

Package: ggoutbreak (via r-universe)

October 7, 2026

Title Estimate Incidence, Proportions and Exponential Growth Rates

Version 0.6.1

Description Simple statistical models and visualisations for calculating the incidence, proportion, exponential growth rate, and reproduction number of infectious disease case time series. This toolkit was largely developed during the COVID-19 pandemic.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE, roclets = c("`collate", "`rd", "`namespace"), packages = "`pkgtools")

RoxygenNote 7.3.3.9007

Suggests downloader, fs, knitr, rappdirs, rmarkdown, here, jsonlite, readr, stringr, testthat (>= 3.0.0), devtools, patchwork, coarseDataTools, splines, lmttest, mirai, carrier, magrittr, ragg, lifecycle, outbreaks, incidence, incidence2, grates

VignetteBuilder knitr

Imports dplyr, nnet, rlang, stats, tidyr, locfit, lubridate, scales, ggplot2, purrr, EpiEstim, withr, MASS, interfacer (>= 0.4.0), cli, PDQutils, signal, mgcv, deSolve, ukc19 (>= 0.0.3), vctrs, forcats

Remotes github::ai4ci/interfacer, github::ai4ci/ukc19

URL <https://ai4ci.github.io/ggoutbreak/>,
<https://github.com/ai4ci/ggoutbreak>,
<https://doi.org/10.5281/zenodo.13165561>

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

LazyDataCompression gzip

Language en-GB

Config/Needs/build terminological/pkgtools
Config/pak/sysreqs cmake make libicu-dev libuv1-dev libxml2-dev
Repository <https://ai4ci.r-universe.dev>
Date/Publication 2025-12-11 15:55:28 UTC
RemoteUrl <https://github.com/ai4ci/ggoutbreak>
RemoteRef 0.6.1
RemoteSha a6a62329917a254179fb8856db147fa86dd3f7a3

Contents

as.time_period	5
breaks_log1p	8
cfg_beta_prob_rng	9
cfg_gamma_ip_fn	10
cfg_ip_sampler_rng	10
cfg_linear_fn	11
cfg_step_fn	12
cfg_transition_fn	12
cfg_weekly_gamma_rng	13
cfg_weekly_ip_fn	14
cfg_weekly_proportion_rng	14
cut_date	15
date_seq	16
date_seq.Date	17
date_seq.numeric	18
date_seq.time_period	18
dbeta2	19
dcgamma	20
dgamma2	21
dlnorm2	22
dlogitnorm	23
dlogitnorm2	24
dnbinom2	25
dnull	26
doubling_time	27
dwedge	28
fdmy	29
format_ip	30
gam_delayed_reporting	30
gam_knots	31
gam_nb_model_fn	32
gam_poisson_model_fn	33
geom_events	34
growth_rate_from_incidence	37
growth_rate_from_prevalence	38
growth_rate_from_proportion	40

infer_population	41
infer_prevalence	42
infer_rate_ratio	44
infer_risk_ratio	45
integer_breaks	47
inv_wallinga_lipsitch	47
is.Date	48
julian.time_period	49
labels.time_period	50
linelist	51
logit_trans	52
make_empirical_ip	53
make_fixed_ip	54
make_gamma_ip	54
make_posterior_ip	56
make_resampled_ip	58
max_date	59
min_date	59
months.time_period	60
multinomial_nnet_model	61
normalise_count	63
normalise_incidence	64
omega_matrix	66
pbeta2	66
pcgamma	67
pgamma2	68
plnorm2	69
plogitnorm	70
plogitnorm2	71
plot_cases	72
plot_counts	73
plot_growth_phase	75
plot_growth_rate	79
plot_incidence	81
plot_ip	84
plot_multinomial	84
plot_prevalence	86
plot_proportion	87
plot_proportions_data	89
plot_rt	90
pnbinom2	92
pnull	94
poisson_gam_model	94
poisson_glm_model	97
poisson_locfit_model	99
proportion_glm_model	101
proportion_locfit_model	103
pwedge	105

qbeta2	106
qcgamma	107
qgamma2	108
qlnorm2	109
qlogitnorm	110
qlogitnorm2	111
qnbinom2	111
qnull	113
quantify_lag	114
quarters.time_period	115
qwedge	116
rbern	117
rbeta2	117
rcategorical	118
rcgamma	119
rdiscgamma	120
reband_discrete	120
rescale_model	122
rexpgrowth	123
rexpgrowthI0	124
rgamma2	125
rlnorm2	126
rlogitnorm	127
rlogitnorm2	128
rnbinom2	128
rnull	130
rt_cori	130
rt_epiestim	132
rt_from_growth_rate	134
rt_from_incidence	136
rt_from_renewal	138
rt_incidence_reference_implementation	140
rt_incidence_timeseries_implementation	143
rwedge	146
scale_x_log1p	146
scale_x_logit	147
scale_y_log1p	147
scale_y_logit	148
score_estimate	148
set_defaults	150
sim_apply_ascertainment	152
sim_apply_delay	153
sim_branching_process	155
sim_convolution	157
sim_delay	159
sim_delayed_observation	160
sim_events	161
sim_geom_function	162

sim_multinomial	164
sim_poisson_model	165
sim_poisson_Rt_model	166
sim_seir_model	168
sim_summarise_linelist	169
sim_test_data	171
summarise_ip	172
time_aggregate	172
time_summarise	173
timeseries	175
vcov_from_residuals	178
wallinga_lipsitch	179
wedge	180
weekdays.time_period	181

Index	183
--------------	------------

as.time_period	<i>Time period S3 class methods</i>
----------------	-------------------------------------

Description

Time periods are just a zero based numeric representation of dates with a time unit baked in. This allows variable length periods (e.g. days or weeks), and fractional days to be represented in a consistent(ish) way between things that want to deal in dates (like ggplot) and things that want to deal in numbers (like model fitting)

Usage

```
as.time_period(x, ...)

## S3 method for class 'time_period'
as.time_period(x, unit = NULL, start_date = NULL, ...)

## S3 method for class 'Date'
as.time_period(x, unit = NULL, anchor = NULL, ...)

## S3 method for class 'numeric'
as.time_period(x, unit = NULL, start_date = NULL, ...)

## S3 method for class 'grates_epiweek'
as.time_period(x, ...)

## S3 method for class 'grates_isoweek'
as.time_period(x, ...)

## S3 method for class 'grates_period'
as.time_period(x, ...)
```

```
## S3 method for class 'time_period'
seq(from, to = from, ...)

is.time_period(x)

date_to_time(dates, unit = NULL, start_date = NULL)

time_to_date(
  timepoints,
  unit = attr(timepoints, "unit"),
  start_date = attr(timepoints, "start_date")
)
```

Arguments

x	a vector of dates, numbers (may be integer or real) or a <code>time_period</code> to convert to a <code>time_period</code>
...	arguments passed to or from methods.
unit	the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another <code>time_period</code> . If x is a <code>time_period</code> , and the unit is different to that of x this will return a rescaled <code>time_period</code> using the new units.
start_date	the zero time date as something that can be coerced to a date. If the x input is already a <code>time_period</code> and this is different to its <code>start_date</code> then x will be recalibrated to use the new start date.
anchor	only relevant if x is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in x it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see set_defaults())
from, to	the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument.
dates	a vector of dates to convert to a <code>time_period</code>
timepoints	a <code>time_period</code> vector to convert to a set of dates.

Value

a vector of class `time_period`

Functions

- `seq(time_period)`: Create a sequence using `time_periods`
- `is.time_period()`: Check is a `time_period`
- `date_to_time()`: Convert a set of dates to numeric timepoints
- `time_to_date()`: Convert a set of time points to dates

Unit tests

```

tmp = as.time_period(grates::as_epiweek("2019-W12", format = "yearweek")+0:2)
testthat::expect_equal(
  lubridate::epiweek(as.Date(tmp)),
  c(12, 13, 14)
)

tmp = as.time_period(grates::as_isoweek("2019-W12", format = "yearweek")+0:2)
testthat::expect_equal(
  lubridate::isoweek(as.Date(tmp)),
  c(12, 13, 14)
)

x = as.time_period(as.Date("2025-01-01")+0:2, anchor="start", unit="1 day")
y = as.time_period(as.Date("2025-01-07")+0:2, anchor="start", unit="1 day")
testthat::expect_equal(as.numeric(c(x, y)), c(0, 1, 2, 6, 7, 8))
testthat::expect_equal(as.numeric(c(y, x)), c(0, 1, 2, -6, -5, -4))

z = as.time_period(as.Date("2025-01-01")+0:2*7, anchor="start", unit="1 week")
testthat::expect_equal(as.numeric(c(x, z)), c(0, 1, 2, 0, 7, 14))
testthat::expect_equal(
  as.numeric(c(z, x)),
  c(0, 1, 2, 0, 0.142857142857143, 0.285714285714286)
)

testthat::expect_equal(
  as.Date(c(y, z)),
  structure(c(20095, 20096, 20097, 20089, 20096, 20103), class = "Date")
)

seq(x[[1]], y[[1]])

```

Examples

```

#' # 100 weeks from 2020-01-01

tmp = as.time_period(0:100, 7, "2020-01-01")
as.Date(tmp)

range(tmp)
min(tmp)
tmp2 = as.integer(as.Date(tmp))
# testthat::expect_true(all(na.omit(tmp2-lag(tmp2)) == 7))

tmp2 = as.time_period(0:23, 1/24, "2020-01-01")
as.POSIXct(tmp2)

```

```

# convert timeseries to new "unit"
tmp = as.time_period(0:100, 7, "2020-01-01")
tmp2 = as.time_period(tmp,1)
testthat::expect_equal(as.numeric(tmp2), 0:100*7)

# 100 weeks from 2020-01-01

tmp = as.time_period(0:100, 7, "2020-01-01")
as.Date(tmp)

range(tmp)
min(tmp)
tmp2 = as.integer(as.Date(tmp))
# testthat::expect_true(all(na.omit(tmp2-lag(tmp2)) == 7))

tmp2 = as.time_period(0:23, 1/24, "2020-01-01")
as.POSIXct(tmp2)

# convert timeseries to new "unit"
tmp = as.time_period(0:100, 7, "2020-01-01")
tmp2 = as.time_period(tmp,1)
testthat::expect_equal(as.numeric(tmp2), 0:100*7)

# Time to date
times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
dates = time_to_date(times)

# Date to time
times = date_to_time(as.Date("2019-12-29")+0:100, "1 week")
dates = time_to_date(times)

```

breaks_log1p

A scales breaks generator for log1p scales

Description

A scales breaks generator for log1p scales

Usage

```
breaks_log1p(n = 5, base = 10)
```

Arguments

n	the number of breaks
base	the base for the breaks

Value

a function for ggplot scale breaks

Examples

```
ggplot2::ggplot(ggplot2::diamonds, ggplot2::aes(x=price))+
  ggplot2::geom_density()+
  ggplot2::scale_x_continuous(trans="log1p", breaks=breaks_log1p())
```

cfg_beta_prob_rng	<i>Generate a random probability based on features of the simulation</i>
-------------------	--

Description

Generate a random probability based on features of the simulation

Usage

```
cfg_beta_prob_rng(probability_fn = ~0.8, kappa_fn = ~0.1)
```

Arguments

probability_fn a function which gives the time-varying mean of a beta distribution, The function will be called minimally with `.x` or `'t'` which will be the time as a time period. Other variables may be present.

kappa_fn a function which gives the time-varying dispersion of a beta distribution. The function will be called with minimally `.x` or `t` which will be the time period and `.y` or `mean` will be the mean. Other variables may be present.

Value

a time dependent function that inputs a time (as a `time_period`) and returns an probability for each day defined by the `probability_fn` and `kappa_fn`

See Also

[rbeta2\(\)](#)

Examples

```
fn = cfg_beta_prob_rng(~ ifelse(.x<=5,0.1,0.9))
fn(1:10)
```

cfg_gamma_ip_fn	<i>Get a IP generating function from time varying mean and SD of a gamma function</i>
-----------------	---

Description

Get a IP generating function from time varying mean and SD of a gamma function

Usage

```
cfg_gamma_ip_fn(mean_fn = ~2, sd_fn = function(mean) sqrt(mean))
```

Arguments

mean_fn	a function which gives the time-varying mean of a gamma distribution, The function will be called minimally with .x or t which will be the time as a time period. Other variables may be present.
sd_fn	a function which gives the time-varying mean of a gamma distribution. The function will be called with minimally .x or t which will be the time period and .y or mean will be the mean. Other variables may be present.

Value

a time dependent function that inputs a time (as a time_period) and returns an ip delay distribution for each day defined by the mean_fn and sd_fn

Examples

```
fn = cfg_gamma_ip_fn(mean_fn = function(t) ifelse(t < 5, 4, 2))
# a gamma function that changes mean at time 5
fn(4)
fn(7)
```

cfg_ip_sampler_rng	<i>Randomly sample from an empirical distribution</i>
--------------------	---

Description

This is used for random sampling from the infectivity profile for times to infection, for example. There is nothing to stop you putting in a delay distribution with negative times but strange things may happen in your simulation.

Usage

```
cfg_ip_sampler_rng(ip = i_empirical_ip)
```

Arguments

ip a long format empirical distribution - a dataframe with columns:

- boot (anything + default(1)) - a bootstrap identifier
- probability (proportion) - the probability of new event during this period.
- a0 (double) - the beginning of the time period (in days)
- a1 (double) - the end of the time period (in days)

Minimally grouped by: boot (and other groupings allowed).

Value

a function which accepts n parameter which produces random samples from the ip distribution

Examples

```
tmp = cfg_ip_sampler_rng(example_ganyani_ip())(10000)

# This discretised ganyani distribution is based on these figures:
# mean: 5.2 (3.78-6.78) and sd: 1.72 (0.91-3.93)
format_ip(example_ganyani_ip())

mean(tmp) # Should be about 5.2
stats::sd(tmp) # Should be about 1.72
```

cfg_linear_fn	<i>Linear function from dataframe</i>
---------------	---------------------------------------

Description

Linear function from dataframe

Usage

```
cfg_linear_fn(changes, ..., col_name = setdiff(colnames(changes), "t"))
```

Arguments

changes a dataframe with t and <col_name> columns which define the change points for a piecewise linear function.

... not used

col_name the value column (optional if only 2 columns)

Value

a function that inputs a vector t and returns a linearly interpolated value from <col_name>

cfg_step_fn	<i>Step function from dataframe</i>
-------------	-------------------------------------

Description

Step function from dataframe

Usage

```
cfg_step_fn(changes, ..., col_name = setdiff(colnames(changes), "t"))
```

Arguments

changes	a dataframe with t and <col_name> columns which define the cut points for a step function.
...	not used
col_name	the value column (optional if only 2 columns)

Value

a function that inputs a vector t and returns the next smallest corresponding value in <col_name> (or the first one)

cfg_transition_fn	<i>Sample from a multinomial transition matrix</i>
-------------------	--

Description

This is particularly designed for use within the fn_list_next_gen parameter of [sim_branching_process\(\)](#) to allow age group contract matrices to be applied for example (assuming a categorical age).

Usage

```
cfg_transition_fn(transition)
```

Arguments

transition	a transition matrix or long format dataframe. If a matrix the columns should add to 1, and the column names are the input class. row names are the output class. the data frame format must have input, output, and probability columns.
------------	--

Value

a function that given an input will return samples of the output class according to the probability distributions.

Examples

```
age = rep(c("child", "adult", "elderly"), 100)

fn = cfg_transition_fn(dplyr::tribble(
  ~input, ~output, ~probability,
  "child", "child", 0.5,
  "child", "adult", 0.5,
  "adult", "child", 0.5,
  "adult", "adult", 0.3,
  "adult", "elderly", 0.2,
  "elderly", "elderly", 0.5,
  "elderly", "adult", 0.5,
))

table(fn(age), age)
```

cfg_weekly_gamma_rng	<i>Weekly delay function with day of week effect</i>
----------------------	--

Description

This function returns a random number generator from a gamma distribution that has a weekly period and configurable degree of additional variability around weekly pattern. It is suited to delays of things like testing which may depend on the day of week.

Usage

```
cfg_weekly_gamma_rng(
  mean = c(1, 1, 1, 1, 4, 3, 2),
  sd = sqrt(mean),
  week_starts = weekdays(as.Date("2024-10-14"))
)
```

Arguments

mean	a mean amount of delay for each day of the week
sd	the SD of the delay
week_starts	locale description of first day of week (default is a "Monday").

Value

a random number generator taking t time parameter and returning a duration the that time t depending on the day of the week.

Examples

```
fn = cfg_weekly_gamma_rng(c(1,1,1,1,4,3,2))
matrix(fn(1:42), ncol=7, byrow=TRUE)
```

cfg_weekly_ip_fn	<i>Weekly convolution distribution function</i>
------------------	---

Description

Weekly convolution distribution function

Usage

```
cfg_weekly_ip_fn(
  mean = c(1, 1, 1, 1, 4, 3, 2),
  sd = sqrt(mean),
  week_starts = weekdays(as.Date("2024-10-14"))
)
```

Arguments

mean	The means of a gamma distributed delay function for each weekday
sd	The sds of a gamma distributed delay function for each weekday
week_starts	locale description of first day of week (default is a "Monday").

Value

a time dependent function that inputs a time (as a time_period) and generates an IP delay distribution for each day varying by day of week

Examples

```
cat(sapply(cfg_weekly_ip_fn()(1:7), format_ip), sep = "\n")
```

cfg_weekly_proportion_rng	<i>Random probability function with day of week effect</i>
---------------------------	--

Description

This function returns a random probability generator that has a weekly period and configurable degree of additional variability around weekly pattern. It is suited to probabilities of things like testing which may depend on the day of week.

Usage

```
cfg_weekly_proportion_rng(
  prob = c(0.8, 0.8, 0.8, 0.8, 0.8, 0.5, 0.5),
  kappa = 0.1,
  week_starts = weekdays(as.Date("2024-10-14"))
)
```

Arguments

prob	the rates of e.g. ascertainment for each day of the week.
kappa	dispersion parameter between 0 and 1. 0 is no dispersion. 1 is maximum
week_starts	locale description of first day of week (default is a "Monday").

Value

a random number generator function taking `t` time parameter and returning a probability of ascertainment for that time, depending on day of week etc.

Examples

```
fn = cfg_weekly_proportion_rng(c(0.9,0.9,0.9,0.9,0.9,0.1,0.1))
matrix(fn(1:42),ncol=7,byrow=TRUE)
```

cut_date	<i>Places a set of dates within a regular time series</i>
----------	---

Description

The counterpart to `date_seq_dates()`. Take an original set of data and place it within a regular time series where the periodicity of the time series may be expressed as numbers of days, weeks, months, quarters, or years, and the periods are defined by an anchoring date, day of the week or by reference to the start or end of the input dates. This can either return the periods as dates or factors (e.g. for plotting) or as a `time_period` for analysis that relies on a numeric representation of the date or duration from the anchor.

Usage

```
cut_date(
  dates,
  unit,
  anchor = "start",
  output = c("date", "factor", "time_period"),
  dfmt = "%d/%b/%y",
  ifmt = "{start} - {end}",
  ...
)
```

Arguments

dates	a set of dates
unit	a period e.g. "1 week"
anchor	one of a date, "start" or "end" or a weekday name e.g. "mon" this will always be one of the start of the time periods we are cutting into

output	return the result as either a "date" (the default), an ordered "factor" with the date ranges as a label, or as a "time_period". The result is named with labels referring to the
dfmt	the strptime format for the dates in the labels
ifmt	a sprintf format for the period label containing %s exactly twice.
...	ignored

Value

a set of dates, times or a factor level, representing the start of the period the date falls into, where the period is defined by the duration and the anchor

Examples

```

dates = as.Date(c("2020-01-01", "2020-02-01", "2020-01-15", "2020-02-03", NA))
fs = date_seq(dates, "2 days")
dates - cut_date(dates, "2 days")
cut_date(dates, unit="2 days", output="time_period")

# A weekly set of dates:
dates2 = Sys.Date() + floor(stats::runif(50, max=10)) * 7

# in this specific situation the final date is not truncated because the
# input data is seen as an exact match for the whole output period.
cut_date(dates2, "1 week", "sun", output="factor")
cut_date(dates2, dfmt = "%d/%b", output="factor", unit = "2 weeks", anchor="sun")

```

date_seq	<i>Create the full sequence of values in a vector</i>
----------	---

Description

This is useful if you want to fill in missing values that should have been observed but weren't. For example, `date_seq(c(1, 2, 4, 6), 1)` will return 1:6.

Usage

```
date_seq(x, period, ...)
```

Arguments

x	a numeric or date vector
period	Gap between each observation. The existing data will be checked to ensure that it is actually of this periodicity.
...	for subtype methods

Value

a vector of the same type as the input

Examples

```
date_seq(c(1, 2, 4, 5, 10), 1)
```

date_seq.Date

Expand a date vector to the full range of possible dates

Description

Derive from a vector of observation dates, a complete ordered sequence of periods in a regular time series, where the length of the periods is specified, as a number of days, weeks, years etcetera. E.g. this can convert a random set of dates to a ordered complete list of 1 week intervals (or 2 month intervals) spanning the same range as the dates. This has some interesting problems regarding where to put breaks within a month or week. Often this is either based on a specific date (e.g. yearly periods starting at 2020-01-01) or a day of week (e.g. 2 weekly periods starting on a Sunday) or maybe relative to the input time series (weekly ending on the last date of the data). There is also a problem when we consider data that may have incomplete starting and end periods, which may not be comparable to other periods, and we may need to exclude these from the result.

Usage

```
## S3 method for class 'Date'
date_seq(x, period = .day_interval(x), anchor = "start", complete = FALSE, ...)
```

Arguments

x	a vector of dates, possibly including NA values
period	the gap between observations as a number of days or as a natural language definition of the period such as "1 week", "2 weeks", "1 month", etcetera. If not given this will be derived from the dates.
anchor	defines a day that appears in the sequence (if it were to extend that far). Given either as a date, or "start", "end" or a day of the week, e.g. "mon".
complete	truncate incomplete start and end periods
...	ignored

Value

a vector of dates for regular periods between the minimum and maximum of dates, with the boundaries defined by the anchor.

Examples

```
date_seq(as.Date(c("2020-01-01", "2020-02-01", "2020-01-15", "2020-02-01", NA)), "2 days")
```

date_seq.numeric	Create the full sequence of values in a vector
------------------	--

Description

This is useful if you want to fill in missing values that should have been observed but weren't. For example, `date_seq(c(1, 2, 4, 6), 1)` will return 1:6.

Usage

```
## S3 method for class 'numeric'
date_seq(x, period = 1, tol = 1e-06, ...)
```

Arguments

<code>x</code>	a numeric or date vector
<code>period</code>	Gap between each observation. The existing data will be checked to ensure that it is actually of this periodicity.
<code>tol</code>	Numerical tolerance for checking periodicity.
<code>...</code>	for subtype methods

Value

a vector of the same type as the input

Examples

```
date_seq(c(1, 2, 4, 5, 10), 1)
```

date_seq.time_period	Expand a time_period vector to the full range of possible times
----------------------	---

Description

Derive from a vector of observation `time_periods`, a complete ordered sequence of periods in a regular time series, where the length of the periods is specified, as a number of days, weeks, years etc. E.g. this can convert a random set of times to a ordered complete list of 1 week intervals (or 2 month intervals) spanning the same range as the dates. This has some interesting problems regarding where to put breaks within a month or week. Often this is either based on a specific date (e.g. yearly periods starting at 2020-01-01) or a day of week (e.g. 2 weekly periods starting on a sunday) or maybe relative to the input time series (weekly ending on the last date of the data). There is also a problem when we consider data that may have incomplete starting and end periods, which may not be comparable to other periods, and we may need to exclude these from the result.

Usage

```
## S3 method for class 'time_period'
date_seq(x, period = attributes(x)$unit, complete = FALSE, ...)
```

Arguments

x	a time period vector
period	the gap between observations as a number of days or as a natural language definition of the period such as "1 week", "2 weeks", "1 month", etc. If not given this will be derived from the dates.
complete	truncate incomplete start and end periods
...	ignored

Value

a vector of time_periods for regular periods between the minimum and maximum of dates, with the boundaries defined by the anchor.

Examples

```
tmp = as.time_period(c(0,10,100), 7, "2020-01-01")
date_seq(tmp, "7 days")
date_seq(tmp, "1 day")
```

dbeta2

The Beta Distribution

Description

Density, distribution function, quantile function and random generation for the Beta distribution with parameters shape1 and shape2 (and optional non-centrality parameter ncp).

Usage

```
dbeta2(x, prob, kappa, log = FALSE)
```

Arguments

x	vector of quantiles
prob	the mean probability (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
log	logical; if TRUE, probabilities p are given as log(p).

Value

dbeta gives the density, pbeta the distribution function, qbeta the quantile function, and rbeta generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rbeta, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

`stats::dbeta()`

Examples

```
dbeta2(c(0.25,0.5,0.75), 0.5, 0.25)
```

dgamma

Density: gamma distribution constrained to have mean > sd

Description

The following conversion describes the parameters mean and kappa

Usage

```
dgamma(x, mean, kappa = 1/mean, log = FALSE)
```

Arguments

x	vector of quantiles
mean	the mean value on the true scale (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
log	logical; if TRUE, probabilities p are given as log(p).

Details

$$\text{shape: } \alpha = \frac{1}{\kappa} \text{rate: } \beta = \frac{1}{\mu \times \kappa} \text{scale: } \sigma = \mu \times \kappa$$

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

`stats::dgamma()`

Examples

```
dcgamma(seq(0,4,0.25), 2, 0.5)
```

dgamma2

The Gamma Distribution

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters shape and scale.

Usage

```
dgamma2(x, mean, sd = sqrt(mean), log = FALSE)
```

Arguments

x	vector of quantiles
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
log	logical; if TRUE, probabilities p are given as log(p).

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

`stats::dgamma()`

Examples

```
dgamma2(seq(0,4,0.25), 2, 1)
```

dlnorm2

The Log Normal Distribution

Description

Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has mean equal to meanlog and standard deviation equal to sdlog.

Usage

```
dlnorm2(x, mean = 1, sd = sqrt(exp(1) - 1), log = FALSE)
```

Arguments

x	vector of quantiles
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
log	logical; if TRUE, probabilities p are given as log(p).

Details

The log normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the logarithm. The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance $Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$ and hence the coefficient of variation is $\sqrt{\exp(\sigma^2) - 1}$ which is approximately σ when that is small (e.g., $\sigma < 1/2$).

Value

dlnorm gives the density, plnorm gives the distribution function, qlnorm gives the quantile function, and rlnorm generates random deviates.

The length of the result is determined by n for rlnorm, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

The cumulative hazard $H(t) = -\log(1 - F(t))$ is `-plnorm(t, r, lower = FALSE, log = TRUE)`.

Source

`dlnorm` is calculated from the definition (in ‘Details’). `[pqr]lnorm` are based on the relationship to the normal.

Consequently, they model a single point mass at `exp(meanlog)` for the boundary case `sdlog = 0`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 14. Wiley, New York.

See Also

[Distributions](#) for other standard distributions, including [dnorm](#) for the normal distribution.

