev[k[i]] * exp(lamb[i] * xoff(time(prev), k[i])),
      lty = 2, lwd = 2, col = "red")

})
```

`seir.library` *Often Used Simulations*

Description

Infectious disease time series simulated using `seir`, for use primarily in examples, tests, and vignettes. Users should not rely on simulation details, which may change between package versions.

Note that simulation code depends on availability of suggested packages **adaptivetau** and **deSolve**. If the dependency cannot be loaded then the value of the data set is `NULL`.

Usage

```
## if (requireNamespace("deSolve"))
data(seir.ts01, package = "fastbeta")
## else ...

## if (requireNamespace("adaptivetau"))
data(seir.ts02, package = "fastbeta")
## else ...
```

Format

A "multiple time series" object, inheriting from class `mts`, always a subset of the result of a call to `seir`, discarding transient behaviour. Simulation parameters may be preserved as attributes.

Source

Scripts sourced by `data` to reproduce the simulations are located in subdirectory 'data' of the **fastbeta** installation; see, e.g. `system.file("data", "seir.ts01.R", package = "fastbeta")`.

See Also

`seir`.

Examples

```
if (requireNamespace("deSolve")) withAutoprint({

data(seir.ts01, package = "fastbeta")
str(seir.ts01)
plot(seir.ts01)

})

if (requireNamespace("adaptivetau")) withAutoprint({

data(seir.ts02, package = "fastbeta")
str(seir.ts02)
plot(seir.ts02)

})
```

---

smallpox *Smallpox Mortality in London, England, 1661-1930*

Description

Time series of deaths due to smallpox, deaths due to all causes, and births in London, England, from 1661 to 1930, as recorded in the London Bills of Mortality and the Registrar General's Weekly Returns.

Usage

```
data(smallpox, package = "fastbeta")
```

Format

A data frame with 13923 observations of 5 variables:

from start date of the record.

nday length of the record, which is the number of days (typically 7) over which deaths and births were counted.

smallpox count of deaths due to smallpox.

allcauses count of deaths due to all causes.

births count of births.

Source

A precise description of the data set and its correspondence to the original source documents is provided in the reference.

A script generating the `smallpox` data frame from a CSV file accompanying the reference is available as `system.file("scripts", "smallpox.R", package = "fastbeta")`.

References

Krylova, O. & Earn, D. J. D. (2020). Patterns of smallpox mortality in London, England, over three centuries. *PLOS Biology*, *18*(12), Article e3000506, 1-27. doi:10.1371/journal.pbio.3000506

Examples

```
data(smallpox, package = "fastbeta")
str(smallpox)
table(smallpox[["nday"]]) # not all 7 days, hence:
plot(7 * smallpox / as.double(nday) ~ from, smallpox, type = "l")
```

Index

bug.report, *2*, *13*

data, *20*
deconvolve, 2, *4*, *7*, *10*

fastbeta, *2*, 4, *6*, *8*, *10*
fastbeta-package, 2
fastbeta.bootstrap, 6
fastbeta.matrix, 8

help, *2*

lsoda, *13*, *17*

mts, *4, 5*, *7, 8*, *10, 11*, *14*, *18, 19*

ptpi, 9

seir, *6, 7*, 12, *16–20*
seir.auxiliary, *14*, 15
seir.canonical, *14*, 16
seir.ee *(*seir.auxiliary*)*, 15
seir.jacobian *(*seir.auxiliary*)*, 15
seir.library, *14*, 19
seir.R0 *(*seir.auxiliary*)*, 15
seir.ts01 *(*seir.library*)*, 19
seir.ts02 *(*seir.library*)*, 19
seq.int, *17*
sir *(*seir*)*, 12
smallpox, 20
ssa.adaptivetau, *13*
summary, *18*
summary.seir.canonical
        *(*seir.canonical*)*, 16
system.file, *20, 21*

tsp, *5*, *7*