Examples

```
dlnorm2(seq(0,4,0.25), 2, 1)
```

dlogitnorm	<i>Logit-normal distribution</i>
------------	----------------------------------

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
dlogitnorm(x, meanlogit = 0, sdlogit = 1, log = FALSE)
```

Arguments

- `x` vector of quantiles ($0 < x < 1$)
- `meanlogit` the mean on the logit scale
- `sdlogit` the sd on the logit scale
- `log` logical; if TRUE, probabilities `p` are given as `log(p)`.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
dlogitnorm(seq(0.1,0.9,0.1), 0, 1)
```

dlogitnorm2

Logit-normal distribution

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
dlogitnorm2(x, prob.0.5 = 0.5, kappa = 1 - exp(-1), log = FALSE)
```

Arguments

x	vector of quantiles ($0 < x < 1$)
prob.0.5	the median on the true scale
kappa	a dispersion parameter from 0 (none) to 1 maximum dispersion
log	logical; if TRUE, probabilities p are given as log(p).

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
q = seq(0.1,0.9,0.1)
eps = sqrt(.Machine$double.eps)
dlogitnorm2(q, 0.75, 0.2)

(plogitnorm2(q+eps, 0.75, 0.2) -
 plogitnorm2(q-eps, 0.75, 0.2)) / (2*eps)
```

Description

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and prob.

Usage

```
dnbinom2(x, mean, sd = sqrt(mean), log = FALSE)
```

Arguments

x	vector of (non-negative integer) quantiles.
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
log	logical; if TRUE, probabilities p are given as log(p).

Details

The negative binomial distribution with size = n and prob = p has density

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

for $x = 0, 1, 2, \dots, n > 0$ and $0 < p \leq 1$.

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached. The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$.

A negative binomial distribution can also arise as a mixture of Poisson distributions with mean distributed as a gamma distribution (see [pgamma](#)) with scale parameter $(1 - \text{prob})/\text{prob}$ and shape parameter size. (This definition allows non-integer values of size.)

An alternative parametrization (often used in ecology) is by the *mean* μ (see above), and size, the *dispersion parameter*, where $\text{prob} = \text{size}/(\text{size} + \mu)$. The variance is $\mu + \mu^2/\text{size}$ in this parametrization.

If an element of x is not integer, the result of dnbinom is zero, with a warning.

The case size == 0 is the distribution concentrated at zero. This is the limiting distribution for size approaching zero, even if μ rather than prob is held constant. Notice though, that the mean of the limit distribution is 0, whatever the value of μ .

The quantile is defined as the smallest value x such that $F(x) \geq p$, where F is the distribution function.

Value

dnbinom gives the density, pnbinom gives the distribution function, qnbinom gives the quantile function, and rnbinom generates random deviates.

Invalid size or prob will result in return value NaN, with a warning.

The length of the result is determined by n for rnbinom, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

rnbinom returns a vector of type [integer](#) unless generated values exceed the maximum representable integer when [double](#) values are returned.

Source

dnbinom computes via binomial probabilities, using code contributed by Catherine Loader (see [dbinom](#)).

pnbinom uses [pbeta](#).

qnbinom uses the Cornish–Fisher Expansion to include a skewness correction to a normal approximation, followed by a search.

rnbinom uses the derivation as a gamma mixture of Poisson distributions, see

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag, New York. Page 480.

See Also

[Distributions](#) for standard distributions, including [dbinom](#) for the binomial, [dpois](#) for the Poisson and [dgeom](#) for the geometric distribution, which is a special case of the negative binomial.

Examples

```
dnbinom2(0:5, 2, sqrt(2))
```

dnull

Null distributions always returns NA

Description

Null distributions always returns NA

Usage

```
dnull(x, ...)
```

Arguments

x	vector of quantiles
...	not used

Examples

```
dnull(c(0.25,0.5,0.75), 0.5, 0.25)
```

doubling_time

Doubling time from growth rate

Description

The unit of doubling times is always days.

Usage

```
doubling_time(x, ...)
```

Arguments

x

proportion or incidence growth rates - EITHER: a dataframe with columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (positive_double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

Any grouping allowed.

OR with columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)

- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

... not used

Value

the same dataframe with additional columns for doubling time or relative doubling time plus confidence intervals.

Examples

```
example_poisson_rt_smooth() %>%
  poisson_locfit_model(window=21) %>%
  doubling_time() %>%
  dplyr::glimpse()
```

dwedge	<i>Wedge distribution</i>
--------	---------------------------

Description

The wedge distribution has a support of 0 to 1 and has a linear probability density function over that support.

Usage

```
dwedge(x, a, lower.tail = TRUE, log = FALSE)
```

Arguments

x	vector of quantiles
a	a gradient from -2 (left skewed) to 2 (right skewed)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log	logical; if TRUE, probabilities p are given as $\log(p)$.

Details

The `rwedge` can be combined with quantile functions to skew standard distributions, or introduce correlation or down weight certain parts of the distribution.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
pwedge(seq(0,1,0.1), a=1)
dwedge(seq(0,1,0.1), a=1)
qwedge(c(0.25,0.5,0.75), a=-1)

stats::cor(
  stats::qnorm(rwedge(1000, a=2)),
  stats::qnorm(rwedge(1000, a=-2))
)
```

fdmy

Format date as dmy

Description

Format date as dmy

Usage

```
fdmy(date)
```

Arguments

date a date to convert

Value

the formatted date

Examples

```
fdmy(Sys.Date())
```

format_ip	<i>Print a summary of an infectivity profile</i>
-----------	--

Description

Print a summary of an infectivity profile

Usage

```
format_ip(ip = i_empirical_ip)
```

Arguments

`ip` the infectivity profile to summarise. `a0` and `a1` columns are optional if `tau` is given. - a dataframe with columns:

- `boot` (anything + `default(1)`) - a bootstrap identifier
- `probability` (proportion) - the probability of new event during this period.
- `a0` (double) - the beginning of the time period (in days)
- `a1` (double) - the end of the time period (in days)

Minimally grouped by: `boot` (and other groupings allowed).

Value

an infectivity profile description

Examples

```
format_ip(example_ganyani_ip())
```

gam_delayed_reporting	<i>Delayed GAM reporting model function generator</i>
-----------------------	---

Description

Delayed GAM reporting model function generator

Usage

```
gam_delayed_reporting(
  window,
  max_delay = 40,
  ...,
  knots_fn = ~gam_knots(.x, window, ...)
)
```

Arguments

window	controls the knot spacing in the GAM (if the default)
max_delay	the maximum delay we expect to model
...	Named arguments passed on to gam_knots
	k alternative to window, if k is given then the behaviour of the knots will be similar to the default mgcv: <code>s(...,k=...)</code> parameter.
knots_fn	a function that takes the data as an input and returns a set of integers as time points for GAM knots, for $s(\text{time})$ term. The default here provides a roughly equally spaced grid determined by window, by a user supplied function could do anything. The input this function is the raw dataframe of data that will be considered for one model fit. It is guaranteed to have at least a time and count column. It is possible to

Details

This function is used to configure a delayed reporting GAM model. The model is of the form:

$\text{count} \sim s(\text{time}, \text{bs} = "cr", k = \text{length}(\text{kts})) + s(\log(\text{tau}), k = 4, \text{pc} = \text{max_delay})$

where tau is the difference between time series observation time and the time of the data point in the time series, and we have multiple observations of the same time series. This function helps specify the knots of the GAM and the maximum expected delay

Value

a list with 2 entries - `model_fn` and `predict` suitable as the input for `poisson_gam_model(model_fn = ..., predict=...)`.

Examples

```
data = example_delayed_observation() %>% dplyr::group_by(obs_time)
cfg = gam_delayed_reporting(14,40)
fit = cfg$model_fn(data)
summary(fit)
```

gam_knots

Derive a set of knot points for a GAM from data

Description

At the moment this does nothing sophisticated. An mostly equally spaced grid of knots with gaps at start and end to prevent over-fitting there. In the future this could look at the number of observations or areas where there is a lot of change to add in more knots.

Usage

```
gam_knots(data = i_incidence_data, window, ..., k = NULL)
```

Arguments

data	the function will be called with incidence data - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
window	the spacing between knots
...	currently not used
k	alternative to window, if k is given then the behaviour of the knots will be similar to the default mgcv::s(...,k=...) parameter.

Value

a vector of times (as a numeric)

Examples

```
gam_knots(example_poisson_rt(), 14)
gam_knots(example_poisson_rt(), k=10)
```

gam_nb_model_fn	<i>Default GAM count negative binomial model.</i>
-----------------	---

Description

This function configures a very simple negative binomial count model and using:

Usage

```
gam_nb_model_fn(window, ..., knots_fn = ~gam_knots(.x, window, ...))
```

Arguments

window	controls the knot spacing in the GAM (if the default)
...	Named arguments passed on to gam_knots k alternative to window, if k is given then the behaviour of the knots will be similar to the default mgcv::s(...,k=...) parameter.
knots_fn	a function that takes the data as an input and returns a set of integers as time points for GAM knots, for s(time) term. The default here provides a roughly equally spaced grid determined by window, by a user supplied function could do anything. The input this function is the raw dataframe of data that will be considered for one model fit. It is guaranteed to have at least a time and count column. It is possible to

Details

```
count ~ s(time, bs = "cr", k = length(kts))
```

The control of knot positions is defined by the `knots_fn` and `window` parameters.

Value

a function suitable as the input for `poisson_gam_model(model_fn = ...)`.

Examples

```
model_fn = gam_nb_model_fn(14)
fit = model_fn(example_poisson_rt()) %>% dplyr::ungroup()
summary(fit)
```

gam_poisson_model_fn	<i>Default GAM count model.</i>
----------------------	---------------------------------

Description

This function configures a very simple poisson count model and using:

Usage

```
gam_poisson_model_fn(window, ..., knots_fn = ~gam_knots(.x, window, ...))
```

Arguments

<code>window</code>	controls the knot spacing in the GAM (if the default)
<code>...</code>	Named arguments passed on to gam_knots
	<code>k</code> alternative to <code>window</code> , if <code>k</code> is given then the behaviour of the knots will be similar to the default <code>mgcv::s(...,k=...)</code> parameter.
<code>knots_fn</code>	a function that takes the data as an input and returns a set of integers as time points for GAM knots, for <code>s(time)</code> term. The default here provides a roughly equally spaced grid determined by <code>window</code> , by a user supplied function could do anything. The input this function is the raw dataframe of data that will be considered for one model fit. It is guaranteed to have at least a time and count column. It is possible to

Details

```
count ~ s(time, bs = "cr", k = length(kts))
```

The control of knot positions is defined by the `knots_fn` and `window` parameters.

Value

a function suitable as the input for `poisson_gam_model(model_fn = ...)`.

Examples

```
model_fn = gam_poisson_model_fn(14)
fit = model_fn(example_poisson_rt()) %>% dplyr::ungroup()
summary(fit)
```

geom_events

Add time series event markers to a time series plot.

Description

The x axis must be a date.

Usage

```
geom_events(
  events = i_events,
  event_label_size = 7,
  event_label_colour = "black",
  event_label_angle = -30,
  event_line_colour = "grey50",
  event_fill_colour = "grey50",
  hide_labels = FALSE,
  guide_axis = ggplot2::derive(),
  x_axis_style = c("date", "time_period"),
  ...
)
```

Arguments

events	Significant events or time spans - a dataframe with columns: - label (character) - the event label - start (date) - the start date, or the date of the event - end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.
event_label_size	how big to make the event label
event_label_colour	the event label colour
event_label_angle	the event label colour
event_line_colour	the event line colour
event_fill_colour	the event area fill

hide_labels	do not show labels at all
guide_axis	a guide axis configuration for the labels (see <code>ggplot2::guide_axis</code> and <code>ggplot2::dup_axis</code>). This can be used to specify a position amongst other things.
x_axis_style	should the x-axis be "date" or a count of "time_period"s since a start date (may be specified in ... or defaults from data)
...	Named arguments passed on to <code>ggplot2::scale_x_date</code>
name	The name of the scale. Used as the axis or legend title. If <code>waiver()</code> , the default, the name of the scale is taken from the first mapping used for that aesthetic. If <code>NULL</code> , the legend title will be omitted.
breaks	One of: • <code>NULL</code> for no breaks • <code>waiver()</code> for the breaks specified by <code>date_breaks</code> • A <code>Date/POSIXct</code> vector giving positions of breaks • A function that takes the limits as input and returns breaks as output
date_breaks	A string giving the distance between breaks like "2 weeks", or "10 years". If both <code>breaks</code> and <code>date_breaks</code> are specified, <code>date_breaks</code> wins. Valid specifications are 'sec', 'min', 'hour', 'day', 'week', 'month' or 'year', optionally followed by 's'.
labels	One of the options below. Please note that when <code>labels</code> is a vector, it is highly recommended to also set the <code>breaks</code> argument as a vector to protect against unintended mismatches. • <code>NULL</code> for no labels • <code>waiver()</code> for the default labels computed by the transformation object • A character vector giving labels (must be same length as <code>breaks</code>) • An expression vector (must be the same length as <code>breaks</code>). See <code>?plot-math</code> for details. • A function that takes the <code>breaks</code> as input and returns labels as output. Also accepts <code>rlang</code> <code>lambda</code> function notation.
date_labels	A string giving the formatting specification for the labels. Codes are defined in <code>strftime()</code> . If both <code>labels</code> and <code>date_labels</code> are specified, <code>date_labels</code> wins.
minor_breaks	One of: • <code>NULL</code> for no breaks • <code>waiver()</code> for the breaks specified by <code>date_minor_breaks</code> • A <code>Date/POSIXct</code> vector giving positions of minor breaks • A function that takes the limits as input and returns minor breaks as output
date_minor_breaks	A string giving the distance between minor breaks like "2 weeks", or "10 years". If both <code>minor_breaks</code> and <code>date_minor_breaks</code> are specified, <code>date_minor_breaks</code> wins. Valid specifications are 'sec', 'min', 'hour', 'day', 'week', 'month' or 'year', optionally followed by 's'.
limits	One of: • <code>NULL</code> to use the default scale range

- A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum
- A function that accepts the existing (automatic) limits and returns new limits. Also accepts rlang `lambda` function notation. Note that setting limits on positional scales will **remove** data outside of the limits. If the purpose is to zoom, use the limit argument in the coordinate system (see `coord_cartesian()`).

expand For position scales, a vector of range expansion constants used to add some padding around the data to ensure that they are placed some distance away from the axes. Use the convenience function `expansion()` to generate the values for the expand argument. The defaults are to expand the scale by 5% on each side for continuous variables, and by 0.6 units on each side for discrete variables.

oob One of:

- Function that handles limits outside of the scale limits (out of bounds). Also accepts rlang `lambda` function notation.
- The default (`scales:::censor()`) replaces out of bounds values with NA.
- `scales:::squish()` for squishing out of bounds values into range.
- `scales:::squish_infinite()` for squishing infinite values into range.

guide A function used to create a guide or its name. See `guides()` for more information.

position For position scales, The position of the axis. left or right for y axes, top or bottom for x axes.

timezone The timezone to use for display on the axes. The default (NULL) uses the timezone encoded in the data.

na.value Missing values will be replaced with this value.

Named arguments passed on to `as.time_period`

unit the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another `time_period`. If `x` is a `time_period`, and the unit is different to that of `x` this will return a rescaled `time_period` using the new units.

start_date the zero time date as something that can be coerced to a date. If the `x` input is already a `time_period` and this is different to its `start_date` then `x` will be recalibrated to use the new start date.

anchor only relevant if `x` is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in `x` it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see `set_defaults()`)

Value

a set of geoms for a time series.

growth_rate_from_incidence

Estimate growth rate from modelled incidence

Description

This assumes a modelled incidence estimate that is log-normal. The exponential growth rate is the first derivative of the mu parameters of this log-normal. On the link scale these are normally distributed. This function assumes that the time series incidence estimates are uncorrelated to estimate the error in the growth rate, which is a conservative approach resulting in more uncertainty in growth rate than might be possible through other methods. This is all based on Savitsky-Golay filters applied to the normally distributed log-incidence estimates.

Usage

```
growth_rate_from_incidence(d = i_incidence_model, window = 7, deg = 2)
```

Arguments

d	a modelled incidence estimate - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - incidence.fit (double) - an estimate of the incidence rate on a log scale - incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale - incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) - incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale) - incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale) Any grouping allowed.
window	the width of the Savitsky-Golay filter - must be odd
deg	the polynomial degree of the filter

Value

the timeseries with growth rate columns: A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (positive_double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate

- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

Any grouping allowed.

Examples

```
data = example_poisson_growth_rate()
tmp2 = data %>%
  poisson_glm_model(window=7,deg=1) %>%
  growth_rate_from_incidence(window = 13, deg=1)

if(interactive()) {
  plot_incidence(tmp2,
    date_labels="%b %y")

  plot_growth_rate(
    tmp2,
    date_labels="%b %y"
  )+
  sim_geom_function(data,colour="red")
}
```

growth_rate_from_prevalence

Estimate relative growth rate from estimated prevalence

Description

[Experimental]

This assumes a prevalence that is logit-normally distributed. The exponential growth rate is the first derivative of the mu parameters of this logit-normal. On the link scale these are normally distributed. This function assumes that the time series incidence estimates are uncorrelated to estimate the error in the growth rate, which is a conservative approach resulting in more uncertainty in growth rate than might be possible through other methods. This is all based on Savitsky-Golay filters applied to the normally distributed logit-proportion estimates.

Usage

```
growth_rate_from_prevalence(d = i_prevalence_model, window = 7, deg = 2)
```

Arguments

- | | |
|---|--|
| d | a modelled proportion estimate - a dataframe with columns: <ul style="list-style-type: none"> • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • prevalence.0.025 (proportion) - lower confidence limit of prevalence (true scale) |
|---|--|

	• prevalence.0.5 (proportion) - median estimate of prevalence (true scale) • prevalence.0.975 (proportion) - upper confidence limit of prevalence (true scale)
	Any grouping allowed.
window	the width of the Savitsky-Golay filter - must be odd
deg	the polynomial degree of the filter

Value

the timeseries with growth rate columns: A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

Examples

```
data = example_poisson_rt_2class()

tmp = data %>%
  proportion_glm_model(window=7,deg=2) %>%
  dplyr::select(time, dplyr::starts_with("proportion")) %>%
  dplyr::rename_with(~ gsub("proportion","prevalence",.x)) %>%
  dplyr::select(-prevalence.fit, -prevalence.se.fit)

tmp = tmp %>%
  growth_rate_from_prevalence(window = 25, deg=1)

plot_growth_rate(
  tmp,
  date_labels="%b %y"
)

data1 = ukc19::ons_infection_survey %>%
  dplyr::filter(name=="England") %>%
  timeseries(count=FALSE)
```

```

tmp2 = data1 %>%
  growth_rate_from_prevalence(window = 25, deg=1)

if(interactive()) {
  plot_growth_rate(
    tmp2,
    date_labels="%b %y",
    mapping = ggplot2::aes(colour=name),
    events = ukc19::timeline
  )
}

```

growth_rate_from_proportion

Estimate relative growth rate from modelled proportion

Description

[Experimental]

This assumes a modelled proportion that is logit-normally distributed. The exponential growth rate is the first derivative of the mu parameters of this logit-normal. On the link scale these are normally distributed. This function assumes that the time series incidence estimates are uncorrelated to estimate the error in the growth rate, which is a conservative approach resulting in more uncertainty in growth rate than might be possible through other methods. This is all based on Savitsky-Golay filters applied to the normally distributed logit-proportion estimates.

Usage

```
growth_rate_from_proportion(d = i_proportion_model, window = 7, deg = 2)
```

Arguments

d	a modelled proportion estimate - a dataframe with columns: • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • proportion.fit (double) - an estimate of the proportion on a logit scale • proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale • proportion.0.025 (proportion) - lower confidence limit of proportion (true scale) • proportion.0.5 (proportion) - median estimate of proportion (true scale) • proportion.0.975 (proportion) - upper confidence limit of proportion (true scale) Any grouping allowed.
window	the width of the Savitsky-Golay filter - must be odd
deg	the polynomial degree of the filter

Value

the timeseries with growth rate columns: A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

Examples

```
data = example_poisson_rt_2class()

tmp2 = data %>%
  proportion_glm_model(window=7,deg=2) %>%
  growth_rate_from_proportion(window = 13, deg=1)

if(interactive()) {
  plot_proportion(tmp2,
    date_labels="%b %y")

  plot_growth_rate(
    tmp2,
    date_labels="%b %y"
  )
}
```

infer_population

Infers a daily baseline population for a timeseries

Description

This function augments any timeseries with a population denominator. The population data may be static estimates, or a set of estimates at time points. The population data may be grouped in which case the grouping might be geographical area or age group or gender for example. The two inputs must have compatible grouping (i.e. all the groups in the population data must be present in the timeseries).

Usage

```
infer_population(df = i_timeseries, pop = i_population_data)
```

Arguments

df	A time series, or a grouped collection of time series. - a dataframe with columns: time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
pop	The population data must be grouped in the same way as df. It might also have a time column as a time_period if the population is not static - a dataframe with columns: population (positive_integer) - Size of population Any grouping allowed.

Value

the df timeseries with additional population column

Examples

```
# The COVID data already has a population column so we are just double checking:
data = example_england_covid_by_age() %>%
  dplyr::rename(pop_old = population)

demog = ukc19::uk_population_2019_by_5yr_age %>% dplyr::group_by(code, name, class)

data %>%
  infer_population(demog) %>%
  dplyr::glimpse()
```

infer_prevalence	<i>Infer the prevalence of disease from incidence estimates and population size.</i>
------------------	--

Description**[Experimental]**

Log-scaled incidence estimates are used to generate samples of incidence. These are convolved with the duration of infection (as the probability still infected) on any given day (as a set of discrete distributions) and the result is evaluated as a fraction of population. The result is fitted to a logit-normal (using moments) and quantiles produced from samples. If test sensitivity is used instead of duration of infection then the prevalence estimate will be the expected proportion of test positives by screening test rather than a true prevalence. Differences between this and the proportion of test positives observed may be due to ascertainment bias or test sensitivity misspecification.

Usage

```
infer_prevalence(
  modelled = i_incidence_model,
  pop = i_population_data,
  ip = i_discrete_ip,
  bootstraps = 1000,
  seed = Sys.time(),
  ...
)
```

Arguments

modelled	Model output from processing the raw dataframe with something like <code>poission_locfit_model</code> - a dataframe with columns: • <code>time</code> (<code>ggoutbreak::time_period + group_unique</code>) - A (usually complete) set of singular observations per unit time as a 'time_period' • <code>incidence.fit</code> (double) - an estimate of the incidence rate on a log scale • <code>incidence.se.fit</code> (positive_double) - the standard error of the incidence rate estimate on a log scale • <code>incidence.0.025</code> (positive_double) - lower confidence limit of the incidence rate (true scale) • <code>incidence.0.5</code> (positive_double) - median estimate of the incidence rate (true scale) • <code>incidence.0.975</code> (positive_double) - upper confidence limit of the incidence rate (true scale) Any grouping allowed.
pop	The population data must be grouped in the same way as <code>modelled</code>. - a dataframe with columns: • <code>population</code> (positive_integer) - Size of population Any grouping allowed.
ip	A discrete distribution representing the duration of infection (or probability of detection of infection). This will not sum to one. - a dataframe with columns: • <code>boot</code> (anything + default(1)) - a bootstrap identifier • <code>probability</code> (proportion) - the probability of new event during this period. • <code>tau</code> (integer + complete) - the days since the index event. Minimally grouped by: <code>boot</code> (and other groupings allowed).
bootstraps	the number of samples to take at each time point. This will be rounded up to a whole multiple of the infection duration distribution length.
seed	a random number seed for reproducibility
...	not used

Value

the modelled input with additional proportion columns

Examples

```
tmp = example_poisson_rt_smooth() %>%
  poisson_locfit_model(window=14) %>%
  infer_prevalence(
    pop = 10000,
    ip = example_ip()
  )

if(interactive()) {
  plot_prevalence(tmp)
}
```

infer_rate_ratio	<i>Calculate a risk ratio from incidence</i>
------------------	--

Description

[Experimental]

This enables incidence rates are able to be compared to a baseline figure for incidence. The baseline could come for example from a population average or average incidence over time. The output is an incidence rate ratio. The `incidence_baseline` column is a rate of events per unit time. The time unit is expected to be the same as that of the date in modelled and this is not checked.

Usage

```
infer_rate_ratio(
  modelled = i_incidence_model,
  base = i_baseline_incidence_data,
  ...
)
```

Arguments

modelled	Model output from something like <code>poisson_locfit_model()</code>. It really makes sense if this is a grouped model. - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - incidence.fit (double) - an estimate of the incidence rate on a log scale - incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale - incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) - incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale) - incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
----------	---

	Any grouping allowed.
base	The baseline data must be grouped in the same way as modelled. It may be a time series but does not have to be. See the example and note this may change in the future. - a dataframe with columns: • baseline_incidence (positive_double) - Baseline raw incidence rate as count data
	Any grouping allowed.
...	not used

Value

a dataframe with incidence rate ratios for each of the classes in modelled. A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rate_ratio.0.025 (positive_double) - lower confidence limit of the rate ratio for a population group
- rate_ratio.0.5 (positive_double) - median estimate of the rate ratio for a population group
- rate_ratio.0.975 (positive_double) - upper confidence limit of the rate ratio for a population group

Any grouping allowed.

Examples

```
# not age stratified
baseline = example_poisson_locfit() %>%
  dplyr::mutate(baseline_incidence = incidence.0.5)

# age stratified rate ratios:
tmp = example_poisson_age_stratified() %>%
  infer_rate_ratio(baseline) %>%
  dplyr::glimpse()
```

infer_risk_ratio	<i>Calculate a normalised risk ratio from proportions</i>
------------------	---

Description

[Experimental]

This assumes that for example, case distribution proportions are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population proportion allows us to compare the relative risk of the outcome in groups, compared to the expected population risk if the outcome is evenly distributed across the population. There may be other proportions other than population fractions that may be useful to compare. At the moment this does not handle any uncertainty.

Usage

```
infer_risk_ratio(
  modelled = i_proportion_model,
  base = i_baseline_proportion_data,
  ...
)
```

Arguments

modelled	Model output from something like <code>proportion_locfit_model()</code> - a dataframe with columns: • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • proportion.fit (double) - an estimate of the proportion on a logit scale • proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale • proportion.0.025 (proportion) - lower confidence limit of proportion (true scale) • proportion.0.5 (proportion) - median estimate of proportion (true scale) • proportion.0.975 (proportion) - upper confidence limit of proportion (true scale) Any grouping allowed.
base	The baseline data must be grouped in the same way as modelled. It may be a time series but does not have to be. - a dataframe with columns: • baseline_proportion (proportion) - Baseline proportion for comparison Any grouping allowed.
...	not used

Value

a dataframe with relative risk / risk ratio columns. A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- risk_ratio.0.025 (positive_double) - lower confidence limit of the excess risk ratio for a population group
- risk_ratio.0.5 (positive_double) - median estimate of the excess risk ratio for a population group
- risk_ratio.0.975 (positive_double) - upper confidence limit of the excess risk ratio for a population group

Any grouping allowed.

Examples

```

demog = ukc19::uk_population_2019_by_5yr_age %>%
  dplyr::filter(name == "England")

tmp = example_proportion_age_stratified() %>%
  infer_risk_ratio(demog) %>%
  dplyr::glimpse()

if(interactive()) {
  plot_growth_phase(tmp,duration = 14*4)+
  ggplot2::scale_colour_viridis_d()+
  ggplot2::coord_cartesian(xlim=c(-0.25,0.25),ylim=c(0.02,20))
}

```

integer_breaks	<i>Strictly integer breaks for continuous scale</i>
----------------	---

Description

Strictly integer breaks for continuous scale

Usage

```
integer_breaks(n = 5, ...)
```

Arguments

n	number of breaks
...	further arguments for methods.

Value

a ggplot breaks function

inv_wallinga_lipsitch	<i>Calculate a growth rate from a reproduction number and an infectivity profile,</i>
-----------------------	---

Description

This solves the relationship between R_t and growth rates as described by Wallinga and Lipsitch, to get a growth rate from R_t and infectivity profile.

Usage

```
inv_wallinga_lipsitch(
  Rt,
  y = i_empirical_ip,
  a1 = seq(0.5, length.out = length(y)),
  a0 = dplyr::lag(a1, default = 0)
)
```

Arguments

Rt	a vector of reproduction numbers
y	an empirical infectivity profile as a probability vector or as a dataframe of format: A dataframe containing the following columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
a1	the end time of the infectivity profile probability estimate (defaults to 0.5,1.5,2.5,...).
a0	the start time of the infectivity profile probability estimate (defaults to 0,0.5,1.5,...).

Details

This function uses a single empirical distribution for the infectivity profile / generation time. If multiple are provided then the average central value is chosen (i.e. this does not propagate uncertainty in infectivity profile)

Value

an vector of growth rates

Examples

```
inv_wallinga_lipsitch(Rt=seq(0.5,2.5,length.out=9), y=example_ip())
```

is.Date

Check whether vector is a date

Description

Check whether vector is a date

Usage

```
is.Date(x)
```

Arguments

`x` a vector to check

Value

TRUE if dates, FALSE otherwise

Examples

```
is.Date(Sys.Date())
```

julian.time_period	<i>Extract Parts of a POSIXt or Date Object</i>
--------------------	---

Description

Extract the weekday, month or quarter, or the Julian time (days since some origin). These are generic functions: the methods for the internal date-time classes are documented here.

Usage

```
## S3 method for class 'time_period'
julian(x, ...)
```

Arguments

`x` an object inheriting from class "POSIXt" or "Date".
`...` arguments for other methods.

Value

weekdays and months return a character vector of names in the locale in use, i.e., [Sys.getlocale\("LC_TIME"\)](#).

quarters returns a character vector of "Q1" to "Q4".

julian returns the number of days (possibly fractional) since the origin, with the origin as a "origin" attribute. All time calculations in R are done ignoring leap-seconds.

Note

Other components such as the day of the month or the year are very easy to compute: just use [as.POSIXlt](#) and extract the relevant component. Alternatively (especially if the components are desired as character strings), use [strftime](#).

See Also

[DateTimeClasses](#), [Date](#); [Sys.getlocale\("LC_TIME"\)](#) crucially for [months\(\)](#) and [weekdays\(\)](#).

Examples

```
## first two are locale dependent:
weekdays(.leap.seconds)
months (.leap.seconds)
quarters(.leap.seconds)

## Show how easily you get month, day, year, day (of {month, week, yr}), ... :
## (remember to count from 0 (!): mon = 0..11, wday = 0..6, etc !!)

##' Transform (Time-)Date vector to convenient data frame :
dt2df <- function(dt, dName = deparse(substitute(dt))) {
  DF <- as.data.frame(unclass(as.POSIXlt( dt )))
  `names<-`(cbind(dt, DF, deparse.level=0L), c(dName, names(DF)))
}
## e.g.,
dt2df(.leap.seconds) # date+time
dt2df(Sys.Date() + 0:9) # date

##' Even simpler: Date -> Matrix - dropping time info {sec,min,hour, isdst}
d2mat <- function(x) simplify2array(unclass(as.POSIXlt(x))[4:7])
## e.g.,
d2mat(seq(as.Date("2000-02-02"), by=1, length.out=30)) # has R 1.0.0's release date

## Julian Day Number (JDN, https://en.wikipedia.org/wiki/Julian\_day)
## is the number of days since noon UTC on the first day of 4317 BCE.
## in the proleptic Julian calendar. To more recently, in
## 'Terrestrial Time' which differs from UTC by a few seconds
## See https://en.wikipedia.org/wiki/Terrestrial\_Time
julian(Sys.Date(), -2440588) # from a day
floor(as.numeric(julian(Sys.time()))) + 2440587.5) # from a date-time
```

labels.time_period	<i>Label a time period</i>
--------------------	----------------------------

Description

Create a set of labels for a time period based on the start and duration of the period. The format is configurable using the start and end dates and the dfmt and ifmt parameters, however if the time period has names then these are used in preference.

Usage

```
## S3 method for class 'time_period'
labels(
  object,
  ...,
  dfmt = "%d/%b",
```

```

    ifmt = "{start} - {end}",
    na.value = "Unknown"
  )

```

Arguments

<code>object</code>	a set of decimal times as a <code>time_period</code>
<code>...</code>	not used
<code>dfmt</code>	a <code>strptime</code> format specification for the format of the date
<code>ifmt</code>	a glue spec referring to start and end of the period as a formatted date
<code>na.value</code>	a label for NA times

Value

a set of character labels for the time

Examples

```

eg = as.time_period(Sys.Date()+0:10*7, unit="1 week", anchor="start")

labels(eg)
labels(eg, ifmt="{start}", dfmt="%d/%b/%y")
labels(eg, ifmt="until {end}", dfmt="%d %b %Y")

# labels retained in constructor:
eg2 = Sys.Date()+0:10*7
names(eg2) = paste0("week ",0:10)
labels(eg2)
labels(as.time_period(eg2, anchor="start"))

```

<code>linelist</code>	<i>Coerce an object to a ggoutbreak compatible case linelist.</i>
-----------------------	---

Description

Coerce an object to a ggoutbreak compatible case linelist.

Usage

```
linelist(x, ...)
```

Arguments

- x An object to coerce
- ... Named arguments passed on to `as.time_period`
- unit the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another `time_period`. If `x` is a `time_period`, and the unit is different to that of `x` this will return a rescaled `time_period` using the new units.
- start_date the zero time date as something that can be coerced to a date. If the `x` input is already a `time_period` and this is different to its `start_date` then `x` will be recalibrated to use the new start date.
- anchor only relevant if `x` is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in `x` it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see `set_defaults()`)

Value

minimally a time stamped linelist dataframe A dataframe containing the following columns:

- time (ggoutbreak::time_period) - A set of events with a timestamp as a `time_period`

Any grouping allowed.

Examples

```
(Sys.Date()+stats::runif(100)*7) %>% linelist()
```

logit_trans	<i>logit scale</i>
-------------	--------------------

Description

Perform logit scaling with correct axis formatting. To not be used directly but with `ggplot` (e.g. `ggplot2::scale_y_continuous(trans = "logit")`)

Usage

```
logit_trans(n = 5, ...)
```

Arguments

- n the number of breaks
- ... not used, for compatibility

Value

A scales object

Examples

```
dplyr::tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
  ggplot2::ggplot(ggplot2::aes(fold_change , pvalue)) +
  ggplot2::geom_point() +
  ggplot2::scale_y_continuous(trans = "logit")
```

make_empirical_ip	<i>Recover a long format infectivity profile from an EpiEstim style matrix</i>
-------------------	--

Description

Recover a long format infectivity profile from an EpiEstim style matrix

Usage

```
make_empirical_ip(omega, normalise = TRUE)
```

Arguments

omega	a matrix of probabilities, starting at time zero, with columns representing one possible infectivity profile, with the first value being the probability at time zero (to 0.5). Alternatively this can be a vector of probabilities for one single profile, resulting in 1 bootstrap.
normalise	is this a probability mass function? In which case we make the sum of this equal to one (the default). If FALSE then the input matrix, or vector is clipped so that its maximum value is one. If this is a number then the PMF is scaled to this value.

Value

a long format ip delay distribution

Examples

```
format_ip(make_empirical_ip(c(0,0,1,1,1,2,2,2,1,1)))
```

make_fixed_ip	<i>Generate a simple discrete infectivity profile from a gamma distribution</i>
---------------	---

Description

Generate a simple discrete infectivity profile from a gamma distribution

Usage

```
make_fixed_ip(mean, sd = sqrt(mean), epiestim_compat = FALSE)
```

Arguments

mean	The mean of a gamma distribution
sd	The sd of a gamma distribution
epiestim_compat	Should the discretisation support EpiEstim (i.e. $\Pr(x < 1) = 0$)

Value

a discrete infectivity profile with 1 bootstrap

Examples

```
tmp = make_fixed_ip(5,2)

if(interactive()) {
  plot_ip(tmp, alpha=1)
}
```

make_gamma_ip	<i>Make an infectivity profile from published data</i>
---------------	--

Description

The infectivity profile is typically fitted to data by MCMC and reported as median and 95% credible intervals, of the mean, and the SD of (usually) a gamma distribution. This function generates a discrete infectivity probability distribution representing the chance that an infectee was infected on any specific day after the infector was infected (given that the infectee was infected).

Usage

```
make_gamma_ip(
  median_of_mean,
  lower_ci_of_mean = median_of_mean,
  upper_ci_of_mean = median_of_mean,
  median_of_sd = sqrt(median_of_mean),
  lower_ci_of_sd = median_of_sd,
  upper_ci_of_sd = median_of_sd,
  correlation = NA,
  n_boots = 100,
  epiestim_compat = FALSE,
  epiestim_sampler = epiestim_compat,
  z_crit = 0.95,
  seed = Sys.time()
)
```

Arguments

median_of_mean, lower_ci_of_mean, upper_ci_of_mean	Quantiles of the infectivity profile mean.
median_of_sd, lower_ci_of_sd, upper_ci_of_sd	Quantiles of the infectivity profile SD.
correlation	the correlation between mean and sd. this is optional and will be inferred if not provided.
n_boots	The number of samples to generate.
epiestim_compat	Use EpiEstim to generate the infectivity profiles. A true value here results in an infectivity profile with probability of 0 for day 0. If false the infectivity profile may include density at zero or even negative values.
epiestim_sampler	Use EpiEstim to generate the random samples using independent truncated normal distributions for mean and SD based on parameters above. If FALSE then it will use a log normal distributions with correlation.
z_crit	the width of the confidence intervals (defaults to 95%).
seed	a RNG seed

Details

EpiEstim generates these distributions by sampling from a truncated normal distribution for both mean and sd. The means and sds thus produced are discretised using a gamma distribution offset by 1 day, to enforce that the probability of infection on day zero is zero.

This constraint changes the shape of the distribution somewhat and may cause a small bias (although there is no ground truth to evaluate). In this function two different sampling and discretisation strategy are provided. The sampler uses log-normal distributions for both mean and SD with a degree of correlation. The discretizer assigns probabilities direct from the CDF of the gamma distribution without offset. This results in non zero values for the probability at time zero and

can only be used with Rt estimation methods that can handle zero/negative serial intervals (e.g. `rt_from_incidence` or `rt_from_renewal`, or `rt_from_growth_rate`). The alternative follows EpiEstims algorithm.

Value

a long format infectivity profile data frame, or a list of dataframes if input is a vector.

Examples

```
# COVID-19 estimates from Ganyani et al 2020.
tmp = make_gamma_ip(5.2, 3.78, 6.78, 1.72, 0.91, 3.93,
  epiestim_sampler=FALSE, epiestim_compat=FALSE)

tmp %>%
  dplyr::group_by(boot) %>%
  dplyr::summarise(
    mean = sum(tau*probability),
    sd = sqrt(sum((tau-sum(tau*probability))^2*probability))
  ) %>%
  dplyr::summarise(
    mean = sprintf("%.2f [%1.2f-%1.2f]",
      stats::quantile(mean,0.5),
      stats::quantile(mean,0.025),
      stats::quantile(mean,0.975)),
    sd = sprintf("%.2f [%1.2f-%1.2f]",
      stats::quantile(sd,0.5),
      stats::quantile(sd,0.025),
      stats::quantile(sd,0.975))
  )

if(interactive()) {
  plot_ip(tmp, alpha=0.1) +
  ggplot2::coord_cartesian(xlim=c(0,15))
}

means = c(3,4,5)
ips = make_gamma_ip(means)

if (interactive()) {
  purrr::map2(ips,means, ~ .x %>% dplyr::mutate(label = sprintf("Mean: %1.2f",.y))) %>%
  purrr::map( ~ plot_ip(.x,alpha=0.1)+ggplot2::facet_wrap(~label))
}
```

Description

The infectivity profile is typically fitted to data by MCMC as a gamma distribution. This function generates a discrete infectivity probability distribution representing the chance that an infectee was infected on any specific day after the infector was infected (given that the infectee was infected), from posterior samples.

Usage

```
make_posterior_ip(
  ...,
  mean,
  sd,
  shape,
  rate,
  scale,
  epiestim_compat = FALSE,
  n_boots = 100
)
```

Arguments

...	not used, must be empty
mean	a vector of gamma distribution means
sd	a vector of gamma distribution sds
shape	a vector of gamma distribution shape parameters
rate	a vector of gamma distribution rate parameters
scale	a vector of gamma distribution scale parameters
epiestim_compat	Use EpiEstim to generate the infectivity profiles. A true value here results in an infectivity profile with probability of 0 for day 0.
n_boots	if there are more posterior samples than this limit then a maximum of n_boots ip distributions will be created (randomly sampled).

Details

If using EpiEstim and `coarseDataTools::dic.fit.mcmc` the output of the MCMC will be a S4 object with a `samples` slot, containing a dataframe of `shape=var1` and `scale=var2` columns. to use this output with `make_posterior_ip` invoke it like this:

```
do.call(make_posterior_ip, SI_fit_clever@samples %>% dplyr::rename(shape=var1, scale=var2))
```

N.b. only one combination of mean and sd, shape and rate, or shape and scale, are required.

Value

a long format ip delay distribution

Examples

```
tmp = make_posterior_ip(
  mean = stats::rnorm(100,5,0.1),
  sd = stats::rnorm(100,1.5,0.1)
)
tmp %>% dplyr::glimpse()
if (interactive()) plot_ip(tmp)
```

make_resampled_ip	<i>Re-sample an empirical IP distribution direct from data</i>
-------------------	--

Description

Suits larger contact tracing data sets where there is a delay between 2 events which may or may not be precisely known.

Usage

```
make_resampled_ip(
  tau,
  min_tau = pmax(tau - 0.5, truncate),
  max_tau = tau + 0.5,
  add_noise = TRUE,
  truncate = 0,
  n_boots = 100,
  seed = Sys.time()
)
```

Arguments

tau	the delay between first and second events
min_tau	the minimum delay for interval censored delays
max_tau	the maximum delay for interval censored delays
add_noise	adds noise to each date point for each replicate
truncate	what is the minimum realistic value of the parameter
n_boots	number of replicates to generate
seed	a random number seed for reproducibility

Value

a long format ip delay distribution

Examples

```
tau = rgamma2(100, 5,2)
ip = make_resampled_ip(min_tau = tau-1, max_tau = tau+1, seed = 100)
if(interactive()) {
  plot_ip(ip,alpha=0.1)
}
```

max_date	<i>The maximum of a set of dates</i>
----------	--------------------------------------

Description

max.Date returns an integer and -Inf for a set of NA dates. This is usually inconvenient.

Usage

```
max_date(x, ...)
```

Arguments

x	a vector of dates
...	ignored

Value

a date. ‘0001-01-01’ if there is no well defined minimum.

Examples

```
max_date(NA)
```

min_date	<i>The minimum of a set of dates</i>
----------	--------------------------------------

Description

min.Date returns an integer and Inf for a set of NA dates. This is usually inconvenient.

Usage

```
min_date(x, ...)
```

Arguments

x	a vector of dates
...	ignored

Value

a date. 9999-12-31 if there is no well defined minimum.

Examples

```
min_date(NA)
```

months.time_period	<i>Extract Parts of a POSIXt or Date Object</i>
--------------------	---

Description

Extract the weekday, month or quarter, or the Julian time (days since some origin). These are generic functions: the methods for the internal date-time classes are documented here.

Usage

```
## S3 method for class 'time_period'
months(x, abbreviate = FALSE)
```

Arguments

x an object inheriting from class "POSIXt" or "Date".

abbreviate logical vector (possibly recycled). Should the names be abbreviated?

Value

weekdays and months return a character vector of names in the locale in use, i.e., `Sys.getlocale("LC_TIME")`.

quarters returns a character vector of "Q1" to "Q4".

julian returns the number of days (possibly fractional) since the origin, with the origin as a "origin" attribute. All time calculations in R are done ignoring leap-seconds.

Note

Other components such as the day of the month or the year are very easy to compute: just use `as.POSIXlt` and extract the relevant component. Alternatively (especially if the components are desired as character strings), use `strftime`.

See Also

`DateTimeClasses`, `Date`; `Sys.getlocale("LC_TIME")` crucially for `months()` and `weekdays()`.

Examples

```
## first two are locale dependent:
weekdays(.leap.seconds)
months (.leap.seconds)
quarters(.leap.seconds)

## Show how easily you get month, day, year, day (of {month, week, yr}), ... :
## (remember to count from 0 (!): mon = 0..11, wday = 0..6, etc !!)

##' Transform (Time-)Date vector to convenient data frame :
dt2df <- function(dt, dName = deparse(substitute(dt))) {
  DF <- as.data.frame(unclass(as.POSIXlt( dt )))
  `names<-`(cbind(dt, DF, deparse.level=0L), c(dName, names(DF)))
}
## e.g.,
dt2df(.leap.seconds) # date+time
dt2df(Sys.Date() + 0:9) # date

##' Even simpler: Date -> Matrix - dropping time info {sec,min,hour, isdst}
d2mat <- function(x) simplify2array(unclass(as.POSIXlt(x))[4:7])
## e.g.,
d2mat(seq(as.Date("2000-02-02"), by=1, length.out=30)) # has R 1.0.0's release date

## Julian Day Number (JDN, https://en.wikipedia.org/wiki/Julian\_day)
## is the number of days since noon UTC on the first day of 4317 BCE.
## in the proleptic Julian calendar. To more recently, in
## 'Terrestrial Time' which differs from UTC by a few seconds
## See https://en.wikipedia.org/wiki/Terrestrial\_Time
julian(Sys.Date(), -2440588) # from a day
floor(as.numeric(julian(Sys.time()))) + 2440587.5 # from a date-time
```

multinomial_nnet_model

Multinomial time-series model.

Description

Takes a list of times, classes and counts, e.g. a COGUK variant like data set with time, (multinomial) class (e.g. variant) and count being the count in that time period. Fits a quadratic B-spline on time to the proportion of the data using `nnet::multinom`, with approx one degree of freedom per class and per window units of the time series.

Usage

```
multinomial_nnet_model(
  d = i_multinomial_input,
  ...,
```

```

window = 14,
frequency = "1 day",
predict = TRUE,
.progress = interactive()
)

```

Arguments

<code>d</code>	A multi-class count input dataframe - a dataframe with columns: • <code>class</code> (factor) - A factor specifying the type of observation. This will be things like variant, or serotype, for a multinomial model. Any missing data points are ignored. • <code>count</code> (positive_integer) - Positive case counts associated with the specified time frame • <code>time</code> (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Minimally grouped by: <code>class</code> (and other groupings allowed).
<code>...</code>	not used
<code>window</code>	a number of data points between knots, smaller values result in less smoothing, large value in more.
<code>frequency</code>	the temporal density of the output estimates.
<code>predict</code>	result a prediction. If false we return the model.
<code>.progress</code>	show a CLI progress bar

Details

Additional groupings are treated as distinct proportions models.

Value

a new dataframe with `time` (as a time period), `class`, and `proportion.0.5`, or a model object

Examples

```

data = example_poisson_rt_2class()
tmp = data %>% multinomial_nnet_model(window=14)

if (interactive()) {
  plot_multinomial(tmp, date_labels="%b %y")
}

```

normalise_count	<i>Calculate a normalised count per capita</i>
-----------------	--

Description

This assumes positive disease counts are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population size allows us to compare groups.

Usage

```
normalise_count(
  raw = i_incidence_data,
  pop = i_population_data,
  ...,
  population_unit = 1e+05,
  normalise_time = FALSE
)
```

Arguments

raw	The count data - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
pop	The population data must be grouped in the same way as raw. - a dataframe with columns: population (positive_integer) - Size of population Any grouping allowed.
...	not used
population_unit	What population unit do you want the count data normalised to e.g. per 100K
normalise_time	The default behaviour for normalising is to keep it in the same time units as the input data. If this parameter is set to TRUE the incidence rates are calculated per year. If given as a lubridate period string e.g. "1 week" then the incidence is calculated over that time period.

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- population (positive_integer) - Size of population
- count (positive_integer) - Positive case counts associated with the specified time frame

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period

Any grouping allowed.

Examples

```
data = example_england_covid_by_age()
demog = ukc19::uk_population_2019_by_5yr_age

data %>%
  normalise_count(demog) %>%
  dplyr::glimpse()
```

normalise_incidence	<i>Calculate a normalised incidence rate per capita</i>
---------------------	---

Description

This assumes positive disease counts are stratified by a population grouping, e.g. geography or age, and we have estimates of the size of that population during that time period. Normalising by population size allows us to compare groups.

Usage

```
normalise_incidence(
  modelled = i_incidence_model,
  pop = i_population_data,
  ...,
  population_unit = 1e+05,
  normalise_time = FALSE
)
```

Arguments

modelled	Model output from processing the raw dataframe with something like <code>poission_locfit_model</code> - a dataframe with columns: • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • incidence.fit (double) - an estimate of the incidence rate on a log scale • incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale • incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) • incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
----------	--

	incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
	Any grouping allowed.
pop	The population data must be grouped in the same way as modelled. - a dataframe with columns: population (positive_integer) - Size of population
	Any grouping allowed.
...	not used
population_unit	what population unit do you want the incidence in e.g. per 100K
normalise_time	The default behaviour for incidence is to keep it in the same time units as the input data. If this parameter is set to TRUE the incidence rates are calculated per year. If given as a lubridate period string e.g. "1 day" then the incidence is calculated over that time period.

Value

a dataframe with incidence rates per unit capita. A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- incidence.per_capita.fit (double) - an estimate of the incidence per capita rate on a log scale
- incidence.per_capita.se.fit (positive_double) - the standard error of the incidence per capita rate estimate on a log scale
- incidence.per_capita.0.025 (positive_double) - lower confidence limit of the incidence per capita rate (true scale)
- incidence.per_capita.0.5 (positive_double) - median estimate of the incidence per capita rate (true scale)
- incidence.per_capita.0.975 (positive_double) - upper confidence limit of the incidence per capita rate (true scale)
- population_unit (double) - The population unit on which the per capita incidence rate is calculated
- time_unit (lubridate::as.period) - The time period over which the per capita incidence rate is calculated

Any grouping allowed.

Examples

```
model = example_poisson_age_stratified()
demog = ukc19::uk_population_2019_by_5yr_age

model %>%
  normalise_incidence(demog) %>%
  dplyr::glimpse()
```

omega_matrix

Generate a infectivity profile matrix from a long format

Description

This converts a long format infectivity profile to something that will work with EpiEstim. In general ggoutbreak will do the conversion internally. This is provided as a utility for examples mostly.

Usage

```
omega_matrix(ip = i_discrete_ip, epiestim_compat = TRUE)
```

Arguments

ip long format infectivity profile - a dataframe with columns:

- boot (anything + default(1)) - a bootstrap identifier
- probability (proportion) - the probability of new event during this period.
- tau (integer + complete) - the days since the index event.

Minimally grouped by: boot (and other groupings allowed).

epestim_compat ensure the matrix works with EpiEstim's `si_from_sample` method by making probability at time zero equal to zero.

Value

a matrix with $0:\max(\text{tau})$ rows and $1:\max(\text{boot})$ columns.

Examples

```
omega_matrix(example_ip())
```

pbeta2

The Beta Distribution

Description

Density, distribution function, quantile function and random generation for the Beta distribution with parameters `shape1` and `shape2` (and optional non-centrality parameter `ncp`).

Usage

```
pbeta2(q, prob, kappa, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles
prob	the mean probability (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

dbeta gives the density, pbeta the distribution function, qbeta the quantile function, and rbeta generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rbeta, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::pbeta\(\)](#)

Examples

```
pbeta2(c(0.25,0.5,0.75), 0.5, 0.25)
```

pcgamma	<i>Cumulative probability: gamma distribution constrained to have mean > sd</i>
---------	--

Description

The following conversion describes the parameters mean and kappa

Usage

```
pcgamma(q, mean, kappa = 1/mean, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles
mean	the mean value on the true scale (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Details

$$\text{shape:}\alpha = \frac{1}{\kappa} \text{rate:}\beta = \frac{1}{\mu \times \kappa} \text{scale:}\sigma = \mu \times \kappa$$

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

```
stats::pgamma()
```

Examples

```
pcgamma(seq(0,4,0.25), 2, 0.5)
```

pgamma2	<i>The Gamma Distribution</i>
---------	-------------------------------

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters shape and scale.

Usage

```
pgamma2(q, mean, sd = sqrt(mean), lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
lower.tail	logical; if TRUE (default), probabilities are P[X<=x] otherwise P[X>x].
log.p	logical; if TRUE, probabilities p are given as log(p).

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

`stats::pgamma()`

Examples

```
pgamma2(seq(0,4,0.25), 2, 1)
```

plnorm2

The Log Normal Distribution

Description

Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has mean equal to meanlog and standard deviation equal to sdlog.

Usage

```
plnorm2(q, mean = 1, sd = sqrt(exp(1) - 1), lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Details

The log normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the logarithm. The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance $Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$ and hence the coefficient of variation is $\sqrt{\exp(\sigma^2) - 1}$ which is approximately σ when that is small (e.g., $\sigma < 1/2$).

Value

dlnorm gives the density, plnorm gives the distribution function, qlnorm gives the quantile function, and rlnorm generates random deviates.

The length of the result is determined by n for rlnorm, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

The cumulative hazard $H(t) = -\log(1 - F(t))$ is `-plnorm(t, r, lower = FALSE, log = TRUE)`.

Source

dlnorm is calculated from the definition (in ‘Details’). [pqr]lnorm are based on the relationship to the normal.

Consequently, they model a single point mass at `exp(meanlog)` for the boundary case `sdlog = 0`.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 14. Wiley, New York.

See Also

[Distributions](#) for other standard distributions, including [dnorm](#) for the normal distribution.

Examples

```
plnorm2(seq(0,4,0.25), 2, 1)
```

plogitnorm

Logit-normal distribution

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
plogitnorm(q, meanlogit = 0, sdlogit = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles ($0 < q < 1$)
meanlogit	the mean on the logit scale
sdlogit	the sd on the logit scale
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
plogitnorm(seq(0.1,0.9,0.1), 0, 1)
```

plogitnorm2	<i>Logit-normal distribution</i>
-------------	----------------------------------

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
plogitnorm2(
  q,
  prob.0.5 = 0.5,
  kappa = 1 - exp(-1),
  lower.tail = TRUE,
  log.p = FALSE
)
```

Arguments

q	vector of quantiles ($0 < q < 1$)
prob.0.5	the median on the true scale
kappa	a dispersion parameter from 0 (none) to 1 maximum dispersion
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
plogitnorm2(seq(0.1,0.9,0.1), 0.75, 0.2)
plogitnorm2(qlogitnorm2(seq(0.1,0.9,0.1), 0.75, 0.2), 0.75, 0.2)
```

plot_cases

*Plot a line-list of cases as a histogram***Description**

Plot a line-list of cases as a histogram

Usage

```
plot_cases(
  raw,
  ...,
  mapping = .check_for_aes(raw, ..., class_aes = "fill"),
  events = i_events
)
```

Arguments

- | | |
|---------|---|
| raw | <p>The raw case data either as a summarised count or as a line-list - EITHER: a dataframe with columns:</p> <ul style="list-style-type: none"> count (positive_integer) - Positive case counts associated with the specified time frame time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period’ <p>Any grouping allowed.</p> <p>OR with columns:</p> <ul style="list-style-type: none"> time (ggoutbreak::time_period) - A set of events with a timestamp as a ‘time_period’ <p>Any grouping allowed.</p> |
| ... | <p>Named arguments passed on to geom_events</p> |
| events | <p>Significant events or time spans - a dataframe with columns:</p> <ul style="list-style-type: none"> label (character) - the event label start (date) - the start date, or the date of the event end (date) - the end date or NA if a single event <p>Any grouping allowed.</p> <p>A default value is defined.</p> |
| mapping | <p>a <code>ggplot2::aes</code> mapping. Most importantly setting the fill to something if there are multiple types of event in the plot. If a class column is present the mapping will default to using this.</p> |

events Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

Any grouping allowed.
A default value is defined.

Value

a ggplot object

Examples

```
with_defaults("2025-01-01" ,"1 day", {
  tmp = dplyr::tibble(
    time = as.time_period( rexpgrowth(100,0.05,40)),
    class = rep(c("one","two","three"), length.out=100)
  )
})

if(interactive()) {
  plot_cases(tmp, mapping=ggplot2::aes(fill = class),linewidth=0.1,colour="white")
}
```

plot_counts	<i>Plot a raw case count timeseries</i>
-------------	---

Description

Plot a raw case count timeseries

Usage

```
plot_counts(raw, ..., mapping = .check_for_aes(raw, ...), events = i_events)
```

Arguments

raw The raw count data, or the raw count data normalised by population (see [normalise_count\(\)](#))
- EITHER: a dataframe with columns:

- population (positive_integer) - Size of population
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- population_unit (double) - The population unit on which the per capita incidence rate is calculated

- `time_unit` (`lubridate::as.period`) - The time period over which the per capita incidence rate is calculated

Any grouping allowed.

OR with columns:

- `count` (`positive_integer`) - Positive case counts associated with the specified time frame
- `time` (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a 'time_period'

Any grouping allowed.

... Named arguments passed on to [geom_events](#)

`events` Significant events or time spans - a dataframe with columns:

- `label` (`character`) - the event label
- `start` (`date`) - the start date, or the date of the event
- `end` (`date`) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

`mapping` a `ggplot2::aes` mapping. Most importantly setting the colour to something if there are multiple incidence timeseries in the plot

`events` Significant events or time spans - a dataframe with columns:

- `label` (`character`) - the event label
- `start` (`date`) - the start date, or the date of the event
- `end` (`date`) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

Value

a `ggplot` object

Examples

```
# example code

tmp = example_england_covid_by_age() %>%
  time_aggregate(count=sum(count)) %>%
  normalise_count(pop=56489700, population_unit=1000, normalise_time=TRUE)

# normalised by England population (56489700 people)

if(interactive()) {
  plot_counts(tmp, colour="blue", size=0.25)
}
```

plot_growth_phase	<i>Plot an incidence or proportion versus growth phase diagram</i>
-------------------	--

Description

Plot an incidence or proportion versus growth phase diagram

Usage

```
plot_growth_phase(
  modelled,
  timepoints = NULL,
  duration = max(dplyr::count(modelled)$n),
  interval = 7,
  mapping = if (interfacer::is_col_present(modelled, class)) {
    ggplot2::aes(colour =
      class)
  } else {
    ggplot2::aes()
  },
  cis = TRUE,
  ...
)
```

Arguments

modelled	Growth rates and incidence / proportion timeseries as the outputs of functions such as <code>proportion_locfit_model</code>, <code>poisson_locfit_model</code>, or similar. - EITHER: a dataframe with columns: • <code>time</code> (<code>ggoutbreak::time_period + group_unique</code>) - A (usually complete) set of singular observations per unit time as a 'time_period' • <code>incidence.per_capita.fit</code> (double) - an estimate of the incidence per capita rate on a log scale • <code>incidence.per_capita.se.fit</code> (positive_double) - the standard error of the incidence per capita rate estimate on a log scale • <code>incidence.per_capita.0.025</code> (positive_double) - lower confidence limit of the incidence per capita rate (true scale) • <code>incidence.per_capita.0.5</code> (positive_double) - median estimate of the incidence per capita rate (true scale) • <code>incidence.per_capita.0.975</code> (positive_double) - upper confidence limit of the incidence per capita rate (true scale) • <code>population_unit</code> (double) - The population unit on which the per capita incidence rate is calculated • <code>time_unit</code> (<code>lubridate::as.period</code>) - The time period over which the per capita incidence rate is calculated
----------	---

- `growth.fit` (double) - an estimate of the growth rate
- `growth.se.fit` (positive_double) - the standard error the growth rate
- `growth.0.025` (double) - lower confidence limit of the growth rate
- `growth.0.5` (double) - median estimate of the growth rate
- `growth.0.975` (double) - upper confidence limit of the growth rate

Any grouping allowed.

OR with columns:

- `time` (`ggoutbreak::time_period` + `group_unique`) - A (usually complete) set of singular observations per unit time as a 'time_period'
- `incidence.fit` (double) - an estimate of the incidence rate on a log scale
- `incidence.se.fit` (positive_double) - the standard error of the incidence rate estimate on a log scale
- `incidence.0.025` (positive_double) - lower confidence limit of the incidence rate (true scale)
- `incidence.0.5` (positive_double) - median estimate of the incidence rate (true scale)
- `incidence.0.975` (positive_double) - upper confidence limit of the incidence rate (true scale)
- `growth.fit` (double) - an estimate of the growth rate
- `growth.se.fit` (positive_double) - the standard error the growth rate
- `growth.0.025` (double) - lower confidence limit of the growth rate
- `growth.0.5` (double) - median estimate of the growth rate
- `growth.0.975` (double) - upper confidence limit of the growth rate

Any grouping allowed.

OR with columns:

- `time` (`ggoutbreak::time_period` + `group_unique`) - A (usually complete) set of singular observations per unit time as a 'time_period'
- `risk_ratio.0.025` (positive_double) - lower confidence limit of the excess risk ratio for a population group
- `risk_ratio.0.5` (positive_double) - median estimate of the excess risk ratio for a population group
- `risk_ratio.0.975` (positive_double) - upper confidence limit of the excess risk ratio for a population group
- `proportion.fit` (double) - an estimate of the proportion on a logit scale
- `proportion.se.fit` (positive_double) - the standard error of proportion estimate on a logit scale
- `proportion.0.025` (proportion) - lower confidence limit of proportion (true scale)
- `proportion.0.5` (proportion) - median estimate of proportion (true scale)
- `proportion.0.975` (proportion) - upper confidence limit of proportion (true scale)
- `relative.growth.fit` (double) - an estimate of the relative growth rate
- `relative.growth.se.fit` (positive_double) - the standard error the relative growth rate

- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

OR with columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

OR with NULL

timepoints	time points (as Date or time_period vector) of dates to plot phase diagrams. If multiple this will result in a sequence of plots as facets. If NULL (the default) it will be the last time point in the series
duration	the length of the growth rate phase trail
interval	the length of time between markers on the phase plot
mapping	a ggplot2::aes() mapping
cis	logical; should the phases be marked with confidence intervals?
...	Named arguments passed on to geom_events

events Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

Named arguments passed on to [ggplot2::facet_wrap](#)

- facets** A set of variables or expressions quoted by `vars()` and defining faceting groups on the rows or columns dimension. The variables can be named (the names are passed to `labeller`).
For compatibility with the classic interface, can also be a formula or character vector. Use either a one sided formula, $\sim a + b$, or a character vector, `c("a", "b")`.
- nrow, ncol** Number of rows and columns.
- scales** Should scales be fixed ("fixed", the default), free ("free"), or free in one dimension ("free_x", "free_y")?
- space** If "fixed" (default), all panels have the same size and the number of rows and columns in the layout can be arbitrary. If "free_x", panels have widths proportional to the length of the x-scale, but the layout is constrained to one row. If "free_y", panels have heights proportional to the length of the y-scale, but the layout is constrained to one column.
- shrink** If TRUE, will shrink scales to fit output of statistics, not raw data. If FALSE, will be range of raw data before statistical summary.
- labeller** A function that takes one data frame of labels and returns a list or data frame of character vectors. Each input column corresponds to one factor. Thus there will be more than one with `vars(cyl, am)`. Each output column gets displayed as one separate line in the strip label. This function should inherit from the "labeller" S3 class for compatibility with `labeller()`. You can use different labeling functions for different kind of labels, for example use `label_parsed()` for formatting facet labels. `label_value()` is used by default, check it for more details and pointers to other options.
- as.table** **[Superseded]** The `as.table` argument is now absorbed into the `dir` argument via the two letter options. If TRUE, the facets are laid out like a table with highest values at the bottom-right. If FALSE, the facets are laid out like a plot with the highest value at the top-right.
- switch** By default, the labels are displayed on the top and right of the plot. If "x", the top labels will be displayed to the bottom. If "y", the right-hand side labels will be displayed to the left. Can also be set to "both".
- drop** If TRUE, the default, all factor levels not used in the data will automatically be dropped. If FALSE, all factor levels will be shown, regardless of whether or not they appear in the data.
- dir** Direction: either "h" for horizontal, the default, or "v", for vertical. When "h" or "v" will be combined with `as.table` to set final layout. Alternatively, a combination of "t" (top) or "b" (bottom) with "l" (left) or "r" (right) to set a layout directly. These two letters give the starting position and the first letter gives the growing direction. For example "rt" will place the first panel in the top-right and starts filling in panels right-to-left.
- strip.position** By default, the labels are displayed on the top of the plot. Using `strip.position` it is possible to place the labels on either of the four sides by setting `strip.position = c("top", "bottom", "left", "right")`
- axes** Determines which axes will be drawn in case of fixed scales. When "margins" (default), axes will be drawn at the exterior margins. "all_x" and "all_y" will draw the respective axes at the interior panels too, whereas "all" will draw all axes at all panels.

`axis.labels` Determines whether to draw labels for interior axes when the scale is fixed and the `axis` argument is not "margins". When "all" (default), all interior axes get labels. When "margins", only the exterior axes get labels, and the interior axes get none. When "all_x" or "all_y", only draws the labels at the interior axes in the x- or y-direction respectively.

Value

a ggplot

Examples

```
data = example_poisson_rt_2class()
tmp2 = data %>% poisson_locfit_model()

timepoints = as.Date(tmp2$time[c(40,80,120,160)])

if(interactive()) {
  plot_growth_phase(tmp2, timepoints, duration=108)
}
```

plot_growth_rate	<i>Growth rate timeseries diagram</i>
------------------	---------------------------------------

Description

Growth rate timeseries diagram

Usage

```
plot_growth_rate(
  modelled,
  ...,
  mapping = .check_for_aes(modelled, ...),
  events = i_events
)
```

Arguments

modelled	a growth rate dataframe. - EITHER: a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - growth.fit (double) - an estimate of the growth rate - growth.se.fit (positive_double) - the standard error the growth rate - growth.0.025 (double) - lower confidence limit of the growth rate - growth.0.5 (double) - median estimate of the growth rate - growth.0.975 (double) - upper confidence limit of the growth rate
----------	--

Any grouping allowed.

OR with columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

...

Named arguments passed on to [geom_events](#)

events Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

Named arguments passed on to [geom_truth](#)

true_df a data frame with a time_period column called time and a value column (name given in true_col). This is optional and will be picked up from the raw parameter if given.

true_col the column name / expression of the true value

true_fmt a list of ggplot formatting to apply to the true value timeseries

... Named arguments passed on to [as.time_period](#)

unit the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another time_period. If x is a time_period, and the unit is different to that of x this will return a rescaled time_period using the new units.

start_date the zero time date as something that can be coerced to a date. If the x input is already a time_period and this is different to its start_date then x will be recalibrated to use the new start date.

anchor only relevant if x is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in x it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see [set_defaults\(\)](#))

mapping

a ggplot2::aes mapping. Most importantly setting the colour to something if there are multiple incidence time series in the plot

events

Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event

- end (date) - the end date or NA if a single event
- Any grouping allowed.
A default value is defined.

Value

a ggplot

Examples

```
data = example_poisson_rt_2class()
tmp2 = data %>% poisson_locfit_model()

if(interactive()) {
  plot_growth_rate(tmp2)
}
```

plot_incidence	<i>Plot an incidence timeseries</i>
----------------	-------------------------------------

Description

Plot an incidence timeseries

Usage

```
plot_incidence(
  modelled,
  raw = i_incidence_data,
  ...,
  mapping = .check_for_aes(modelled, ...),
  events = i_events
)
```

Arguments

- | | |
|----------|--|
| modelled | <p>An optional estimate of the incidence time series. If modelled is missing then it is estimated from raw using a poisson_locfit_model. In this case parameters window and deg may be supplied to control the fit. modelled can also be the output from normalise_incidence in which case the plot uses the per capita rates calculated by that function. - EITHER: a dataframe with columns:</p> <ul style="list-style-type: none"> • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • incidence.per_capita.fit (double) - an estimate of the incidence per capita rate on a log scale |
|----------|--|

- `incidence.per_capita.se.fit` (positive_double) - the standard error of the incidence per capita rate estimate on a log scale
- `incidence.per_capita.0.025` (positive_double) - lower confidence limit of the incidence per capita rate (true scale)
- `incidence.per_capita.0.5` (positive_double) - median estimate of the incidence per capita rate (true scale)
- `incidence.per_capita.0.975` (positive_double) - upper confidence limit of the incidence per capita rate (true scale)
- `population_unit` (double) - The population unit on which the per capita incidence rate is calculated
- `time_unit` (lubridate::as.period) - The time period over which the per capita incidence rate is calculated

Any grouping allowed.

OR with columns:

- `time` (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- `incidence.fit` (double) - an estimate of the incidence rate on a log scale
- `incidence.se.fit` (positive_double) - the standard error of the incidence rate estimate on a log scale
- `incidence.0.025` (positive_double) - lower confidence limit of the incidence rate (true scale)
- `incidence.0.5` (positive_double) - median estimate of the incidence rate (true scale)
- `incidence.0.975` (positive_double) - upper confidence limit of the incidence rate (true scale)

Any grouping allowed.

`raw` The raw count data (optional - if given overlaid as points on top of modelled estimate) - a dataframe with columns:

- `count` (positive_integer) - Positive case counts associated with the specified time frame
- `time` (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

Any grouping allowed.

... Named arguments passed on to [geom_events](#)

`events` Significant events or time spans - a dataframe with columns:

- `label` (character) - the event label
- `start` (date) - the start date, or the date of the event
- `end` (date) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

Named arguments passed on to [poisson_locfit_model](#)

`window` a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - default 14

deg polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - default 2

frequency the density of the output estimates as a time period such as 7 days or 2 weeks. - default 1 day

Named arguments passed on to [geom_truth](#)

true_df a data frame with a time_period column called time and a value column (name given in true_col). This is optional and will be picked up from the raw parameter if given.

true_col the column name / expression of the true value

true_fmt a list of ggplot formatting to apply to the true value timeseries

... Named arguments passed on to [as.time_period](#)

unit the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another time_period. If x is a time_period, and the unit is different to that of x this will return a rescaled time_period using the new units.

start_date the zero time date as something that can be coerced to a date. If the x input is already a time_period and this is different to its start_date then x will be recalibrated to use the new start date.

anchor only relevant if x is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in x it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see [set_defaults\(\)](#))

mapping a ggplot2::aes mapping. Most importantly setting the colour to something if there are multiple incidence timeseries in the plot

events Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

Any grouping allowed.

A default value is defined.

Value

a ggplot object

Examples

```
# example code

tmp = example_poisson_rt_2class()
tmp2 = tmp %>% poisson_locfit_model()

if(interactive()) {
  plot_incidence(tmp2, tmp, size=0.25)
}
```

plot_ip	<i>Plot an infectivity profile</i>
---------	------------------------------------

Description

Plot an infectivity profile

Usage

```
plot_ip(ip = i_empirical_ip, alpha = NULL, ...)
```

Arguments

ip	A long format infectivity profile - a dataframe with columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
alpha	the alpha value of the bootstrap lines
...	passed onto geom_segment for controlling line thickness, alpha etc.

Value

an ggplot object

Examples

```
if(interactive()) {
  plot_ip(example_ganyani_ip())
}
```

plot_multinomial	<i>Plot a multinomial proportions model</i>
------------------	---

Description

A multinomial proportions model will tell you what proportion each class has versus others in the data set. In this case the denominator is the total count across across all classes.

Usage

```
plot_multinomial(
  modelled = i_multinomial_proportion_model,
  ...,
  mapping = ggplot2::aes(fill = class),
  events = i_events,
  normalise = FALSE
)
```

Arguments

- | | |
|-----------|--|
| modelled | <p>the multinomial count data - a dataframe with columns:</p> <ul style="list-style-type: none"> • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • class (factor) - A factor specifying the type of observation. This will be things like variant, or serotype, for a multinomial model. Any missing data points are ignored. • proportion.0.5 (proportion) - median estimate of proportion (true scale) <p>Must be grouped by: class (exactly).</p> |
| ... | <p>Named arguments passed on to geom_events</p> |
| events | <p>Significant events or time spans - a dataframe with columns:</p> <ul style="list-style-type: none"> • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event <p>Any grouping allowed.
A default value is defined.</p> |
| mapping | <p>a ggplot2::aes mapping. Usually this will be left as the default</p> |
| events | <p>Significant events or time spans - a dataframe with columns:</p> <ul style="list-style-type: none"> • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event <p>Any grouping allowed.
A default value is defined.</p> |
| normalise | <p>make sure the probabilities add up to one - this can be a bad idea if you know you may have missing values, on the other hand not all proportions models are guaranteed to add up to one.</p> |

Value

a ggplot

Examples

```
tmp = example_proportion_age_stratified() %>%
  dplyr::group_by(class) %>%
  dplyr::glimpse()

if(interactive()) {
  plot_multinomial(tmp, normalise=TRUE)+
  ggplot2::scale_fill_viridis_d()
}
```

plot_prevalence	<i>Plot a timeseries of disease prevalence</i>
-----------------	--

Description

[Experimental]

Usage

```
plot_prevalence(
  modelled = i_prevalence_model,
  raw = i_proportion_data,
  ...,
  mapping = .check_for_aes(modelled, ...),
  events = i_events
)
```

Arguments

modelled	Prevalence estimates - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - prevalence.0.025 (proportion) - lower confidence limit of prevalence (true scale) - prevalence.0.5 (proportion) - median estimate of prevalence (true scale) - prevalence.0.975 (proportion) - upper confidence limit of prevalence (true scale)
	Any grouping allowed.
raw	Raw proportion data - a dataframe with columns: - denom (positive_integer) - Total test counts associated with the specified time frame - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

	Any grouping allowed.
...	Named arguments passed on to geom_events
events	Significant events or time spans - a dataframe with columns: • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.
mapping	a <code>ggplot2::aes</code> mapping. Most importantly setting the colour to something if there are multiple incidence timeseries in the plot
events	Significant events or time spans - a dataframe with columns: • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.

Value

a ggplot object

Examples

```
if(interactive()) {

  plot_prevalence(
    ukc19::ons_infection_survey %>%
      dplyr::mutate(time = as.time_period(date,"1 day")),
    mapping = ggplot2::aes(colour=name))
}
```

plot_proportion	<i>Plot a proportions timeseries</i>
-----------------	--------------------------------------

Description

Plot a proportions timeseries

Usage

```
plot_proportion(
  modelled = i_proportion_model,
  raw = i_proportion_data,
  ...,
  mapping = .check_for_aes(modelled, ...),
  events = i_events
)
```

Arguments

modelled	Proportion model estimates - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - proportion.fit (double) - an estimate of the proportion on a logit scale - proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale - proportion.0.025 (proportion) - lower confidence limit of proportion (true scale) - proportion.0.5 (proportion) - median estimate of proportion (true scale) - proportion.0.975 (proportion) - upper confidence limit of proportion (true scale) Any grouping allowed.
raw	Raw count data with denominator - a dataframe with columns: - denom (positive_integer) - Total test counts associated with the specified time frame - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
...	Named arguments passed on to geom_events
events	Significant events or time spans - a dataframe with columns: - label (character) - the event label - start (date) - the start date, or the date of the event - end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.
mapping	a ggplot2::aes mapping. Most importantly setting the colour to something if there are multiple incidence timeseries in the plot
events	Significant events or time spans - a dataframe with columns: - label (character) - the event label - start (date) - the start date, or the date of the event - end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.

Value

a ggplot object

Examples

```
tmp = example_poisson_rt_2class() %>%
  proportion_locfit_model(window=21) %>%
  dplyr::glimpse()

if(interactive()) {
  plot_proportion(tmp)+
    ggplot2::scale_fill_viridis_d(aesthetics = c("fill","colour"))
}
```

plot_proportions_data *Plot a raw case count proportion timeseries*

Description

Plot a raw case count proportion timeseries

Usage

```
plot_proportions_data(
  raw = i_proportion_data,
  ...,
  mapping = .check_for_aes(raw, ...),
  events = i_events
)
```

Arguments

raw	The raw count and denominator data - a dataframe with columns: • denom (positive_integer) - Total test counts associated with the specified time frame • count (positive_integer) - Positive case counts associated with the specified time frame • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
...	Named arguments passed on to geom_events
events	Significant events or time spans - a dataframe with columns: • label (character) - the event label • start (date) - the start date, or the date of the event • end (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.
mapping	a ggplot2::aes mapping. Most importantly setting the colour to something if there are multiple count timeseries in the data

events Significant events or time spans - a dataframe with columns:

- label (character) - the event label
- start (date) - the start date, or the date of the event
- end (date) - the end date or NA if a single event

Any grouping allowed.
A default value is defined.

Value

a ggplot object

Examples

```
tmp = example_england_covid_by_age() %>%
  dplyr::filter(class %in% c("50_54", "80_84"))

if(interactive()) {
  plot_proportions_data(tmp, mapping= ggplot2::aes(colour=class))+ggplot2::geom_line()
}
```

plot_rt

Reproduction number timeseries diagram

Description

Reproduction number timeseries diagram

Usage

```
plot_rt(
  modelled = i_reproduction_number,
  ...,
  mapping = .check_for_aes(modelled, ...),
  events = i_events
)
```

Arguments

modelled the modelled Rt estimate - a dataframe with columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number

	• <code>rt.0.975</code> (double) - upper confidence limit of the reproduction number
	Any grouping allowed.
...	Named arguments passed on to geom_events
events	Significant events or time spans - a dataframe with columns: • <code>label</code> (character) - the event label • <code>start</code> (date) - the start date, or the date of the event • <code>end</code> (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.
	Named arguments passed on to geom_truth
	<code>true_df</code> a data frame with a <code>time_period</code> column called <code>time</code> and a <code>value</code> column (name given in <code>true_col</code>). This is optional and will be picked up from the <code>raw</code> parameter if given.
	<code>true_col</code> the column name / expression of the true value
	<code>true_fmt</code> a list of ggplot formatting to apply to the true value timeseries
...	Named arguments passed on to as.time_period
	<code>unit</code> the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another <code>time_period</code> . If <code>x</code> is a <code>time_period</code> , and the unit is different to that of <code>x</code> this will return a rescaled <code>time_period</code> using the new units.
	<code>start_date</code> the zero time date as something that can be coerced to a date. If the <code>x</code> input is already a <code>time_period</code> and this is different to its <code>start_date</code> then <code>x</code> will be recalibrated to use the new start date.
	<code>anchor</code> only relevant if <code>x</code> is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in <code>x</code> it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see set_defaults())
mapping	a <code>ggplot2::aes</code> mapping. Most importantly setting the colour to something if there are multiple incidence time series in the plot
events	Significant events or time spans - a dataframe with columns: • <code>label</code> (character) - the event label • <code>start</code> (date) - the start date, or the date of the event • <code>end</code> (date) - the end date or NA if a single event Any grouping allowed. A default value is defined.

Value

a ggplot timeseries

Examples

```
# example code
if (interactive()) {
```

```

tmp2 = example_poisson_locfit() %>%
  dplyr::filter(as.Date(time) >= "2021-01-01" & as.Date(time) < "2022-01-01") %>%
  rt_from_incidence(ip = example_ganyani_ip())

# comparing RT from growth rates with England consensus Rt
# (N.B. offset by 14 days to align with estimates):

plot_rt(tmp2, colour="blue")+
  ggplot2::geom_errorbar(
    data= ukc19::spim_consensus %>%
      dplyr::filter(date-14 >= "2021-01-01" & date-14 < "2022-01-01"),
    mapping=ggplot2::aes(x=date-14, ymin=rt.low, ymax=rt.high),
    colour="red")
}

```

pnbinom2

*The Negative Binomial Distribution***Description**

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and prob.

Usage

```
pnbinom2(q, mean, sd = sqrt(mean), lower.tail = TRUE, log.p = FALSE)
```

Arguments

q	vector of quantiles.
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as log(p).

Details

The negative binomial distribution with size = n and prob = p has density

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

for $x = 0, 1, 2, \dots, n > 0$ and $0 < p \leq 1$.

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached. The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$.

A negative binomial distribution can also arise as a mixture of Poisson distributions with mean distributed as a gamma distribution (see [pgamma](#)) with scale parameter $(1 - \text{prob})/\text{prob}$ and shape parameter size . (This definition allows non-integer values of size .)

An alternative parametrization (often used in ecology) is by the *mean* μ (see above), and size , the *dispersion parameter*, where $\text{prob} = \text{size}/(\text{size} + \mu)$. The variance is $\mu + \mu^2/\text{size}$ in this parametrization.

If an element of x is not integer, the result of `dnbinom` is zero, with a warning.

The case $\text{size} == 0$ is the distribution concentrated at zero. This is the limiting distribution for size approaching zero, even if μ rather than prob is held constant. Notice though, that the mean of the limit distribution is 0, whatever the value of μ .

The quantile is defined as the smallest value x such that $F(x) \geq p$, where F is the distribution function.

Value

`dnbinom` gives the density, `pnbinom` gives the distribution function, `qnbinom` gives the quantile function, and `rnbinom` generates random deviates.

Invalid size or prob will result in return value `NaN`, with a warning.

The length of the result is determined by n for `rnbinom`, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

`rnbinom` returns a vector of type [integer](#) unless generated values exceed the maximum representable integer when [double](#) values are returned.

Source

`dnbinom` computes via binomial probabilities, using code contributed by Catherine Loader (see [dbinom](#)).

`pnbinom` uses [pbeta](#).

`qnbinom` uses the Cornish–Fisher Expansion to include a skewness correction to a normal approximation, followed by a search.

`rnbinom` uses the derivation as a gamma mixture of Poisson distributions, see

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag, New York. Page 480.

See Also

[Distributions](#) for standard distributions, including [dbinom](#) for the binomial, [dpois](#) for the Poisson and [dgeom](#) for the geometric distribution, which is a special case of the negative binomial.

Examples

```
pnbinom2(0:5, 2, sqrt(2))
```

pnull	<i>Null distributions always returns NA</i>
-------	---

Description

Null distributions always returns NA

Usage

```
pnull(q, ...)
```

Arguments

q	vector of quantiles
...	not used

Examples

```
pnull(c(0.25,0.5,0.75), 0.5, 0.25)
```

poisson_gam_model	<i>GAM poisson time-series model</i>
-------------------	--------------------------------------

Description

This function lets the user supply a fitting function that models incidence, and provides a set of machinery for applying it to groups, extracting incidence, growth rates, and optionally reproduction numbers from the fit(s). This is more advanced than other estimators as there are far more configuration of GAM models, so this aims to provide sensible defaults with the option to bring your own model.

Usage

```
poisson_gam_model(
  d,
  ...,
  frequency = "1 day",
  ip = i_discrete_ip,
  quick = FALSE,
  .progress = interactive()
)
```

Arguments

d	input data - EITHER: a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - obs_time (ggoutbreak::time_period) - The time of the observation of the time series, as a 'time_period' Minimally grouped by: obs_time (and other groupings allowed). OR with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
...	Named arguments passed on to poisson_gam_model.censored model_fn a function that takes data relating to one time series (e.g. the input data d on a group by group basis) and returns a fitted GAM. The default is creates a delayed reporting model gam_delayed_reporting(). predict if the GAM model in model_fn introduces other variables we need to know what their values should be fixed at for prediction. This is a named list of defaults for variables in the model supplied by model_fn. These defaults will be used in prediction. This may be supplied as part of the model function generator (e.g. gam_delayed_reporting(...)\$predict). If this is set to exactly FALSE no prediction is performed and a list column of fitted GAM models returned instead. Named arguments passed on to poisson_gam_model.incidence model_fn a function that takes data relating to one time series (e.g. the input data d on a group by group basis) and returns a fitted GAM. The default creates a simple poisson model based on count alone (gam_poisson_model_fn()). predict if the GAM model in model_fn introduces other variables we need to know what their values should be fixed at for prediction. This is a named list of defaults for variables in the model supplied by model_fn. These defaults will be used in prediction. This may be supplied as part of the model function generator (e.g. gam_delayed_reporting(...)\$predict). If this is set to exactly FALSE no prediction is performed and a list column of fitted GAM models returned instead.
frequency	the density of the output estimates as a time period such as 7 days or 2 weeks.
ip	An infectivity profile (optional) if not given (the default) the Rt value will not be estimated - a dataframe with columns: - boot (anything + default(1)) - a bootstrap identifier - probability (proportion) - the probability of new event during this period. - tau (integer + complete) - the days since the index event.

	Minimally grouped by: boot (and other groupings allowed).
quick	if ip is provided, and quick is TRUE Rt estimation will be done assuming independence which is quicker but less accurate. Setting this to false will use a full variance-covariance matrix.
.progress	show a CLI progress bar

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (positive_double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

Any grouping allowed.

additionally if ip is given: A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
# Simple poisson model
data = example_poisson_rt_smooth()
tmp2 = poisson_gam_model(data, window=7, ip=example_ip(), quick=TRUE)

if (interactive()) {
  plot_incidence(
```

```

    tmp2,
    date_labels="%b %y",
    raw=data
  )

  plot_rt(
    tmp2,
    date_labels="%b %y"
  )+
  sim_geom_function(data, colour="red")
}

# example with delayed observation model.
# This data is all the day by day observations of the whole timeseries from
# the beginning of the outbreak.
data2 = example_delayed_observation()
model = gam_delayed_reporting(window = 14)
tmp3 = data2 %>% poisson_gam_model(
  model_fn = model$model_fn,
  predict = model$predict,
  ip=example_ip())

if (interactive()) {
  plot_incidence(tmp3)+
  ggplot2::geom_line(
    data=data2 %>% dplyr::filter(obs_time %% 10 == 0),
    mapping = ggplot2::aes(x=as.Date(time), y=count, colour=as.factor(obs_time))
  )

  plot_rt(tmp3)
}

```

poisson_glm_model *Poisson time-series model.*

Description

This uses a generalised linear model to fit a quasi-poisson model with a time varying rate as a natural cubic spline with approx one degree of freedom per window units of the time series.

Usage

```

poisson_glm_model(
  d = i_incidence_input,
  ...,
  window = 14,
  frequency = "1 day",
  .progress = interactive()
)

```

Arguments

d	Count model input - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
...	not used and present to allow proportion model to be used in a group_modify
window	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - default 14
frequency	the density of the output estimates as a time period such as 7 days or 2 weeks. - default 1 day
.progress	show a CLI progress bar

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)

Any grouping allowed.

Examples

```
data = example_poisson_growth_rate()

tmp2 = data %>% poisson_glm_model(window=7,deg=2)
tmp3 = data %>% poisson_glm_model(window=14,deg=1)

comp = dplyr::bind_rows(
  tmp2 %>% dplyr::mutate(class="7:2"),
  tmp3 %>% dplyr::mutate(class="14:1"),
) %>% dplyr::group_by(class)

if (interactive()) {
  plot_incidence(comp, date_labels="%b %y", raw=data, true_col=rate)
}
```

poisson_locfit_model *Poisson time-series model.*

Description

Takes a list of times and counts and fits a quasi-poisson model fitted with a log link function to count data using local regression using the package locfit.

Usage

```
poisson_locfit_model(
  d = i_incidence_input,
  ...,
  window = 14,
  deg = 2,
  frequency = "1 day",
  predict = TRUE,
  ip = i_discrete_ip,
  quick = FALSE,
  .progress = interactive()
)
```

Arguments

d	input data - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
...	not used and present to allow proportion model to be used in a group_modify
window	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - default 14
deg	polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - default 2
frequency	the density of the output estimates as a time period such as 7 days or 2 weeks. - default 1 day
predict	result is a prediction dataframe. If false we return the locfit models (advanced). - default TRUE
ip	An infectivity profile (optional) if not given (the default) the Rt value will not be estimated - a dataframe with columns:

- boot (anything + default(1)) - a bootstrap identifier
- probability (proportion) - the probability of new event during this period.
- tau (integer + complete) - the days since the index event.

Minimally grouped by: boot (and other groupings allowed).

quick if ip is provided, and quick is TRUE Rt estimation will be done assuming independence which is quicker but less accurate. Setting this to false will use a full variance-covariance matrix.

.progress show a CLI progress bar

Details

This results is an incidence rate estimate plus an absolute exponential growth rate estimate both based on the time unit of the input data (e.g. for daily data the rate will be cases per day and the growth rate will be daily).

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (positive_double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

Any grouping allowed.

Examples

```
data = example_poisson_rt()
tmp = data %>% poisson_locfit_model(window=14,deg=2, ip=example_ip(), quick=TRUE)
plot_rt(tmp,
  raw = data,
  true_col = rt)

data = example_poisson_growth_rate()
tmp2 = data %>% poisson_locfit_model(window=7,deg=1)
```

```

tmp3 = data %>% poisson_locfit_model(window=14,deg=2)

comp = dplyr::bind_rows(
  tmp2 %>% dplyr::mutate(class="7:1"),
  tmp3 %>% dplyr::mutate(class="14:2"),
) %>% dplyr::group_by(class)

if (interactive()) {
  plot_incidence(
    comp,
    date_labels="%b %y",
    raw=data,
    true_col = rate
  )

  plot_growth_rate(
    comp,
    date_labels="%b %y",
    raw = data,
    true_col = growth
  )
  # sim_geom_function(data,colour="black")
}

```

proportion_glm_model *Binomial time-series model.*

Description

This uses a generalised linear model to fit a quasi-binomial model with a time varying rate as a natural cubic spline with approx one degree of freedom per window units of the time series.

Usage

```

proportion_glm_model(
  d = i_proportion_input,
  ...,
  window = 14,
  frequency = "1 day",
  .progress = interactive()
)

```

Arguments

d Proportion model input - a dataframe with columns:

	• <code>denom</code> (<code>positive_integer</code>) - Total test counts associated with the specified time frame • <code>count</code> (<code>positive_integer</code>) - Positive case counts associated with the specified time frame • <code>time</code> (<code>ggoutbreak::time_period + group_unique</code>) - A (usually complete) set of singular observations per unit time as a 'time_period'
	Any grouping allowed.
<code>...</code>	not used and present to allow proportion model to be used in a <code>group_modify</code>
<code>window</code>	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - default 14
<code>frequency</code>	the density of the output estimates as a time period such as 7 days or 2 weeks. - default 1 day
<code>.progress</code>	show a CLI progress bar

Value

A dataframe containing the following columns:

- `time` (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a `time_period`
- `proportion.fit` (`double`) - an estimate of the proportion on a logit scale
- `proportion.se.fit` (`positive_double`) - the standard error of proportion estimate on a logit scale
- `proportion.0.025` (`proportion`) - lower confidence limit of proportion (true scale)
- `proportion.0.5` (`proportion`) - median estimate of proportion (true scale)
- `proportion.0.975` (`proportion`) - upper confidence limit of proportion (true scale)

Any grouping allowed.

Examples

```
data = example_poisson_rt_2class()

tmp2 = data %>% proportion_glm_model(window=7,deg=2)
tmp3 = data %>% proportion_glm_model(window=14,deg=1)

comp = dplyr::bind_rows(
  tmp2 %>% dplyr::mutate(model="7:2"),
  tmp3 %>% dplyr::mutate(model="14:1"),
) %>% dplyr::group_by(model,class)

if (interactive()) {
  plot_proportion(
    comp,
    date_labels="%b %y",
    mapping=ggplot2::aes(colour=model),
    raw=data
  )
}
```

```

    )+
    ggplot2::facet_wrap(~class)
  }

  # TODO: deal with error conditions
  # "observations with zero weight not used for calculating dispersion

```

```
proportion_locfit_model
```

A binomial proportion estimate and associated exponential growth rate

Description

takes a list of times, counts and a denominator and fits a quasi-binomial model using a logit link function to proportion data using local regression using the package `locfit`.

Usage

```

proportion_locfit_model(
  d = i_proportion_input,
  ...,
  window = 14,
  deg = 1,
  frequency = "1 day",
  predict = TRUE,
  .progress = interactive()
)

```

Arguments

<code>d</code>	the input - a dataframe with columns: • <code>denom</code> (<code>positive_integer</code>) - Total test counts associated with the specified time frame • <code>count</code> (<code>positive_integer</code>) - Positive case counts associated with the specified time frame • <code>time</code> (<code>ggoutbreak::time_period + group_unique</code>) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
<code>...</code>	not used and present to allow proportion model to be used in a <code>group_modify</code>
<code>window</code>	a number of data points defining the bandwidth of the estimate, smaller values result in less smoothing, large value in more. The default value of 14 is calibrated for data provided on a daily frequency, with weekly data a lower value may be preferred. - default 14
<code>deg</code>	polynomial degree (min 1) - higher degree results in less smoothing, lower values result in more smoothing. A degree of 1 is fitting a linear model piece wise. - default 2

frequency	the density of the output estimates as a time period such as 7 days or 2 weeks. - default 1 day
predict	result is a prediction dataframe. If false we return the locfit models (advanced). - default TRUE
.progress	show a CLI progress bar

Details

This expects `d` to contain one combination of:

- time and count and denom columns - e.g. all tests conducted.

This results is a one versus others comparison binomial proportion estimate plus a relative growth rate estimate specifying how much quicker this is growing compared to the growth of the denominator.

The denominator maybe the sum of all subgroups `denom = sum(count)`, e.g. in the situation where there are multiple variants of a disease circulating. In which case the relative growth is that of the subgroup compared to the overall. You can make this a one-versus-others comparison by making the denominator exclude the current item (e.g. `denom = sum(count)-count`).

The denominator can also be used to express the size of the population tested. This gives us a relative growth rate that is different in essence to the previous and may be a better estimate of the true growth rate in the situation where testing effort is variable, or capacity saturated.

Value

A dataframe containing the following columns:

- time (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a `time_period`
- `proportion.fit` (double) - an estimate of the proportion on a logit scale
- `proportion.se.fit` (positive_double) - the standard error of proportion estimate on a logit scale
- `proportion.0.025` (proportion) - lower confidence limit of proportion (true scale)
- `proportion.0.5` (proportion) - median estimate of proportion (true scale)
- `proportion.0.975` (proportion) - upper confidence limit of proportion (true scale)
- `relative.growth.fit` (double) - an estimate of the relative growth rate
- `relative.growth.se.fit` (positive_double) - the standard error the relative growth rate
- `relative.growth.0.025` (double) - lower confidence limit of the relative growth rate
- `relative.growth.0.5` (double) - median estimate of the relative growth rate
- `relative.growth.0.975` (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

Examples

```

data = example_poisson_rt_2class()

tmp2 = data %>% proportion_locfit_model(window=7,deg=2)
tmp3 = data %>% proportion_locfit_model(window=14,deg=1)

comp = dplyr::bind_rows(
  tmp2 %>% dplyr::mutate(model="7:2"),
  tmp3 %>% dplyr::mutate(model="14:1"),
) %>% dplyr::group_by(model,class)

if (interactive()) {
  plot_proportion(
    comp,
    date_labels="%b %y",
    mapping=ggplot2::aes(colour=model),
    raw=data
  )+ggplot2::facet_wrap(~class)
}

```

pwedge

*Wedge distribution***Description**

The wedge distribution has a support of 0 to 1 and has a linear probability density function over that support.

Usage

```
pwedge(q, a, log.p = FALSE)
```

Arguments

q	vector of quantiles
a	a gradient from -2 (left skewed) to 2 (right skewed)
log.p	logical; if TRUE, probabilities p are given as log(p).

Details

The rwedge can be combined with quantile functions to skew standard distributions, or introduce correlation or down weight certain parts of the distribution.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```

pwedge(seq(0,1,0.1), a=1)
dwedge(seq(0,1,0.1), a=1)
qwedge(c(0.25,0.5,0.75), a=-1)

stats::cor(
  stats::qnorm(rwedge(1000, a=2)),
  stats::qnorm(rwedge(1000, a=-2))
)

```

qbeta2

*The Beta Distribution***Description**

Density, distribution function, quantile function and random generation for the Beta distribution with parameters shape1 and shape2 (and optional non-centrality parameter ncp).

Usage

```
qbeta2(p, prob, kappa, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities
prob	the mean probability (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

dbeta gives the density, pbeta the distribution function, qbeta the quantile function, and rbeta generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rbeta, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::qbeta\(\)](#)

Examples

```
qbeta2(c(0.25,0.5,0.75), 0.5, 0.25)
```

qcgamma

*Quantile: gamma distribution constrained to have mean > sd***Description**

The following conversion describes the parameters mean and kappa

Usage

```
qcgamma(p, mean, kappa = 1/mean, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities
mean	the mean value on the true scale (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Details

$$\text{shape: } \alpha = \frac{1}{\kappa} \quad \text{rate: } \beta = \frac{1}{\mu \times \kappa} \quad \text{scale: } \sigma = \mu \times \kappa$$

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::qgamma\(\)](#)

Examples

```
qcgamma(c(0.25, 0.5, 0.75), 2, 0.5)
```

qgamma2*The Gamma Distribution*

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters shape and scale.

Usage

```
qgamma2(p, mean, sd = sqrt(mean), lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::qgamma\(\)](#)

Examples

```
qgamma2(c(0.25, 0.5, 0.75), 2, 1)
```

Description

Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has mean equal to `meanlog` and standard deviation equal to `sdlog`.

Usage

```
qlnorm2(p, mean = 1, sd = sqrt(exp(1) - 1), lower.tail = TRUE, log.p = FALSE)
```

Arguments

<code>p</code>	vector of probabilities.
<code>mean</code>	the mean value on the true scale (vectorised)
<code>sd</code>	the standard deviation on the true scale (vectorised)
<code>lower.tail</code>	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
<code>log.p</code>	logical; if TRUE, probabilities <code>p</code> are given as $\log(p)$.

Details

The log normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the logarithm. The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance $Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$ and hence the coefficient of variation is $\sqrt{\exp(\sigma^2) - 1}$ which is approximately σ when that is small (e.g., $\sigma < 1/2$).

Value

`dlnorm` gives the density, `plnorm` gives the distribution function, `qlnorm` gives the quantile function, and `rlnorm` generates random deviates.

The length of the result is determined by `n` for `rlnorm`, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than `n` are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

The cumulative hazard $H(t) = -\log(1 - F(t))$ is `-plnorm(t, r, lower = FALSE, log = TRUE)`.

Source

dlnorm is calculated from the definition (in ‘Details’). [pqr]lnorm are based on the relationship to the normal.

Consequently, they model a single point mass at $\exp(\text{meanlog})$ for the boundary case $\text{sdlog} = 0$.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 14. Wiley, New York.

See Also

[Distributions](#) for other standard distributions, including [dnorm](#) for the normal distribution.

Examples

```
qlnorm2(c(0.25,0.5,0.72), 2, 1)
```

qlogitnorm	<i>Logit-normal distribution</i>
------------	----------------------------------

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
qlogitnorm(p, meanlogit = 0, sdlogit = 1, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities
meanlogit	the mean on the logit scale
sdlogit	the sd on the logit scale
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
qlogitnorm(c(0.25,0.5,0.75), 0, 1)
```

qlogitnorm2	<i>Logit-normal distribution</i>
-------------	----------------------------------

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
qlogitnorm2(
  p,
  prob.0.5 = 0.5,
  kappa = 1 - exp(-1),
  lower.tail = TRUE,
  log.p = FALSE
)
```

Arguments

p	vector of probabilities
prob.0.5	the median on the true scale
kappa	a dispersion parameter from 0 (none) to 1 maximum dispersion
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
qlnorm2(c(0.25, 0.5, 0.72), 2, 1)
```

qnbinom2	<i>The Negative Binomial Distribution</i>
----------	---

Description

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and prob.

Usage

```
qnbinom2(p, mean, sd = sqrt(mean), lower.tail = TRUE, log.p = FALSE)
```

Arguments

<code>p</code>	vector of probabilities.
<code>mean</code>	the mean value on the true scale (vectorised)
<code>sd</code>	the standard deviation on the true scale (vectorised)
<code>lower.tail</code>	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
<code>log.p</code>	logical; if TRUE, probabilities <code>p</code> are given as $\log(p)$.

Details

The negative binomial distribution with $\text{size} = n$ and $\text{prob} = p$ has density

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

for $x = 0, 1, 2, \dots, n > 0$ and $0 < p \leq 1$.

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached. The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$.

A negative binomial distribution can also arise as a mixture of Poisson distributions with mean distributed as a gamma distribution (see [pgamma](#)) with scale parameter $(1 - \text{prob})/\text{prob}$ and shape parameter `size`. (This definition allows non-integer values of `size`.)

An alternative parametrization (often used in ecology) is by the *mean* μ (see above), and `size`, the *dispersion parameter*, where $\text{prob} = \text{size}/(\text{size} + \mu)$. The variance is $\mu + \mu^2/\text{size}$ in this parametrization.

If an element of `x` is not integer, the result of `dnbinom` is zero, with a warning.

The case `size == 0` is the distribution concentrated at zero. This is the limiting distribution for `size` approaching zero, even if μ rather than `prob` is held constant. Notice though, that the mean of the limit distribution is 0, whatever the value of μ .

The quantile is defined as the smallest value x such that $F(x) \geq p$, where F is the distribution function.

Value

`dnbinom` gives the density, `pnbinom` gives the distribution function, `qnbinom` gives the quantile function, and `rnbinom` generates random deviates.

Invalid `size` or `prob` will result in return value NaN, with a warning.

The length of the result is determined by `n` for `rnbinom`, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than `n` are recycled to the length of the result. Only the first elements of the logical arguments are used.

`rnbinom` returns a vector of type [integer](#) unless generated values exceed the maximum representable integer when [double](#) values are returned.

Source

dnbinom computes via binomial probabilities, using code contributed by Catherine Loader (see [dbinom](#)).

pnbinom uses [pbeta](#).

qnbinom uses the Cornish–Fisher Expansion to include a skewness correction to a normal approximation, followed by a search.

rnbinom uses the derivation as a gamma mixture of Poisson distributions, see

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag, New York. Page 480.

See Also

[Distributions](#) for standard distributions, including [dbinom](#) for the binomial, [dpois](#) for the Poisson and [dgeom](#) for the geometric distribution, which is a special case of the negative binomial.

Examples

```
qnbinom2(c(0.25,0.5,0.75), 5, sqrt(5))
```

qnull

Null distributions always returns NA

Description

Null distributions always returns NA

Usage

```
qnull(p, ...)
```

Arguments

p	vector of probabilities
...	not used

Examples

```
qnull(c(0.25,0.5,0.75), 0.5, 0.25)
```

quantify_lag

Identify estimate lags in a model

Description

A specific parameter or set of parameters can be estimated by a pipeline. This function applies the pipeline to a synthetic epidemic with sawtooth incidence resulting from a stepped growth rate function. The lag between synthetic input and estimate is assessed by minimising the root mean square error of input and estimated based on different lag offsets.

Usage

```
quantify_lag(pipeline, ip = i_empirical_ip, lags = -10:30)
```

Arguments

pipeline	a function taking an input dataset and an infectivity profile as inputs and producing an estimate as output. This is the whole parametrised pipeline including any other inputs. This can be a purrr style function, in which case the <code>.x</code> variable is the input dataset and <code>.y</code> is the infectivity profile.
ip	the infectivity profile. - a dataframe with columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
lags	a vector with the delays to test. Defaults to -10 to +30 days

Value

a lag analysis dataframe containing the estimate type and the lag in days that the estimate is behind the actual observation

Examples

```
set_default_start("2025-01-01")
# lags from a locfit incidence model with Rt estimation.
# This model has no estimator lag:
pipeline = ~ .x %>% poisson_locfit_model() %>% rt_from_incidence(ip = .y)
quantify_lag(pipeline, ip = example_ip())

# lags from an epiestim Rt estimation
# this model's lags depend on the infectivity profile.
# In this case it is 8 days
pipeline2 = ~ .x %>% rt_epiestim(ip = .y)
quantify_lag(pipeline2, ip=example_ip() )
```

quarters.time_period *Extract Parts of a POSIXt or Date Object*

Description

Extract the weekday, month or quarter, or the Julian time (days since some origin). These are generic functions: the methods for the internal date-time classes are documented here.

Usage

```
## S3 method for class 'time_period'
quarters(x, ...)
```

Arguments

`x` an object inheriting from class "POSIXt" or "Date".
`...` arguments for other methods.

Value

weekdays and months return a character vector of names in the locale in use, i.e., `Sys.getlocale("LC_TIME")`.

quarters returns a character vector of "Q1" to "Q4".

julian returns the number of days (possibly fractional) since the origin, with the origin as a "origin" attribute. All time calculations in R are done ignoring leap-seconds.

Note

Other components such as the day of the month or the year are very easy to compute: just use `as.POSIXlt` and extract the relevant component. Alternatively (especially if the components are desired as character strings), use `strftime`.

See Also

`DateTimeClasses`, `Date`; `Sys.getlocale("LC_TIME")` crucially for `months()` and `weekdays()`.

Examples

```
## first two are locale dependent:
weekdays(.leap.seconds)
months (.leap.seconds)
quarters(.leap.seconds)

## Show how easily you get month, day, year, day (of {month, week, yr}), ... :
## (remember to count from 0 (!): mon = 0..11, wday = 0..6, etc !!)

##' Transform (Time-)Date vector to convenient data frame :
dt2df <- function(dt, dName = deparse(substitute(dt))) {
  DF <- as.data.frame(unclass(as.POSIXlt( dt )))
```

```

    `names<-`(cbind(dt, DF, deparse.level=0L), c(dName, names(DF)))
  }
  ## e.g.,
  dt2df(.leap.seconds)    # date+time
  dt2df(Sys.Date() + 0:9) # date

  ##' Even simpler: Date -> Matrix - dropping time info {sec,min,hour, isdst}
  d2mat <- function(x) simplify2array(unclass(as.POSIXlt(x))[4:7])
  ## e.g.,
  d2mat(seq(as.Date("2000-02-02"), by=1, length.out=30)) # has R 1.0.0's release date

  ## Julian Day Number (JDN, https://en.wikipedia.org/wiki/Julian\_day)
  ## is the number of days since noon UTC on the first day of 4317 BCE.
  ## in the proleptic Julian calendar. To more recently, in
  ## 'Terrestrial Time' which differs from UTC by a few seconds
  ## See https://en.wikipedia.org/wiki/Terrestrial\_Time
  julian(Sys.Date(), -2440588) # from a day
  floor(as.numeric(julian(Sys.time()))) + 2440587.5 # from a date-time

```

qwedge

Wedge distribution

Description

The wedge distribution has a support of 0 to 1 and has a linear probability density function over that support.

Usage

```
qwedge(p, a, lower.tail = TRUE, log.p = FALSE)
```

Arguments

p	vector of probabilities
a	a gradient from -2 (left skewed) to 2 (right skewed)
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
log.p	logical; if TRUE, probabilities p are given as $\log(p)$.

Details

The `rwedge` can be combined with quantile functions to skew standard distributions, or introduce correlation or down weight certain parts of the distribution.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```

pwedge(seq(0,1,0.1), a=1)
dwedge(seq(0,1,0.1), a=1)
qwedge(c(0.25,0.5,0.75), a=-1)

stats::cor(
  stats::qnorm(rwedge(1000, a=2)),
  stats::qnorm(rwedge(1000, a=-2))
)

```

rbern

*A random Bernoulli sample as a logical value***Description**

A random Bernoulli sample as a logical value

Usage

```
rbern(n, prob)
```

Arguments

n	number of observations
prob	the mean probability (vectorised)

Value

a vector of logical values of size n

Examples

```
table(rbern(100, 0.25))
```

rbeta2

*The Beta Distribution***Description**

Density, distribution function, quantile function and random generation for the Beta distribution with parameters shape1 and shape2 (and optional non-centrality parameter ncp).

Usage

```
rbeta2(n, prob, kappa)
```

Arguments

n	number of observations
prob	the mean probability (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)

Value

dbeta gives the density, pbeta the distribution function, qbeta the quantile function, and rbeta generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rbeta, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::rbeta\(\)](#)

Examples

```
rbeta2(3, c(0.1,0.5,0.9),0.1)
```

rcategorical

Sampling from the multinomial equivalent of the Bernoulli distribution

Description

Sampling from the multinomial equivalent of the Bernoulli distribution

Usage

```
rcategorical(n, prob, factor = FALSE)
```

Arguments

n	a sample size
prob	a (optionally named) vector of probabilities that will be normalised to sum to 1
factor	if not FALSE then factor levels will either be taken from the names of prob first, or if this is a character vector from this.

Value

a vector of random class labels of length n. Labels come from names of prob or from a character vector in factor.

Examples

```

prob = c("one"=0.1,"two"=0.2,"seven"=0.7)
table(rcategorical(1000,prob))
rcategorical(10,prob,factor=TRUE)
rcategorical(10,rep(1,26),factor=letters)

```

rcgamma

*Sampling: gamma distribution constrained to have mean > sd***Description**

The following conversion describes the parameters mean and kappa

Usage

```
rcgamma(n, mean, kappa = 1/mean)
```

Arguments

n	number of observations
mean	the mean value on the true scale (vectorised)
kappa	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)

Details

$$\text{shape: } \alpha = \frac{1}{\kappa} \quad \text{rate: } \beta = \frac{1}{\mu \times \kappa} \quad \text{scale: } \sigma = \mu \times \kappa$$

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::rgamma\(\)](#)

Examples

```
rcgamma(10, 2, 0.5)
```

rdisgamma

Random count data from a discrete gamma distribution

Description

For count data that is under-dispersed we can use a gamma distribution rounded to the nearest whole number. This is the same as the method of discretisation in `make_gamma_ip`, and suits delay distributions where there is less variability than can be represented with a Poisson or negative binomial distribution.

Usage

```
rdisgamma(n, mean, sd, kappa)
```

Arguments

<code>n</code>	number of observations
<code>mean</code>	the mean value on the true scale (vectorised)
<code>sd</code>	the standard deviation on the true scale (vectorised)
<code>kappa</code>	a coefficient of variation. where 0 is no variability and 1 is maximally variability (vectorised)

Value

an integer valued vector from a gamma distribution.

See Also

`stats::rgamma()`

Examples

```
rdisgamma(10, 2, 1)
```

reband_discrete

Reband any discrete distribution

Description

e.g. age banded population, or a discrete probability distribution e.g. a serial interval distribution. This method fits a monotonically increasing spline to the cumulative distribution (including the upper and lower limits) and interpolating using that spline to the new cut points.

Usage

```
reband_discrete(
  x,
  y,
  xout,
  xlim = c(0, NA),
  ytotal = c(0, sum(y)),
  digits = 0,
  labelling = c("positive_integer", "inclusive", "exclusive"),
  sep = "-"
)
```

Arguments

x	a set of upper limits of bands, e.g. for age: 0-14;15-64;65-79;80+ is 15,65,80,NA
y	a set of quantities for each band e.g. population figures
xout	a set of new upper limits
xlim	Upper and lower limits for x. if the last band is e.g 80+ in the input and we want to know the 85+ band in the output some kind of maximum upper limit is needed to interpolate to.
ytotal	upper and lower limits for y. If the interpolation values fall outside of x then the minimum and maximum limits of y are given by this. This would be c(0,1) for a probability distribution, for example.
digits	if the xout value is continuous then how many significant figures to put in the labels
labelling	are the xout values interpretable as an inclusive upper limit, or an exclusive upper limit, or as an upper limit of an ‘positive_integer’ quantity
sep	separator for names e.g. 18-24 or 18 to 24

Value

a re-banded set of discrete values, guaranteed to sum to the same as y

Examples

```
england_demographics = ukc19::uk_population_2019_by_10yr_age %>%
  dplyr::filter(name=="England")

ul = stringr::str_extract(england_demographics$class, "_[0-9]+", group = 1) %>%
  as.numeric()
# ul is currently inclusive so add 1:
ul = ul + 1

tmp = reband_discrete(
  ul, england_demographics$population,
  c(5,10,15,40,80), xlim=c(0,120))

tmp
```

```
sum(tmp)
sum(england_demographics$population)
```

rescale_model

Rescale a timeseries in the temporal dimension

Description

Sometimes we may have, for example, modelled incidence or growth rates on weekly data resulting in cases per week and growth rate per week. We may wish to use this to estimate the reproduction number, using algorithms that assume a daily incidence. Not everything has a dependence on time, and things such as proportions, or prevalence will not change.

Usage

```
rescale_model(df = i_timeseries, time_unit)
```

Arguments

df A data frame containing modelled output. This will modify the following columns if present:

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- incidence.fit (double) - an estimate of the incidence rate on a log scale
- incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale
- incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale)
- incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale)
- incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)

Any grouping allowed.

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- growth.fit (double) - an estimate of the growth rate
- growth.se.fit (positive_double) - the standard error the growth rate
- growth.0.025 (double) - lower confidence limit of the growth rate
- growth.0.5 (double) - median estimate of the growth rate
- growth.0.975 (double) - upper confidence limit of the growth rate

Any grouping allowed.

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- proportion.fit (double) - an estimate of the proportion on a logit scale
- proportion.se.fit (positive_double) - the standard error of proportion estimate on a logit scale
- proportion.0.025 (proportion) - lower confidence limit of proportion (true scale)
- proportion.0.5 (proportion) - median estimate of proportion (true scale)
- proportion.0.975 (proportion) - upper confidence limit of proportion (true scale)
- relative.growth.fit (double) - an estimate of the relative growth rate
- relative.growth.se.fit (positive_double) - the standard error the relative growth rate
- relative.growth.0.025 (double) - lower confidence limit of the relative growth rate
- relative.growth.0.5 (double) - median estimate of the relative growth rate
- relative.growth.0.975 (double) - upper confidence limit of the relative growth rate

Any grouping allowed.

time_unit a lubridate period string such as "1 day"

Value

the same time series with different time unit, and adjusted incidence and growth rate figures.

Examples

```
sim = sim_poisson_model(time_unit = "1 week")
incidence = sim %>% poisson_locfit_model(frequency = "1 day", deg = 2, window=5)
incidence2 = incidence %>% rescale_model(time_unit = "1 day")
incidence2 %>% dplyr::glimpse()
```

rexpgrowth

Randomly sample incident times in an exponentially growing process

Description

Randomly sample incident times in an exponentially growing process

Usage

```
rexpgrowth(n, r, t_end, t_start = 0)
```

Arguments

n	the number of items to sample
r	and exponential growth rate (per unit time)
t_end	the end of the observation period
t_start	the start of the observation period

Value

a vector of n samples from an exponential growth process

Examples

```
graphics::hist(rexpgrowth(1000,0.1,40), breaks=40)
graphics::hist(rexpgrowth(1000,-0.1,40), breaks=40)
```

rexpgrowthI0	<i>Randomly sample incident times in an exponentially growing process with initial case load</i>
--------------	--

Description

Randomly sample incident times in an exponentially growing process with initial case load

Usage

```
rexpgrowthI0(I0, r, t_end, t_start = 0)
```

Arguments

I0	the expected number of cases observed in the first day
r	and exponential growth rate (per unit time)
t_end	the end of the observation period
t_start	the start of the observation period

Value

a vector of n samples from an exponential growth process

Examples

```
graphics::hist(rexpgrowthI0(10,0.1,20), breaks=40)
graphics::hist(rexpgrowthI0(1000,-0.1,40), breaks=40)
```

rgamma2	<i>The Gamma Distribution</i>
---------	-------------------------------

Description

Density, distribution function, quantile function and random generation for the Gamma distribution with parameters shape and scale.

Usage

```
rgamma2(n, mean, sd = sqrt(mean))
```

Arguments

n	number of observations
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)

Value

dgamma gives the density, pgamma gives the distribution function, qgamma gives the quantile function, and rgamma generates random deviates.

Invalid arguments will result in return value NaN, with a warning.

The length of the result is determined by n for rgamma, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

See Also

[stats::rgamma\(\)](#)

Examples

```
rgamma2(10, 2, 1)
```

rlnorm2

*The Log Normal Distribution***Description**

Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has mean equal to meanlog and standard deviation equal to sdlog.

Usage

```
rlnorm2(n, mean = 1, sd = sqrt(exp(1) - 1))
```

Arguments

n	number of observations. If length(n) > 1, the length is taken to be the number required.
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)

Details

The log normal distribution has density

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma x} e^{-(\log(x)-\mu)^2/2\sigma^2}$$

where μ and σ are the mean and standard deviation of the logarithm. The mean is $E(X) = \exp(\mu + 1/2\sigma^2)$, the median is $med(X) = \exp(\mu)$, and the variance $Var(X) = \exp(2\mu + \sigma^2)(\exp(\sigma^2) - 1)$ and hence the coefficient of variation is $\sqrt{\exp(\sigma^2) - 1}$ which is approximately σ when that is small (e.g., $\sigma < 1/2$).

Value

dlnorm gives the density, plnorm gives the distribution function, qlnorm gives the quantile function, and rlnorm generates random deviates.

The length of the result is determined by n for rlnorm, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

Note

The cumulative hazard $H(t) = -\log(1 - F(t))$ is -plnorm(t, r, lower = FALSE, log = TRUE).

Source

dlnorm is calculated from the definition (in ‘Details’). [pqr]lnorm are based on the relationship to the normal.

Consequently, they model a single point mass at $\exp(\text{meanlog})$ for the boundary case $\text{sdlog} = 0$.

References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

Johnson, N. L., Kotz, S. and Balakrishnan, N. (1995) *Continuous Univariate Distributions*, volume 1, chapter 14. Wiley, New York.

See Also

[Distributions](#) for other standard distributions, including [dnorm](#) for the normal distribution.

Examples

```
rlnorm2(10, 2, 1)
```

rlogitnorm

Logit-normal distribution

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
rlogitnorm(n, meanlogit = 0, sdlogit = 1)
```

Arguments

n	number of observations
meanlogit	the mean on the logit scale
sdlogit	the sd on the logit scale

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
rlogitnorm(10, 0, 1)
```

rlogitnorm2	<i>Logit-normal distribution</i>
-------------	----------------------------------

Description

The logit-normal distribution has a support of 0 to 1.

Usage

```
rlogitnorm2(n, prob.0.5 = 0.5, kappa = 1 - exp(-1))
```

Arguments

n	number of observations
prob.0.5	the median on the true scale
kappa	a dispersion parameter from 0 (none) to 1 maximum dispersion

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
mean(rlogitnorm2(10000, 0.75, 0.2))
```

rnbinom2	<i>The Negative Binomial Distribution</i>
----------	---

Description

Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and prob.

Usage

```
rnbinom2(n, mean, sd = sqrt(mean))
```

Arguments

n	number of observations. If length(n) > 1, the length is taken to be the number required.
mean	the mean value on the true scale (vectorised)
sd	the standard deviation on the true scale (vectorised)

Details

The negative binomial distribution with $\text{size} = n$ and $\text{prob} = p$ has density

$$p(x) = \frac{\Gamma(x+n)}{\Gamma(n)x!} p^n (1-p)^x$$

for $x = 0, 1, 2, \dots, n > 0$ and $0 < p \leq 1$.

This represents the number of failures which occur in a sequence of Bernoulli trials before a target number of successes is reached. The mean is $\mu = n(1-p)/p$ and variance $n(1-p)/p^2$.

A negative binomial distribution can also arise as a mixture of Poisson distributions with mean distributed as a gamma distribution (see [pgamma](#)) with scale parameter $(1 - \text{prob})/\text{prob}$ and shape parameter size . (This definition allows non-integer values of size .)

An alternative parametrization (often used in ecology) is by the *mean* μ (see above), and *size*, the *dispersion parameter*, where $\text{prob} = \text{size}/(\text{size} + \mu)$. The variance is $\mu + \mu^2/\text{size}$ in this parametrization.

If an element of x is not integer, the result of `dnbinom` is zero, with a warning.

The case $\text{size} == 0$ is the distribution concentrated at zero. This is the limiting distribution for size approaching zero, even if μ rather than prob is held constant. Notice though, that the mean of the limit distribution is 0, whatever the value of μ .

The quantile is defined as the smallest value x such that $F(x) \geq p$, where F is the distribution function.

Value

`dnbinom` gives the density, `pnbinom` gives the distribution function, `qnbinom` gives the quantile function, and `rnbinom` generates random deviates.

Invalid size or prob will result in return value `NaN`, with a warning.

The length of the result is determined by n for `rnbinom`, and is the maximum of the lengths of the numerical arguments for the other functions.

The numerical arguments other than n are recycled to the length of the result. Only the first elements of the logical arguments are used.

`rnbinom` returns a vector of type [integer](#) unless generated values exceed the maximum representable integer when [double](#) values are returned.

Source

`dnbinom` computes via binomial probabilities, using code contributed by Catherine Loader (see [dbinom](#)).

`pnbinom` uses [pbeta](#).

`qnbinom` uses the Cornish–Fisher Expansion to include a skewness correction to a normal approximation, followed by a search.

`rnbinom` uses the derivation as a gamma mixture of Poisson distributions, see

Devroye, L. (1986) *Non-Uniform Random Variate Generation*. Springer-Verlag, New York. Page 480.

See Also

[Distributions](#) for standard distributions, including [dbinom](#) for the binomial, [dpois](#) for the Poisson and [dgeom](#) for the geometric distribution, which is a special case of the negative binomial.

Examples

```
rnbinom2(10, 5, sqrt(5))
```

rnull	<i>Null distributions always returns NA</i>
-------	---

Description

Null distributions always returns NA

Usage

```
rnull(n, ...)
```

Arguments

n	number of observations
...	not used

Examples

```
rnull(3, c(0.1,0.5,0.9),0.1)
```

rt_cori	<i>Reproduction number estimate using the Cori method</i>
---------	---

Description

Calculate a reproduction number estimate from incidence data using a reimplementa-tion of the Cori method and an empirical generation time distribution. This uses a mixture distribution to transmit uncertainty in generation time estimates. A number of changes compared to the original EpiEstim implementation have been made. Firstly there is no technical limitation to the infectivity profile being strictly positive in time. This allows use of serial intervals and secondary potentially delayed observations. Secondly this implementation should tolerate missing count values (NA values must be filtered out though). Thirdly for a given time point t this applies all Rt estimates for which the window spans time point t rather than end on time point t, which tends to address lag issues with the original, and fourthly this implementation allows multiple window widths to be calculated in parallel and aggregated. All of this tends to increase uncertainty in the result particularly in the time dimension, which addresses some of the issue seem with EpiEstim during the pandemic. Finally it is quite a bit quicker, especially if approximate quantiles are all that are needed.

Usage

```
rt_cori(
  df = i_incidence_input,
  ip = i_discrete_ip,
  window = 14,
  mean_prior = 1,
  std_prior = 2,
  ...,
  epiestim_compat = FALSE,
  approx = FALSE,
  .progress = interactive()
)
```

Arguments

df	The count data. Extra groups are allowed. - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
ip	A long format infectivity profile. - a dataframe with columns: - boot (anything + default(1)) - a bootstrap identifier - probability (proportion) - the probability of new event during this period. - tau (integer + complete) - the days since the index event. Minimally grouped by: boot (and other groupings allowed).
window	the widths of the Cori method window to include in the estimate. This can be a vector of values and all windows will be calculated and aggregated.
mean_prior	the prior for the $R_{t,t}$ estimate. When sample size is low the $R_{t,t}$ estimate will revert to this prior. In EpiEstim the default is a high number to allow detection of insufficient data but this tends to create anomalies in the early part of infection time series. A possible value is $R_{t,0}$ but in fact this also will be a poor choice for the value of $R_{t,t}$ when case numbers drop to a low value.
std_prior	the prior for the $R_{t,t}$ SD.
...	not used
epiestim_compat	produce an estimate of R_t using only windows that end on the time t rather than all windows that span time t. If this option is selected there can also only be one value for window.
approx	approximate the quantiles of the mixture distribution with a gamma distribution with the same first mean and SD.
.progress	show a CLI progress bar

Details

There are still issues with large R_t estimates in the early part of a time series, which is a result of the renewal equation method.

This will calculate a reproduction number for each group in the input dataframe.

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
data = example_poisson_rt_smooth()

tmp2 = data %>% rt_cori(ip=example_ip(), epiestim_compat = TRUE)
tmp3 = data %>% rt_cori(ip=example_ip(), window=c(5:14), approx=TRUE)

comp = dplyr::bind_rows(
  tmp2 %>% dplyr::mutate(class = "EpiEstim"),
  tmp3 %>% dplyr::mutate(class = "Cori+")
) %>% dplyr::group_by(class)

if (interactive()) {
  plot_rt(comp, date_labels="%b %y")+sim_geom_function(data,colour="black")+
  ggplot2::coord_cartesian(ylim=c(0.5,3.0))
}
```

rt_epiestim

EpiEstim *reproduction number wrapper function*

Description

Calculate a reproduction number estimate from incidence data using the EpiEstim library and an empirical generation time distribution. This uses resampling to transmit uncertainty in generation time estimates. This is quite slow for each time series depending on the number of bootstraps and samples in the infectivity profile.

Usage

```
rt_epiestim(
  df = i_incidence_input,
  ip = i_discrete_ip,
  bootstraps = 2000,
  window = 14,
  mean_prior = 1,
  std_prior = 2,
  ...,
  .progress = interactive()
)
```

Arguments

df	Count data. Extra groups are allowed. - a dataframe with columns: - count (positive_integer) - Positive case counts associated with the specified time frame - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
ip	infectivity profile - a dataframe with columns: - boot (anything + default(1)) - a bootstrap identifier - probability (proportion) - the probability of new event during this period. - tau (integer + complete) - the days since the index event. Minimally grouped by: boot (and other groupings allowed).
bootstraps	the number of bootstraps to take to calculate for each point.
window	the width of the EpiEstim window
mean_prior	the prior for the R_t estimate. When sample size is low the R_t estimate will revert to this prior. In EpiEstim the default is a high number to allow detection of insufficient data but this tends to create anomalies in the early part of infection time series. A possible value is R_0 but in fact this also will be a poor choice for the value of R_t when case numbers drop to a low value.
std_prior	the prior for the R_t SD.
...	not used
.progress	show a CLI progress bar

Details

This will calculate a reproduction number for each group in the input dataframe.

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period

- `rt.fit` (double) - an estimate of the reproduction number
- `rt.se.fit` (positive_double) - the standard error of the reproduction number
- `rt.0.025` (double) - lower confidence limit of the reproduction number
- `rt.0.5` (double) - median estimate of the reproduction number
- `rt.0.975` (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
data = example_poisson_rt_smooth()

tmp2 = data %>%
  rt_epiestim(ip=example_ip())

if (interactive()) {
  plot_rt(tmp2, date_labels="%b %y")+sim_geom_function(data,colour="red")
}
```

rt_from_growth_rate	<i>Wallinga-Lipsitch reproduction number from growth rates</i>
---------------------	--

Description

Calculate a reproduction number estimate from growth rate using the Wallinga and Lipsitch 2007 estimation using empirical generation time distribution. This uses resampling to transmit uncertainty in growth rate estimates. This also handles time-series that are not on a daily cadence (although this is experimental). The reproduction number estimate is neither a instantaneous (backward looking) nor case (forward looking) reproduction number but somewhere between the two, as the method looks at a flux of infection at a single point in time.

Usage

```
rt_from_growth_rate(
  df = i_growth_rate,
  ip = i_empirical_ip,
  bootstraps = 1000,
  seed = Sys.time(),
  .progress = interactive()
)
```

Arguments

df	Growth rate estimates - a dataframe with columns: • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' • growth.fit (double) - an estimate of the growth rate • growth.se.fit (positive_double) - the standard error the growth rate • growth.0.025 (double) - lower confidence limit of the growth rate • growth.0.5 (double) - median estimate of the growth rate • growth.0.975 (double) - upper confidence limit of the growth rate Any grouping allowed.
ip	Infectivity profile - a dataframe with columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
bootstraps	• the number of bootstraps to take to calculate for each point.
seed	a random number generator seed
.progress	show a CLI progress bar

Details

This method is quite slow compared to some others and the default is non deterministic.

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
data = example_poisson_rt_smooth()

tmp = data %>%
  poisson_locfit_model() %>%
  rt_from_growth_rate(ip=example_ip())

if (interactive()) {
  plot_rt(tmp, date_labels="%b %y")+sim_geom_function(data,colour="red")
}
```

rt_from_incidence	<i>Reproduction number from modelled incidence</i>
-------------------	--

Description

Calculate a reproduction number estimate from modelled incidence using the methods described in the vignette "Estimating the reproduction number from modelled incidence" and using a set of empirical generation time distributions. This assumes that modelled incidence has the same time unit as the ip distribution, and that this is daily, if this is not the case then [rescale_model\(\)](#) may be able to fix it.

Usage

```
rt_from_incidence(
  df = i_incidence_model,
  ip = i_discrete_ip,
  raw = i_incidence_data,
  approx = TRUE,
  .progress = interactive()
)
```

Arguments

df	modelled incidence estimate - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - incidence.fit (double) - an estimate of the incidence rate on a log scale - incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale - incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) - incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale) - incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale)
----	--

	Any grouping allowed.
ip	an infectivity profile (aka generation time distribution) - a dataframe with columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • tau (integer + complete) - the days since the index event. Minimally grouped by: boot (and other groupings allowed).
raw	the raw data that the modelled incidence is based on. This is optional. If not given the algorithm will assume independence, which will be faster to estimate (much faster for long time-series), but with more uncertain Rt estimates. In some circumstances this assumption of independence can cause underestimation of Rt. If this is a risk there will be a warning given, and this parameter may need to be supplied. - a dataframe with columns: • count (positive_integer) - Positive case counts associated with the specified time frame • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Any grouping allowed.
approx	use a faster, but approximate, estimate of quantiles
.progress	show a CLI progress bar

Details

N.B. for certain estimators (e.g. [poisson_gam_model\(\)](#), [poisson_locfit_model\(\)](#)) a version of this function will be called automatically if the infectivity profile is supplied. The inbuilt version is preferred over this function for these estimators, as the full covariance matrix may be used and the initial part of the outbreak can be predicted more accurately. This function is for supporting the other count models in ggoutbreak. If you are rolling your own incidence estimates and want an Rt estimate then [rt_incidence_timeseries_implementation\(\)](#) maybe better suited to your task.

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
tmp = example_poisson_rt_smooth() %>%
  poisson_locfit_model() %>%
  rt_from_incidence(
    ip = example_ip(),
    raw = example_poisson_rt_smooth(),
    approx=FALSE
  )

# This will assume independence and
tmp2 = example_poisson_rt_smooth() %>%
  poisson_locfit_model() %>%
  rt_from_incidence(ip = example_ip(), approx=TRUE)

plot_data = dplyr::bind_rows(
  tmp %>% dplyr::mutate(class = "exact"),
  tmp2 %>% dplyr::mutate(class = "approx"),
) %>% dplyr::group_by(class)

if (interactive()) {
  plot_rt(plot_data, date_labels="%b %y")+
    sim_geom_function(example_poisson_rt_smooth())+
    ggplot2::coord_cartesian(ylim=c(0.5,3))+
    ggplot2::facet_wrap(~class)
}
```

rt_from_renewal	<i>Reproduction number from renewal equation applied to modelled incidence using statistical re-sampling</i>
-----------------	--

Description

Calculate a reproduction number estimate from modelled incidence estimates, by statistical sampling from a log-normally distributed incidence estimate, such as you get from a $I_t \sim \text{Poisson}(I_0 e^{\{rt\}})$ model using a log link function. This is combined with uncertain infectivity profile specified as multiple discrete empirical distributions, to calculate the range of possible values for R_t .

Usage

```
rt_from_renewal(
  df = i_incidence_model,
  ip = i_discrete_ip,
  bootstraps = 1000,
  seed = Sys.time(),
  .progress = interactive()
)
```

Arguments

df	modelled incidence estimate - a dataframe with columns: - time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' - incidence.fit (double) - an estimate of the incidence rate on a log scale - incidence.se.fit (positive_double) - the standard error of the incidence rate estimate on a log scale - incidence.0.025 (positive_double) - lower confidence limit of the incidence rate (true scale) - incidence.0.5 (positive_double) - median estimate of the incidence rate (true scale) - incidence.0.975 (positive_double) - upper confidence limit of the incidence rate (true scale) Any grouping allowed.
ip	infectivity profile - a dataframe with columns: - boot (anything + default(1)) - a bootstrap identifier - probability (proportion) - the probability of new event during this period. - tau (integer + complete) - the days since the index event. Minimally grouped by: boot (and other groupings allowed).
bootstraps	the number of samples to take at each time point. This will be rounded up to a whole multiple of the infectivity profile distribution length.
seed	a random number seed for reproducibility
.progress	show a CLI progress bar

Details

This method is moderately slow and non deterministic by default.

Value

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```
data = example_poisson_rt_smooth()

tmp2 = data %>%
  poisson_locfit_model() %>%
  rt_from_renewal(ip=example_ip())

if (interactive()) {
  plot_rt(tmp2, date_labels="%b %y")+sim_geom_function(data,colour="red")
}
```

```
rt_incidence_reference_implementation
```

Reference implementation of the Rt from modelled incidence algorithm

Description

This function estimates the reproduction number for a specific point in time given a time series of log-normally distributed incidence estimates, a set of infectivity profiles. This version of the algorithm only works for a single time point and is not optimised for running over a whole time series. For that please see [rt_incidence_timeseries_implementation\(\)](#).

Usage

```
rt_incidence_reference_implementation(
  mu_t,
  vcov_ij = diag(sigma_t^2),
  omega,
  sigma_t = NULL,
  tau_offset = 0
)
```

Arguments

<code>mu_t</code>	a vector of meanlog parameters of a lognormal distribution of modelled incidence. This is a vector of length k .
<code>vcov_ij</code>	a log scale variance-covariance matrix of predictions. It is optional but if not given and <code>sigma_t</code> is given then this will be inferred assuming independence between estimates. This should be a matrix with dimensions $k \times k$
<code>omega</code>	a matrix (or vector) representing the infectivity profile as a discrete time probability distribution. This must be a $k \times n$ matrix or a vector of length k . Each of the n columns is a individual estimate of the infectivity profile (N.B. this is the same format as <code>EpiEstim</code>). In <code>EpiEstim</code> the first row of this matrix must be zero and represents a delay of zero. This constraint does not apply here.
<code>sigma_t</code>	if <code>vcov_ij</code> is not given then this must be supplied as the sdlog of the log normal incidence estimate

`tau_offset` In most cases the infectivity profile has support for delays from $0:(k-1)$ and the R_t estimate is made at time $(k-1)$. If there is a negative component of a serial interval being used as a proxy then the support will be $-\text{tau_offset}:(k-\text{tau_offset}-1)$.

Details

N.B. the description of this algorithm is given here: <https://ai4ci.github.io/ggoutbreak/articles/rt-from-incidence.html>

Value

a list with the following items:

- `time_Rt`: the time point of the R_t estimate (usually $k-1$)
- `mean_Rt_star`: the mean of the R_t estimate
- `var_Rt_star`: the variance of the R_t estimate
- `meanlog_Rt_star`: log normal parameter for approximate distribution of R_t
- `sdlog_Rt_star`: log normal parameter for approximate distribution of R_t
- `mu_Rt_mix`: a vector of log normal parameters for more exact mixture distribution of R_t
- `sigma_Rt_mix`: a vector of log normal parameters for more exact mixture distribution of R_t
- `quantile_Rt_fn`: A log-normal mixture quantile function for R_t

Quantiles can either be obtained from the quantile function or from e.g. `qlnorm(p, meanlog_Rt_star, sdlog_Rt_star)`

Examples

```
data = example_poisson_rt_smooth()

omega = omega_matrix(example_ip())
k = nrow(omega)

pred_time = 50
index = pred_time-k:1+1

# we will try and estimate the Rt at time k+10:
print(data$rt[pred_time])

# first we need a set of incidence estimates
# in the simplest example we use a GLM:

newdata = dplyr::tibble(time = 1:100)

model = stats::glm(count ~ splines::bs(time,df=8), family = "poisson", data=data)
pred = stats::predict(model, newdata, se.fit = TRUE)

# I've picked a df to make this example work. In real life you would need
# to validate the incidence model is sensible and not over fitting
```

```

# before using it to estimate RT:
# ggplot2::ggplot()+
#   ggplot2::geom_point(data=data, ggplot2::aes(x=time, y=count))+
#   ggplot2::geom_line(
#     data = newdata %>% dplyr::mutate(fit = exp(pred$fit)
#     ), ggplot2::aes(x=time,y=fit))

mu_t = pred$fit[index]
sigma_t = pred$se.fit[index]

# prediction vcov is not simple from GLM models.
rt_est = rt_incidence_reference_implementation(mu_t=mu_t,sigma_t = sigma_t, omega = omega)

# quantiles from a mixture distribution:
rt_est$quantile_Rt_fn(c(0.025,0.5,0.975))
# and from the rough estimate based on matching of moments:
stats::qlnorm(c(0.025,0.5,0.975), rt_est$meanlog_Rt_star, rt_est$sdlog_Rt_star)

# GLM do not produce vcov matrices we can estimate them if we have access to
# the data:
vcov_glm = vcov_from_residuals(data$count[1:100], pred$fit, pred$se.fit)$vcov_matrix
vcov_glm_ij = vcov_glm[index, index]

# Using an estimated vcov we get very similar answers:
rt_est_vcov = rt_incidence_reference_implementation(mu_t=mu_t,vcov_ij = vcov_glm_ij, omega = omega)
rt_est_vcov$quantile_Rt_fn(c(0.025,0.5,0.975))
# In theory assuming independence leads to excess uncertainty and possible
# underestimation bias. In most situations though this appears low risk.

# Lets do the same with a GAM:
model2 = mgcv::gam(count ~ s(time), family = "poisson", data=data)
pred2 = stats::predict(model2, newdata, se.fit = TRUE)

# we can get prediction level vcov from GAMs easily
Xp = stats::predict(model2, newdata, type = "lpmatrix")
pred_vcov = Xp %*% stats::vcov(model2) %*% t(Xp)
vcov_ij = pred_vcov[index, index]
mu_t2 = pred2$fit[index]

rt_est2 = rt_incidence_reference_implementation(mu_t=mu_t2, vcov_ij=vcov_ij, omega = omega)
rt_est2$quantile_Rt_fn(c(0.025,0.5,0.975))

# How does this compare to EpiEstim:
# N.B. setting seed to make deterministic
withr::with_seed(100, {
  epi = EpiEstim::estimate_R(
    data$count,
    method = "si_from_sample",
    si_sample = omega,
    config = EpiEstim::make_config(
      method = "si_from_sample",
      t_start = pred_time-7,
      t_end = pred_time,

```

```

      n2 = 100)
    )
  })

  epi$R %>%
    dplyr::select(`Quantile.0.025(R)`, `Median(R)`, `Quantile.0.975(R)`)

```

```
rt_incidence_timeseries_implementation
```

Time series implementation of the Rt from modelled incidence algorithm

Description

This function estimates the reproduction number for a while time series given log-normally distributed incidence estimates, and a set of infectivity profiles. This version of the algorithm is optimised for running over all of a single time series. The algorithm will not produce Rt estimates until there is at least the

Usage

```

rt_incidence_timeseries_implementation(
  time,
  mu,
  vcov = NULL,
  sigma = NULL,
  ip = i_discrete_ip,
  tidy = FALSE,
  ...
)

```

Arguments

time	a set of time points as a numeric vector. This is a vector of length k.
mu	a time series of meanlog parameters of a lognormal distribution of modelled incidence. This is a vector of length k.
vcov	a log scale variance-covariance matrix of predictions. It is optional but if not given and sigma_t is given then this will be inferred assuming independence between estimates. This should be a matrix with dimensions k * k
sigma	if vcov is not given then this must be supplied as the sdlog of the log normal incidence estimate. If this is the case the estimates will be made assuming estimate independence. A vector of length k. Checks will be made that determine if there is risk of bias, and this can make this slower than using the full covariance matrix.
ip	a long format infectivity profile dataframe. - a dataframe with columns:

	• boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • tau (integer + complete) - the days since the index event.
	Minimally grouped by: boot (and other groupings allowed).
tidy	do you want detailed raw output (FALSE - default) or summary output with quantiles predicted.
...	passed onto output formatter if tidy=TRUE, at the moment only approx = FALSE

Details

N.B. the description of this algorithm is given here: <https://ai4ci.github.io/ggoutbreak/articles/rt-from-incidence.html>

Value

a dataframe with k rows with the following columns:

- time_Rt: the time point of the Rt estimate (usually k-1)
- mean_Rt_star: the mean of the Rt estimate
- var_Rt_star: the variance of the Rt estimate
- meanlog_Rt_star: log normal parameter for approximate distribution of Rt
- sdlog_Rt_star: log normal parameter for approximate distribution of Rt
- mu_Rt_mix: a list of vectors of log normal parameters for more exact mixture distribution of Rt
- sigma_Rt_mix: a list of vectors of log normal parameters for more exact mixture distribution of Rt

Approximate quantiles can be obtained from e.g. `qlnorm(0.5, meanlog_Rt_star, sdlog_Rt_star)`

Alternatively if tidy is true the output will be post processed to conform to:

A dataframe containing the following columns:

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period
- rt.fit (double) - an estimate of the reproduction number
- rt.se.fit (positive_double) - the standard error of the reproduction number
- rt.0.025 (double) - lower confidence limit of the reproduction number
- rt.0.5 (double) - median estimate of the reproduction number
- rt.0.975 (double) - upper confidence limit of the reproduction number

Any grouping allowed.

Examples

```

data = example_poisson_rt_smooth()
ip = example_ip()

# first we need a set of incidence estimates
# we fit a poisson model to counts using a GAM:
model = mgcv::gam(count ~ s(time), family = "poisson", data=data)
ip_len = max(ip$tau)
newdata = dplyr::tibble(time = -ip_len:100)

pred = stats::predict(model, newdata, se.fit = TRUE)

# we can get prediction vcov from GAMs fairly easily
Xp = stats::predict(model, newdata, type = "lpmatrix")
pred_vcov = Xp %*% stats::vcov(model) %*% t(Xp)

# Now we estimate rt
rt_est = rt_incidence_timeseries_implementation(
  time = newdata$time,
  mu = pred$fit,
  vcov = pred_vcov,
  ip = ip)

# Lets compare epiestim on the same data
withr::with_seed(100, {
  epi = EpiEstim::estimate_R(
    data$count,
    method = "si_from_sample",
    si_sample = omega_matrix(ip),
    config = EpiEstim::make_config(
      method = "si_from_sample",
      t_start = 10:100-7,
      t_end = 10:100,
      n2 = 100)
  )
})

ggplot2::ggplot()+
  ggplot2::geom_line(
    data=data %>% dplyr::filter(time<100),
    ggplot2::aes(x=time,y=rt), colour="black")+
  ggplot2::geom_line(
    data = rt_est, ggplot2::aes(x=time, y=mean_Rt_star, colour="gam+rt"))+
  ggplot2::geom_line(
    data = epi$R, ggplot2::aes(x=t_end, y=~Mean(R)~, colour="epiestim"))

# mean bias of GAM+rt estimate:
mean(data$rt[seq_along(rt_est$mean_Rt_star)] - rt_est$mean_Rt_star)

```

rwedge	<i>Wedge distribution</i>
--------	---------------------------

Description

The wedge distribution has a support of 0 to 1 and has a linear probability density function over that support.

Usage

rwedge(n, a)

Arguments

- n number of observations
- a a gradient from -2 (left skewed) to 2 (right skewed)

Details

The rwedge can be combined with quantile functions to skew standard distributions, or introduce correlation or down weight certain parts of the distribution.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
pwedge(seq(0,1,0.1), a=1)
dwedge(seq(0,1,0.1), a=1)
qwedge(c(0.25,0.5,0.75), a=-1)

stats::cor(
  stats::qnorm(rwedge(1000, a=2)),
  stats::qnorm(rwedge(1000, a=-2))
)
```

scale_x_log1p	<i>A log1p x scale</i>
---------------	------------------------

Description

A log1p x scale

Usage

scale_x_log1p(..., n = 5, base = 10, dp = 0)

Arguments

...	Other arguments passed on to scale_(x y)_continuous()
n	the number of major breaks
base	the base for the logarithm
dp	decimal points

Value

a ggplot scale

scale_x_logit	<i>A logit x scale</i>
---------------	------------------------

Description

A logit x scale

Usage

scale_x_logit(...)

Arguments

...	Other arguments passed on to scale_(x y)_continuous()
-----	---

Value

a ggplot scale

scale_y_log1p	<i>A log1p y scale</i>
---------------	------------------------

Description

A log1p y scale

Usage

scale_y_log1p(..., n = 5, base = 10, dp = 0)

Arguments

...	Other arguments passed on to scale_(x y)_continuous()
n	the number of major breaks
base	the base for the logarithm
dp	decimal points

Value

a ggplot scale

scale_y_logit	<i>A logit y scale</i>
---------------	------------------------

Description

A logit y scale

Usage

```
scale_y_logit(...)
```

Arguments

... Other arguments passed on to `scale_(x|y)_continuous()`

Value

a ggplot scale

score_estimate	<i>Calculate scoring statistics from predictions.</i>
----------------	---

Description

This performs a range of continuous scoring metrics for each estimate time-point using cumulative distribution functions for each estimate. Point quality metrics are calculated for each estimate provided and summarised. Summarisation is performed using bootstrap resampling to generate confidence intervals for summary statistics, which are presented as median +/- 95% CI.

Usage

```
score_estimate(  
  est,  
  obs,  
  lags = NULL,  
  summarise_by = est %>% dplyr::groups(),  
  bootstraps = 1000,  
  raw_bootstraps = FALSE,  
  seed = 100  
)
```

Arguments

<code>est</code>	a dataframe of estimates of incidence, growth rate of reproduction number based off a simulation or data with known parameters. Each group in <code>est</code> is expected to contain multiple estimates and each group is scored separately. Estimates in <code>est</code> must be in the form of a column named <code>XXX.cdf</code> containing a cumulative distribution function for the estimate and <code>XXX.link</code> containing a link function specification (one if identity, log or logit). These are not generated by default by the <code>ggoutbreak</code> estimators but are triggered by setting the option: <code>options("ggoutbreak.keep_cdf"=TRUE)</code> before running the estimator. The CDFs generated will be analytical, if the estimator generates a parametrised output (or a mixture thereof), empirical if the estimator uses resampling, or inferred if the estimator produces quantiles only.
<code>obs</code>	a dataframe of the ground truth, sharing the same grouping and columns as <code>est</code> with at least one column(s) named <code>XXX.obs</code> with <code>XXX</code> being e.g. <code>rt.growth</code> or <code>incidence</code> or any other column group predicted in <code>est</code> (i.e. if <code>obs</code> has a column <code>XXX.obs</code> , <code>est</code> must have one called <code>XXX.cdf</code>).
<code>lags</code>	a data frame of estimate types and lags as output by <code>quantify_lag()</code> if multiple models are included then the columns must match those in <code>obs</code> . It must have 2 columns, one called <code>estimate</code> with values matching <code>incidence</code> , <code>rt.growth</code> , <code>proportion</code> , <code>relative.growth</code> and a <code>lag</code> column, with (whole) number of days.
<code>summarise_by</code>	by default every group is treated separately. This can be overridden with a <code>dplyr</code> specification of the groupings we want to see in the final summarised output (e.g. if we want to differentiate performance on a particular type of scenario or timeframe). If this is exactly <code>FALSE</code> the function will return all the raw point estimates.
<code>bootstraps</code>	the number of bootstrap replicates to draw for assessing metric confidence. If <code>FALSE</code> then no bootstrapping will be done and the metrics returned will have no confidence intervals.
<code>raw_bootstraps</code>	(defaults to <code>FALSE</code>) return the summary metrics for each bootstrap rather than the quantiles of the summary metrics.
<code>seed</code>	a random seed for reproducibility

Value

a dataframe of scoring metrics, with one row per group. This includes the following columns:

- `mean_quantile_bias` - the average of the universal residuals. Lower values are better.
- `mean_trans_bias` - the bias on the link function scale.
- `link` - the link function
- `mean_bias` - the bias on the natural scale (which may be interpreted as additive or multiplicative depending on the link)
- `pit_was` - an unadjusted probability integral transform histogram Wasserstein distance from the uniform (lower values are better).
- `unbiased_pit_was` - an PIT Wasserstein distance from the uniform, adjusted for estimator bias (lower values are better). This is a measure of calibration.

- `directed_pit_was` - a PIT Wasserstein distance from the uniform, directed away from the centre, adjusted for estimator bias (values closer to zero are better, positive values indicate overconfidence, and negative values excessively conservative estimates).
- `percent_iqr_coverage` - the percentage of estimators that include the true value in their IQR. For a perfectly calibrated estimate this should be 0.5. Lower values reflect overconfidence, higher values reflect excessively conservative estimates. This is a measure of calibration but is influenced by bias.
- `unbiased_percent_iqr_coverage` - the percentage of estimators that include the true value in their IQR once adjusted for bias. This should be 0.5. This is a measure of calibration, and tells you which direction (smaller numbers are over-confident, larger values excessively conservative).
- `mean_prediction_interval_width_50` - the prediction interval width is a measure of sharpness (smaller values are sharper). Sharper estimators are superior if they are unbiased and well calibrated.
- `mean_crps` - the mean value of the continuous rank probability score for each point estimate (lower values are better)
- `mean_unbiased_crps` - the mean value of the continuous rank probability score for each point estimate assessed after adjustment for bias (lower values are better)
- `threshold_misclassification_probability` - if a metric has a natural threshold like 1 for R_t then this measures how probable it is that the estimate will propose the epidemic is shrinking when it is growing and vice versa. Lower is better

other outputs are possible if `summarise_by` is false.

Examples

```
data = example_poisson_rt_smooth()

pipeline = ~ .x %>% poisson_locfit_model(ip = .y, quick=TRUE)
lags = quantify_lag(pipeline, ip = example_ip())

withr::with_options(list("ggoutbreak.keep_cdf"=TRUE),{
  est = data %>% poisson_locfit_model(ip = example_ip(), quick=TRUE)
})

if (interactive()) plot_rt(est)+sim_geom_function(data, colour="red")

obs = data %>% dplyr::mutate(rt.obs = rt, incidence.obs = rate)
score_estimate(est,obs,lags) %>% dplyr::glimpse()
```

Description

This function is generally not needed, and will be called automatically when the first date conversion is performed. If no other information is given then the default origin will be decided by the start of the first use of the `time_period` class in a session. This helps to keep defaults consistent during a single run, if they are not specified.

Usage

```
set_defaults(start_date, unit)

with_defaults(start_date, unit, expr)

set_default_start(date)

set_default_unit(unit)
```

Arguments

<code>start_date</code>	the zero time date as something that can be coerced to a date. If the <code>x</code> input is already a <code>time_period</code> and this is different to its <code>start_date</code> then <code>x</code> will be recalibrated to use the new start date.
<code>unit</code>	the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another <code>time_period</code> . If <code>x</code> is a <code>time_period</code> , and the unit is different to that of <code>x</code> this will return a rescaled <code>time_period</code> using the new units.
<code>expr</code>	an expression to evaluate with the defaults set to the provided values.
<code>date</code>	A date, or something that can be cast to one, that represents day zero for an outbreak.

Value

depending on the methods the original default start date / the original default unit, a list of both or the result of evaluating the expression `expr`

Functions

- `with_defaults()`: Set defaults temporarily and execute expression
- `set_default_unit()`: Set the default unit only

Examples

```
# set default origin and cadence:
old = set_defaults("2025-01-01", "1 week")

# this sets the default for interpreting underqualified time_periods:
print(as.time_period(1:10))

# The default can always be overridden on a case by case basis:
```

```

print(as.time_period(1:10, unit="1 day"))

# or for a whole expression:
with_defaults("2020-01-01", "1 day", {
  print(as.time_period(1:10))
})

# components can be changed individually, firstly origin:
set_default_start("2025-01-01")
print(as.time_period(1:10))

# now cadence:
set_default_unit("1 day")
print(as.time_period(1:10))

# clear the values:
set_defaults(NULL,NULL)

# A sufficiently qualified call will set the defaults:
defined = as.time_period(as.Date("2024-01-01")+0:10*7, anchor="start")
inherit = as.time_period(0:10)

all(defined == inherit)

# restoring the original values (which might be null)
set_defaults(old)

```

sim_apply_ascertainment

Apply a ascertainment bias to the observed case counts.

Description

Apply a ascertainment bias to the observed case counts.

Usage

```
sim_apply_ascertainment(df = i_sim_count_data, fn_asc = ~1, seed = Sys.time())
```

Arguments

- | | |
|----|--|
| df | a count dataframe from e.g. <code>sim_poisson_model()</code> or <code>sim_summarise_linelist()</code>
- a dataframe with columns: <ul style="list-style-type: none"> • statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths • count (positive_integer) - Positive case counts associated with the specified time frame |
|----|--|

- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period’
- Minimally grouped by: statistic (and other groupings allowed).
- fn_asc a function that takes a single input vector t and returns a probability of ascertainment, e.g. ~ stats::rbeta(.x, 20, 80) or ~ rbeta2(.x,prob=<probability>,kappa=<dispersion> or [cfg_weekly_proportion_rng\(\)](#)
- seed a RNG seed

Value

a dataframe with original column, and count column modified to include ascertainment bias.

Examples

```
with_defaults("2025-01-01" ,"1 day", {
  dplyr::tibble(
    statistic = "incidence",
    time=as.time_period(1:10,"1 day"),
    count=rep(100,10)
  ) %>%
  dplyr::group_by(statistic) %>%
  sim_apply_ascertainment(~ ifelse(.x<=5,0.1,0.9))
})
```

sim_apply_delay	<i>Apply delay distribution to count or linelist data</i>
-----------------	---

Description

Events include symptom onset, admission, death, test sampling, test processing

Usage

```
sim_apply_delay(
  df,
  ...,
  fn_p_symptomatic = ~0.5,
  fn_p_admitted = ~0.1,
  fn_p_died = ~0.05,
  fn_p_tested = ~0.8,
  seed = Sys.time()
)
```

Arguments

df	a line list dataframe arising from e.g. <code>sim_branching_process()</code> - EITHER: a dataframe with columns: • id (unique_id) - Patient level unique id • time (ggoutbreak::time_period) - Time of infection. A 'time_period' Any grouping allowed. OR with columns: • statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths • count (positive_integer) - Positive case counts associated with the specified time frame • time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period' Minimally grouped by: statistic (and other groupings allowed).
...	Named arguments passed on to <code>sim_apply_delay.linelist</code> <code>fn_symptom_delay</code>, <code>fn_admission_delay</code>, <code>fn_death_delay</code> a function that calculates the time to event onset from infection. This will be called with a vector of infection times as the first parameter (time) but all other columns of df are also available as well as the symptomatic, died, and admitted flags. The function must be vectorised on its inputs (and consume additional inputs with ...). A purrr style lambda is OK e.g. <code>~ stats::rgamma(.x, shape = 3)</code>, and the first parameter will be infection time. if you have a discrete probability profile for this then you can use <code>cfg_ip_sampler_rng(ip_symptoms)</code>. <code>fn_sample_delay</code> This function returns the time from either symptom onset (symptomatic) or from infection (asymptomatic) until a sample is taken. (N.B. this might be better to do a screening test probability plus screening test frequency rather than overloading this.) <code>fn_result_delay</code> Identical to other functions except the first parameter will be sample_time rather than time of infection. This is the time from sampling to the result being available. Named arguments passed on to <code>sim_apply_delay.count_data</code> <code>fn_symptom_profile</code>, <code>fn_admission_profile</code>, <code>fn_death_profile</code> a function that takes time and returns the probability density of symptoms, admissions, or deaths over time since infection (i.e. tau) as an ip delay distribution. If possible it is a very good idea to pre-compute these distributions as they need to be assigned to every line in the input and this can be very slow. <code>fn_sample_profile</code> a function that takes time and returns the probability density of test sample being taken over time since symptoms. <code>fn_result_profile</code> a function that takes time and returns the probability density of test result being available over time since test sampling. <code>fn_p_symptomatic</code>, <code>fn_p_admitted</code>, <code>fn_p_died</code>, <code>fn_p_tested</code> Function that returns a probability between 0 and 1 for each row of the input dataframe. A purrr style lambda is OK (e.g. <code>~ 1</code> for always true) the first parameter of this will be time of infection. The function must be vectorised on its inputs (and consume additional inputs with ...)
seed	RNG seed for reproducibility

Value

Depends on input, either:

- a wide format line list with additional XX, XX_time and XX_delay columns, for each of the set of statistics generated.
- a long format set of counts of different statistics i.e. infections, symptoms, admission, death, sample (tests taken), results (test results) .

Examples

```
tmp = sim_branching_process(
  changes = dplyr::tibble(t = c(0,20,40,60,80,110), R = c(1.8,1.5,0.9,1.5,0.8,1.2)),
  max_time = 120,
  seed = 100
)

tmp2 = tmp %>% sim_apply_delay()
tmp2 %>% dplyr::glimpse()
```

sim_branching_process *Generate a line list from a branching process model parametrised by reproduction number*

Description

Generate a line list from a branching process model parametrised by reproduction number

Usage

```
sim_branching_process(
  changes = dplyr::tibble(t = c(0, 40), rt = c(2, 0.8)),
  max_time = 80,
  seed = Sys.time(),
  fn_Rt = cfg_step_fn(changes),
  fn_ip = ~example_ip(),
  fn_kappa = ~1,
  imports_df = NULL,
  fn_imports = ~ifelse(.x == 0, 30, 0),
  fn_list_next_gen = list(),
  max_size = 10000,
  ...
)
```

Arguments

changes	a dataframe containing a <code>t</code> time column and <code>R</code> reproduction number parameter. This parameter is optional if <code>fn_Rt</code> is specified
max_time	maximum duration of simulation
seed	random seed
fn_Rt	can be specified instead of changes df. This is a vectorised function that accepts a time parameter and returns a reproduction number. If both this and changes are specified this takes preference.
fn_ip	a function that takes input vector <code>t</code> (and/or <code>class</code>) and returns an infectivity profile at times <code>t</code> .
fn_kappa	a vectorised function taking <code>t</code> and other imported case metadata returning a dispersion parameter controlling the likelihood of individual super-spreading. This must be between 1 and <code>Inf</code> with 1 being standard poisson dispersion and larger values representing over dispersion.
imports_df	a data frame containing minimally time and count columns plus any meta-data about the imports in additional columns. Metadata columns can inform the <code>fn_Rt</code> , <code>fn_kappa</code> and <code>fn_ip</code> functions as additional parameters.
fn_imports	a time varying function the defines the number of infected importations. If <code>imports_df</code> is defined then this is used instead
fn_list_next_gen	a named list of functions. The name corresponds to metadata columns in the simulation, the function is a purrr style mapping that will replace the old value in the named column with a new one. Such a function can be generated with <code>cfg_transition_fn()</code> when a transition probability matrix is involved, or it can be specified directly as a <code>case_when</code> style function. The function must be vectorised and assume no grouping structure. If the function has named parameters it can reference any of the metadata columns, or time (as <code>t</code>). The <code>rcategorical()</code> function may be useful in this scenario.
max_size	the maximum size of a single generation. If a generation exceeds this limit the branching process terminates with a warning that the simulation is incomplete.
...	not used

Value

a line list of cases for a simulated outbreak

A dataframe containing the following columns:

- `id` (`unique_id`) - Patient level unique id
- `time` (`ggoutbreak::time_period`) - Time of infection. A 'time_period'

Any grouping possible.

Examples

```
tmp = sim_branching_process(
  changes = dplyr::tibble(t = c(0,40), R = c(1.5,0.8)),
  max_time = 120,
  seed = 100,
  fn_imports = ~ ifelse(.x<10,1,0)
)

if(interactive()) {
  plot_cases(tmp, mapping=ggplot2::aes(fill=as.factor(generation)),linewidth=0.1, colour="white")
}

# imports can also be specified as a dataframe, which allows additional
# metadata in the line list. An example of which is as follows:
imports_df = dplyr::tribble(
  ~time, ~variant, ~count,
  0:4, "wild-type", 100,
  10:14, "alpha", 5,
)
```

sim_convolution

Apply a time varying probability and convolution to count data

Description

Standard convolution assumes one delay distribution. This is actually not what we see in reality as delays can depend on any factors, including the day of week. This function applies a convolution to an input time-series when that convolution is expressed as a function (usually of time, but can be of anything in the input dataframe). The convolution is then sampled using a poisson or negative binomial

Usage

```
sim_convolution(
  df = i_sim_count_data,
  p_fn,
  delay_fn,
  ...,
  input = "infections",
  output,
  kappa = 1,
  from = c("count", "rate")
)
```

Arguments

df a count dataframe from e.g. [sim_poisson_model\(\)](#) or [sim_summarise_linelist\(\)](#)
 - a dataframe with columns:

- statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

Minimally grouped by: statistic (and other groupings allowed).

p_fn	a function that takes a time parameter and potentially and returns probability of observation given something occurs. a no-op for this parameter is ~ 1.
delay_fn	a function that takes time and returns the probability of observation (given it occurred) over time since infection (i.e. tau) as an ip delay distribution. This does not have to sum to 1 (e.g. mapping incidence to prevalence) but if not then the combination of p_fn and delay_fn is less easy to interpret. This should behave sensibly if p changes halfway through a convolution. See cfg_weekly_ip_fn() and cfg_gamma_ip_fn() for helper functions to construct this parameter. A no-op for this parameter would be ~ ifelse(.x==0, 1, 0).
...	not used
input	the input statistic
output	the output statistic
kappa	dispersion. scaled such that poisson dispersion is 1. Values must be 0 (no dispersion), 1 (poisson dispersion) or greater than 1 for over-dispersion.
from	Controls if you base future counts on previous counts or on underlying rate, defaults to count but rate is a possibility if you want to base the convolution off a more theoretical value rather than observed cases. Either way the convolution generates a new rate which is in turn sampled into a poisson or negative binomial count.

Value

return the result of applying this convolution to the data.

Examples

```
weekday_delay = make_fixed_ip(mean = 5, sd = 2)
weekend_delay = make_fixed_ip(mean = 6, sd = 2)

delay_fn = ~ ifelse(.x %% 7 %in% c(6,7), list(weekend_delay), list(weekday_delay))
p_fn = ~ ifelse(.x < 20, 0.5, 0.75)

data = dplyr::tibble(
  time=1:40,
  count = rep(100,40),
  rate = rep(100,40),
  statistic="infections") %>% dplyr::group_by(statistic)
delayed = data %>%
  sim_convolution(p_fn,delay_fn,output="delayed") %>%
  dplyr::filter(statistic=="delayed")
```

```

if (interactive()) ggplot2::ggplot(delayed,ggplot2::aes(x=time))+
  ggplot2::geom_line(ggplot2::aes(y=rate))+
  ggplot2::geom_line(ggplot2::aes(y=count))

# other example delay functions
delay_fn = cfg_gamma_ip_fn( ~ ifelse(.x<5, 8, 4))

```

sim_delay

*Apply a time-varying probability and delay function to linelist data***Description**

Apply a time-varying probability and delay function to linelist data

Usage

```

sim_delay(
  df = i_sim_linelist,
  p_fn,
  delay_fn,
  input = "time",
  output = "event",
  seed = Sys.time()
)

```

Arguments

df	a line list dataframe arising from e.g. <code>sim_branching_process()</code> - a dataframe with columns: • id (unique_id) - Patient level unique id • time (ggoutbreak::time_period) - Time of infection. A ‘time_period’ Any grouping allowed.
p_fn	Function that returns a probability between 0 and 1 for each row of the input dataframe. A purrr style lambda is OK (e.g. <code>~ 1</code> for always true) the first parameter of this will be time of infection. The function must be vectorised on its inputs (and consume additional inputs with <code>...</code>)
delay_fn	A function that calculates the time to event onset from the input time. This will be called with a vector of infection times as the first parameter (time) but all other columns of df are also available as well as the symptomatic,died,and admitted flags. The function must be vectorised on its inputs (and consume additional inputs with <code>...</code>). A purrr style lambda is OK e.g. <code>~ stats::rgamma(.x, shape = 3)</code>, and the first parameter will be infection time. if you have an discrete probability profile for this then you can use <code>cfg_ip_sampler_rng(ip_symptoms)</code> without the tilde.
input	a time column from which to calculate the delay from.
output	an output column set name (defaults to "event")
seed	RNG seed for reproducibility

Value

the line list with extra columns with prefix given by output, specifying whether the event was observed, the delay and the simulation time.

Examples

```
tmp = sim_branching_process(
  changes = dplyr::tibble(t = c(0,20,40,60,80,110), R = c(1.8,1.5,0.9,1.5,0.8,1.2)),
  max_time = 120,
  seed = 100
)

tmp2 = tmp %>% sim_delay(
  p_fn = ~ rbern(.x, 0.8),
  delay_fn = ~ rgamma2(.x, mean = 5),
)
tmp2 %>% dplyr::glimpse()
```

sim_delayed_observation

Apply a right censoring to count data.

Description

Delayed observations means that if, for example a case is only attributed to a disease after a delay there is right censoring of the data. There can be very complex patterns of right censoring if for example observations are batched and published weekly. During COVID in the UK some death data was published frequently but some was retrospectively reported on monthly intervals, depending on where the patient died, which lead to complex time dependent biases in the death data. Given the description of the delay, this function will simulate this effect for count data. In another example there were delays reporting test results in the run up to Christmas which resulted in case rates apparently dropping as schools broke up. This could have affected timing of the 2021 Christmas lockdown.

Usage

```
sim_delayed_observation(
  df = i_sim_count_data,
  delay_fn,
  ...,
  input = "infections",
  output = input,
  max_time = max(df$time)
)
```

Arguments

df	a count dataframe from e.g. <code>sim_poisson_model()</code> or <code>sim_summarise_linelist()</code>
delay_fn	a function that takes time and returns the probability of observation (given it occurred) over time since infection (i.e. τ) as an ip delay distribution. This does not have to sum to 1 (e.g. mapping incidence to prevalence) but if not then it will behave as if some fraction of events are not observed (or observed multiple times). See <code>cfg_weekly_ip_fn()</code> and <code>cfg_gamma_ip_fn()</code> for helper functions to construct this parameter.
...	not used
input	the input statistic (defaults to count)
output	the output column name (defaults to same as input)
max_time	the date on which censoring is taking place.

Value

the result of applying this right censoring to the data.

Examples

```
weekday_delay = make_fixed_ip(mean = 5, sd = 2)
weekend_delay = make_fixed_ip(mean = 7, sd = 2)

delay_fn = ~ ifelse(.x %% 7 %in% c(6,7), list(weekend_delay), list(weekday_delay))

data = dplyr::tibble(time=1:40, count = rep(100,40), statistic="infections") %>%
  dplyr::group_by(statistic) %>%
  sim_delayed_observation(delay_fn,output="delayed")

if (interactive()) ggplot2::ggplot(data,ggplot2::aes(x=time,colour=statistic))+
  ggplot2::geom_line(ggplot2::aes(y=count))
```

sim_events

Extract the events dataframe from a simulation output

Description

All the simulations should include details of major changes in the simulation input parameters, particularly for step functions. This set of events can be directly represented on a ggoutbreak plot using the events parameter

Usage

```
sim_events(df)
```

Arguments

df	the output of a ggoutbreak simulation
----	---------------------------------------

Value

an events dataframe

Examples

```
sim_events(example_poisson_rt())
```

sim_geom_function	<i>The principal input function to a ggoutbreak simulation as a ggplot2 layer.</i>
-------------------	--

Description

simulations are typically parameterised by time varying R_t or by growth rate, and this relationship is embedded in the simulation outputs. Plotting the value of these on the default ggoutbreak plots requires extracting this function and rescaling it to align with the dates, which is what this function does.

Usage

```
sim_geom_function(df, ...)
```

Arguments

df	the output of a ggoutbreak simulation, typically this is going to be the input of an estimator.
...	Named arguments passed on to <code>ggplot2::geom_function</code>

mapping Set of aesthetic mappings created by `aes()`. If specified and `inherit.aes = TRUE` (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.

data Ignored by `stat_function()`, do not use.

stat The statistical transformation to use on the data for this layer. When using a `geom_*()` function to construct a layer, the `stat` argument can be used to override the default coupling between geoms and stats. The `stat` argument accepts the following:

- A Stat ggproto subclass, for example `StatCount`.
- A string naming the stat. To give the stat as a string, strip the function name of the `stat_` prefix. For example, to use `stat_count()`, give the stat as "count".
- For more information and other ways to specify the stat, see the [layer stat](#) documentation.

position A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The `position` argument accepts the following:

- The result of calling a position function, such as `position_jitter()`. This method allows for passing extra arguments to the position.
 - A string naming the position adjustment. To give the position as a string, strip the function name of the `position_` prefix. For example, to use `position_jitter()`, give the position as "jitter".
 - For more information and other ways to specify the position, see the [layer position](#) documentation.
- ... Other arguments passed on to `layer()`'s `params` argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the `position` argument, or aesthetics that are required can *not* be passed through Unknown arguments that are not part of the 4 categories below are ignored.
- Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, `colour = "red"` or `linewidth = 3`. The geom's documentation has an **Aesthetics** section that lists the available options. The 'required' aesthetics cannot be passed on to the `params`. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.
 - When constructing a layer using a `stat_*()` function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
 - Inversely, when constructing a layer using a `geom_*()` function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
 - The `key_glyph` argument of `layer()` may also be passed on through This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.
- `arrow` Arrow specification, as created by `grid::arrow()`.
- `arrow.fill` fill colour to use for the arrow head (if closed). NULL means use colour aesthetic.
- `lineend` Line end style (round, butt, square).
- `linejoin` Line join style (round, mitre, bevel).
- `linemitre` Line mitre limit (number greater than 1).
- `na.rm` If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
- `show.legend` logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
- `inherit.aes` If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data

and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. `annotation_borders()`.

geom The geometric object to use to display the data for this layer. When using a `stat_*()` function to construct a layer, the `geom` argument can be used to override the default coupling between stats and geoms. The `geom` argument accepts the following:

- A `Geom` ggproto subclass, for example `GeomPoint`.
- A string naming the geom. To give the geom as a string, strip the function name of the `geom_` prefix. For example, to use `geom_point()`, give the geom as "point".
- For more information and other ways to specify the geom, see the [layer geom](#) documentation.

fun Function to use. Either 1) an anonymous function in the base or `rlang` formula syntax (see `rlang::as_function()`) or 2) a quoted or character name referencing a function; see examples. Must be vectorised.

xlim Optionally, specify the range of the function.

n Number of points to interpolate along the x axis.

args List of additional arguments passed on to the function defined by `fun`.

Value

a `geom_function` of the parameter

Examples

```
ggplot2::ggplot()+
  sim_geom_function(example_poisson_rt(), xlim=as.Date("2019-12-29")+c(0,80))+
  ggplot2::scale_x_date()
```

sim_multinomial	<i>Generate a multinomial outbreak defined by per class growth rates and a poisson model</i>
-----------------	--

Description

Generate a multinomial outbreak defined by per class growth rates and a poisson model

Usage

```
sim_multinomial(
  changes = dplyr::tibble(t = c(0, 20, 40, 60, 80), variant1 = c(0.1, 0, -0.1, 0, 0.1),
    variant2 = c(0.15, 0.05, -0.05, -0.01, 0.05), variant3 = c(0, 0.05, -0.05, +0.05,
    -0.05), ),
  initial = c(100, 100, 100),
  time_unit = "1 day",
  ...
)
```

Arguments

changes	a list of time points in column t and growth rates per week per class, in other columns.
initial	the size of the initial outbreak per class. There should be one entry per class
time_unit	e.g. a daily or weekly time series: "1 day", "7 days"
...	Named arguments passed on to sim_poisson_model
fn_imports	a function that takes input vector t and returns the number of imported cases at times t.
seed	a random seed
kappa	a dispersion parameter. 1 is no dispersion (compared to poisson), smaller values mean more dispersion.
max_time	the desired length of the time series,
fn_growth	a function that takes input vector t and returns the growth rates at times t

Value

a case count time series including class, count and time columns

Examples

```
if (interactive()) {
  plot_counts(
    sim_multinomial() %>% dplyr::glimpse()
  )
}
```

sim_poisson_model	<i>Generate an outbreak case count series defined by growth rates using a poisson model.</i>
-------------------	--

Description

Generate an outbreak case count series defined by growth rates using a poisson model.

Usage

```
sim_poisson_model(
  changes = dplyr::tibble(t = c(0, 20, 40, 60, 80), growth = c(0.1, 0, -0.1, 0, 0.1)),
  kappa = 1,
  max_time = 104,
  seed = Sys.time(),
  fn_growth = cfg_step_fn(changes),
  fn_imports = ~ifelse(.x == 0, 100, 0),
  time_unit = "1 day"
)
```

Arguments

changes	a dataframe holding change time points (t) and growth rate per week (growth) columns
kappa	a dispersion parameter. 1 is no dispersion (compared to poisson), smaller values mean more dispersion.
max_time	the desired length of the time series,
seed	a random seed
fn_growth	a function that takes input vector t and returns the growth rates at times t
fn_imports	a function that takes input vector t and returns the number of imported cases at times t.
time_unit	e.g. a daily or weekly time series: "1 day", "7 days"

Value

A dataframe of case counts

A dataframe containing the following columns:

- statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

Minimally grouped by: statistic (and other groupings may be present).

Examples

```
tmp2 = sim_poisson_model(seed=100, fn_imports = ~ ifelse(.x %in% c(0,50),100,0))

if (interactive()) {
  ggplot2::ggplot(tmp2)+ggplot2::geom_point(ggplot2::aes(x=time,y=count))
}
```

sim_poisson_Rt_model	<i>Generate an outbreak case count series defined by Reproduction number using a poisson model.</i>
----------------------	---

Description

Generate an outbreak case count series defined by Reproduction number using a poisson model.

Usage

```
sim_poisson_Rt_model(
  changes = dplyr::tibble(t = c(0, 40), rt = c(2.5, 0.8)),
  kappa = 1,
  max_time = 80,
  seed = Sys.time(),
  fn_Rt = cfg_step_fn(changes),
  fn_imports = ~ifelse(.x == 0, 30, 0),
  fn_ip = ~example_ip(),
  time_unit = "1 day"
)
```

Arguments

changes	a dataframe holding change time points (t) and reproduction number (rt) columns
kappa	a dispersion parameter. 1 is no dispersion (compared to poisson), smaller values mean more dispersion.
max_time	the desired length of the time series,
seed	a random seed
fn_Rt	a function that takes input vector t and returns the instantaneous reproduction number at time t
fn_imports	a function that takes input vector t and returns the number of imported cases at times t.
fn_ip	a function that takes input vector t and returns an infectivity profile at times t.
time_unit	e.g. a daily or weekly time series: "1 day", "7 days"

Value

A dataframe of case counts

A dataframe containing the following columns:

- statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a 'time_period'

Minimally grouped by: statistic (and other groupings may be present).

Examples

```
tmp = sim_poisson_Rt_model(kappa=1, seed=100, fn_imports = ~ ifelse(.x %in% c(0,50),100,0))

if (interactive()) {
  ggplot2::ggplot(tmp,ggplot2::aes(x=time,y=count))+ggplot2::geom_point()+
  ggplot2::geom_line()
}
```

sim_seir_model	<i>SEIR model with time-varying transmission parameter</i>
----------------	--

Description

This function simulates an SEIR (Susceptible-Exposed-Infectious-Recovered) model where the transmission rate (beta) varies over time.

Usage

```
sim_seir_model(
  changes = dplyr::tibble(t = c(0, 30), dBeta = c(1, 0.5)),
  mean_latent_period,
  mean_gen_time,
  R0 = 2.5,
  fn_dBeta = cfg_step_fn(changes),
  N = 10000,
  imports = 10,
  max_time = 104,
  seed = Sys.time()
)
```

Arguments

changes	a dataframe holding change time points (t) and the proportional increase in the transmission rate in the dBeta column
mean_latent_period	Mean time from infection to becoming infectious (E to I), assumed to be exponentially distributed.
mean_gen_time	The average generation time (will be latent period+infectious duration).
R0	the initial reproduction number
fn_dBeta	A function of time t that returns the multiple of transmission rate at that time point. Transmission (beta) at time zero is defined by R0 and the infectious duration, This is multiplied at each time by this factor.
N	Total population size. Defaults to 10,000.
imports	Initial number of infectious individuals. Defaults to 10.
max_time	Maximum simulation time (in days or time units). Defaults to 100.
seed	a random seed

Details

The latent period (time from infection to becoming infectious) is assumed to be exponentially distributed. The infectious period is derived from the given mean of the generation time and the latent period.

The model assumes:

- Latent period $\sim$ Exponential(sigma), where $\text{sigma} = 1 / \text{mean_latent_period}$
- Infectious period $\sim$ Exponential(gamma), where $\text{gamma} = 1 / (\text{mean_gen_time} - \text{mean_latent_period})$
- Generation time $\sim$ Exponential(sigma) + Exponential(gamma)
- $\text{beta0} = R0 * \text{gamma}$
- Transmission rate $\text{beta}(t) = \text{fn_dBeta}(t) * \text{beta0}$

The generation time is defined as the sum of the latent and infectious periods.

Value

A dataframe containing the following columns:

- statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a time_period

Minimally grouped by: statistic (and other groupings allowed).

Examples

```
# Example: Lockdown after day 30

seir_output <- sim_seir_model(
  mean_latent_period = 4,
  mean_gen_time = 7,
  R0 = 2.5,
  fn_dBeta = function(t) ifelse(t < 30, 1, 0.5)
)

if (interactive()) {
  plot_ip(attr(seir_output, "ip"))
  plot_counts(seir_output)
}
```

sim_summarise_linelist

Summarise a line list

Description

This function converts a line list into a daily count of incident cases, plus infections, admissions, deaths, test samples, test results if present. Censoring of these counts can also be defined. Whilst summarising various network measures such as the forward looking case reproduction number are also calculated.

Usage

```
sim_summarise_linelist(
  df = i_sim_linelist,
  ...,
  censoring = list(admitted = function(t) rgamma2(t, mean = 5), death = function(t)
    rgamma2(t, mean = 10), sample = function(t, result_delay) result_delay),
  max_time = max(df$time)
)
```

Arguments

df	a line list dataframe arising from e.g. <code>sim_branching_process()</code> - a dataframe with columns: • id (unique_id) - Patient level unique id • time (ggoutbreak::time_period) - Time of infection. A ‘time_period’ Any grouping allowed.
...	the grouping to include in the summarisation.
censoring	a named list of column names (without the _time suffix) of the kind created by <code>sim_delay()</code> or <code>sim_apply_delay()</code> , and an associated function defining the delay of reporting that the column experiences. For this function <code>t</code> (or <code>.x</code> for a purrr lambda) will refer to the <code>XX_time</code> column, i.e. whenever the event that is being reported happened and <code>time</code> the simulation infection time. N.B. since infection is not observed you can’t censor it.
max_time	the censoring time for this observation. If this is a vector there will be multiple time series in the output

Value

a count data frame with additional statistics.

A dataframe containing the following columns:

- statistic (character) - An identifier for the statistic, whether that be infections, admissions, deaths
- count (positive_integer) - Positive case counts associated with the specified time frame
- time (ggoutbreak::time_period + group_unique) - A (usually complete) set of singular observations per unit time as a ‘time_period’

Minimally grouped by: statistic (and other groupings may be present).

Examples

```
sim = sim_branching_process(
  changes = dplyr::tibble(t = c(0,40), R = c(1.7,0.8)),
  max_time = 120,
  seed = 100,
  fn_imports = ~ ifelse(.x==0,100,0)
)
```

```

tmp = sim %>% sim_summarise_linelist()

p1 = plot_counts(tmp)

p2 = ggplot2::ggplot(tmp, ggplot2::aes(x=as.Date(time)))+
  ggplot2::geom_point(ggplot2::aes(y=rt.case,colour="case"))+
  ggplot2::geom_point(ggplot2::aes(y=rt.inst,colour="instantaneous"))+
  ggplot2::geom_line(ggplot2::aes(y=rt.weighted))+
  ggplot2::coord_cartesian(ylim=c(0,3.5))+
  ggplot2::xlab(NULL)

patchwork::wrap_plots(p1,p2,ncol=1,axes="collect")

```

sim_test_data	<i>Generate a simple time-series of cases based on a growth rate step function</i>
---------------	--

Description

This time-series has no statistical noise and is useful for testing things. It has a fixed known value for infections and growth rate (fixed at 0.05 and -0.05 per day), and a instantaneous reproduction number which is based on the provided infectivity profile. A fixed denominator gives a known proportion and a relative growth rate that is the same as the growth rate.

Usage

```
sim_test_data(ip = example_ip(), duration = 500, period = 50)
```

Arguments

ip	an infectivity profile. any uncertainty will be collapsed into the central distribution.
duration	the total length of the time-series
period	the duration of each positive or negative growth phase

Value

a time series with count, incidence, growth, rt, proportion and relative.growth columns

Examples

```
sim_test_data() %>% dplyr::glimpse()
```

summarise_ip	<i>Generate a single infectivity profile from multiple bootstraps</i>
--------------	---

Description

Generate a single infectivity profile from multiple bootstraps

Usage

```
summarise_ip(ip = i_empirical_ip)
```

Arguments

ip	the infectivity profile to summarise. a0 and a1 columns are optional if tau is given. - a dataframe with columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
----	--

Value

an infectivity profile

time_aggregate	<i>Aggregate time series data preserving the time series</i>
----------------	--

Description

This function operates on timeseries data not linelists (see [time_summarise\(\)](#) for line lists. If a very granular timeseries is regrouped and this function is applied the resulting dataframe will be

Usage

```
time_aggregate(
  df = i_timestamped,
  ...,
  .groups = NULL,
  .cols = NULL,
  .fns = NULL
)
```

Arguments

<code>df</code>	an optionally grouped time series. Grouping should not include the time column. The grouping works differently from <code>dplyr::summarise</code> in that the last level of non-time groups is lost in this operation, so the subgroup you wish to aggregate should be included in the grouping.
<code>...</code>	A set of <code>dplyr::summarise</code> statements, or additional parameters for <code>.fns</code>
<code>.groups</code>	as per <code>dplyr::summarise</code>
<code>.cols</code>	Optional dplyr column specification for the data that will be summarised. This is useful if there are lots of columns you want to summarise using the same function (e.g. <code>sum</code> usually). <code>.cols</code> is passed to a <code>dplyr::across</code> call. if <code>.fns</code> is given and the <code>.cols</code> parameter is not specified then the columns to summarise are automatically identified. In doing this any Date columns are dropped. If this is not what you want then <code>.cols</code> or <code>...</code> must be given
<code>.fns</code>	Optional a set of function specifications as per <code>dplyr::across</code>

Value

the summarised time series preserving the time column, and with the grouping structure involving one fewer levels than the input

Examples

```
example_england_covid_by_age() %>%
  time_aggregate(count = sum(count), denom = sum(denom)) %>%
  dplyr::glimpse()

example_england_covid_by_age() %>%
  time_aggregate(.fns=mean, .cols=dplyr::where(is.numeric)) %>%
  dplyr::glimpse()
```

time_summarise	<i>Summarise data from a line list to a time-series of counts.</i>
----------------	--

Description

This principally is designed to take a record of single events and produce a summary time-series count of events by group, class and date. The default behaviour is to guess the cadence of the input data and summarise the event line list to a (set of) regular time-series counts for use in incidence and growth rate estimates.

Usage

```
time_summarise(
  df = i_dated,
  unit,
  anchor = "start",
```

```

    rectangular = FALSE,
    ...,
    .fill = list(count = 0)
  )

```

Arguments

<code>df</code>	a line list of data you want to summarise, optionally grouped. If this is grouped then each group is treated independently. The remaining columns must contain a date column and may contain a <code>class</code> column. If a count column is present the counts will be summed, otherwise each individual row will be counted as a single event (as a line list)
<code>unit</code>	a period e.g. "1 week"
<code>anchor</code>	one of a date, "start" or "end" or a weekday name e.g. "mon" this will always be one of the start of the time periods we are cutting into
<code>rectangular</code>	should the resulting time series be the same length for all groups? This is only the case if you can be sure that your data is complete for all subgroups, otherwise missing data will be treated as zero counts. This is important if leading and trailing missing data in one subgroup can be due to a reporting delay in that subgroup, in which case a rectangular time series will erroneously fill in zero counts for this missing data.
<code>...</code>	a specification for a <code>dplyr::summary(...)</code> - optional, and if not provided a <code>count = dplyr::n()</code> or a <code>count = sum(count)</code> is performed.
<code>.fill</code>	a list similar to <code>tidyr::complete</code> for default values to fill variables with.

Details

If the data is given with a `class` column the time series are interpreted as having a denominator, consisting of all the different classes within a time period. This may be subtypes (e.g. variants, serotypes) or markers for test positivity. In either case the resulting time series will have counts for all classes and denominators for the combination.

There is flexibility for other kinds of summarisation if the raw data is not count based (e.g. means of continuous variables) but in this case a the `slider` package is usually going to be better, as `time_summarise` will only look at non overlapping time periods with fixed lengths.

There is another use case where an existing timeseries on a particular frequency is aggregated to another less frequent basis (e.g. moving from a daily timeseries to a weekly one). In this case the input will contain a `count` column. In this mode no checks are made that the more frequent events are all present before summarisation so the result may include different numbers of input periods (e.g. going from weeks to months may be 4 or 5 weeks in each month). If a `class` column is present the a classwise denominator is calculated

Value

The output depends on whether or not the input was grouped and had a `class` column. The most detailed output will be:

A dataframe containing the following columns:

- `denom` (`positive_integer`) - Total test counts associated with the specified time frame
- `count` (`positive_integer`) - Positive case counts associated with the specified time frame
- `time` (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a `time_period`

Any grouping allowed.

or a more minimal output if the input is only a plain list of dated events:

A dataframe containing the following columns:

- `count` (`positive_integer`) - Positive case counts associated with the specified time frame
- `time` (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a `time_period`

Any grouping allowed.

Examples

```
# a set of random dates with a class column:
input = dplyr::tibble(
  class = rep(c("A", "B"), 1000),
  date = as.Date("2020-01-01") + sample.int(100, 2000, TRUE)
)

# summarise daily counts, including denominators:
daily = time_summarise(input, unit="1 day") %>% dplyr::glimpse()

# summarise weekly counts, for sample data:
time_summarise(input, unit="1 week") %>% dplyr::glimpse()

# summarise daily counts into weekly:
time_summarise(daily, unit="1 week") %>% dplyr::glimpse()
```

timeseries

Coerce an object to a ggoutbreak compatible time series dataframe

Description

Coerce an object to a ggoutbreak compatible time series dataframe

Usage

```
timeseries(x, ...)

## Default S3 method:
timeseries(x, ...)

## S3 method for class 'incidence2'
timeseries(x, ...)
```

```
## S3 method for class 'data.frame'
timeseries(
  x,
  ...,
  date = NULL,
  count = NULL,
  denom = NULL,
  population = NULL,
  class = NULL,
  declutter = FALSE
)

## S3 method for class 'incidence'
timeseries(x, ..., declutter = FALSE)
```

Arguments

<code>x</code>	An object to coerce e.g. a data frame
<code>...</code>	Named arguments passed on to as.time_period
	<code>unit</code> the length of one unit of time. This will be either a integer number of days, or a specification such as "1 week", or another <code>time_period</code> . If <code>x</code> is a <code>time_period</code> , and the unit is different to that of <code>x</code> this will return a rescaled <code>time_period</code> using the new units.
	<code>start_date</code> the zero time date as something that can be coerced to a date. If the <code>x</code> input is already a <code>time_period</code> and this is different to its <code>start_date</code> then <code>x</code> will be recalibrated to use the new start date.
	<code>anchor</code> only relevant if <code>x</code> is a vector of dates, this is a date, or "start" or "end" or a weekday name e.g. "mon". With the vector of dates in <code>x</code> it will use this anchor to find a reference date for the time-series. If not provided then the current defaults will be used. (see set_defaults())
<code>date</code>	R expression defining a date column (as a Date - optional)
<code>count</code>	R expression defining a count column (as an int - optional)
<code>denom</code>	R expression defining a denominator column (as an int - optional)
<code>population</code>	R expression defining a population column (as an numeric - optional)
<code>class</code>	R expression defining a class column (as an factor - optional)
<code>declutter</code>	logical flag. If TRUE then unnecessary original columns will be dropped

Value

minimally a case count dataframe A dataframe containing the following columns:

- `count` (positive_integer) - Positive case counts associated with the specified time frame
- `time` (`ggoutbreak::time_period + group_unique`) - A (usually complete) set of singular observations per unit time as a `time_period`

Any grouping allowed.

Examples

```

utils::data("mers_2014_15", package="EpiEstim")

set_default_unit("1 day")

# Use an expression to generate a case count:
# N.B. complex column names need to be surrounded with backticks like this:

tmp = mers_2014_15$incidence %>%
  timeseries(
    date = `mers$dates`,
    count = local+imported
  ) %>%
  dplyr::glimpse()

if (interactive()) {
  ip = make_fixed_ip(mean = mers_2014_15$si$mean_si, sd = mers_2014_15$si$std_si)
  tmp %>% poisson_locfit_model(ip=ip,window=14) %>% plot_rt()
}

# remove columns not needed by `ggoutbreak`
outbreaks::dengue_fais_2011 %>%
  timeseries(date = onset_date, count = value, declutter = TRUE) %>%
  dplyr::glimpse()

# date column already exists, could use `onset` or `death` as count
outbreaks::ebola_kikwit_1995 %>%
  timeseries(count = onset) %>%
  dplyr::glimpse()

# This data set needs grouping to make it a unique timeseries:
ukc19::ltla_cases %>%
  timeseries(anchor="start", unit="1 day") %>%
  dplyr::glimpse()

# from an incidence object:
if (requireNamespace("incidence",quietly = TRUE)) {

  onset = outbreaks::ebola_sim$linelist$date_of_onset
  sex = outbreaks::ebola_sim$linelist$gender
  inc.week.gender = incidence::incidence(onset, interval = 7, groups = sex, standard = FALSE)

  inc.week.gender %>% timeseries() %>% dplyr::glimpse()

  d = Sys.Date() + sample(-3:10, 10, replace = TRUE)
  di = incidence::incidence(d, interval = "week", first_date = Sys.Date() - 10, standard = TRUE)
  di %>% timeseries(declutter=TRUE) %>% dplyr::glimpse()
}

if (requireNamespace("incidence2",quietly = TRUE)) {

```

```

dat = outbreaks::ebola_sim_clean$linelist
x = incidence2::incidence(dat, "date_of_onset", groups = c("gender", "hospital"), interval="week")
x %>% timeseries() %>% dplyr::glimpse()
}

```

vcov_from_residuals *Estimate Parametric VCOV Matrix from Residuals*

Description

Estimates the variance-covariance matrix of log-incidence estimates by modelling residual autocorrelation with an exponential decay model.

Usage

```

vcov_from_residuals(
  data,
  mu,
  sigma,
  max_lag = 10,
  min_alpha = 0.01,
  max_alpha = 2
)

```

Arguments

data	Numeric vector: observed case counts I_t
mu	Numeric vector: estimated log-incidence (log λ_t)
sigma	Numeric vector: estimated standard error of log λ_t
max_lag	Integer: maximum lag for ACF estimation (default: 10)
min_alpha	Positive number: minimum decay rate (default: 0.01)
max_alpha	Positive number: maximum decay rate (default: 2.0)

Details

This function was generated by Qwen3-235B-A22B-2507

Value

List with:

- vcov_matrix: T x T estimated VCOV matrix for log λ_t
- alpha: fitted decay parameter
- acf_obs: observed ACF of Pearson residuals
- acf_fit: fitted ACF values
- residuals: Pearson residuals used
- times: time indices

Examples

```

T <- 50
mu <- 5 + 0.05 * (1:T) + stats::arima.sim(list(ar = 0.8), n = T, sd = 0.3)
sigma <- rep(0.2, T)
lambda <- exp(mu)
data <- stats::rpois(T, lambda)

result <- vcov_from_residuals(data, mu, sigma, max_lag = 8)
dim(result$vcov_matrix) # Should be T x T

```

wallinga_lipsitch	<i>Calculate the reproduction number from a growth rate estimate and an infectivity profile</i>
-------------------	---

Description

This function uses a single empirical distribution for the infectivity profile / generation time. If multiple are provided then the average central value is chosen (i.e. this does not propagate uncertainty in infectivity profile)

Usage

```

wallinga_lipsitch(
  r,
  y = i_empirical_ip,
  a1 = seq(0.5, length.out = length(y)),
  a0 = dplyr::lag(a1, default = 0)
)

```

Arguments

r	a growth rate (may be a vector)
y	an empirical infectivity profile either as a probability vector or as a dataframe of format: A dataframe containing the following columns: • boot (anything + default(1)) - a bootstrap identifier • probability (proportion) - the probability of new event during this period. • a0 (double) - the beginning of the time period (in days) • a1 (double) - the end of the time period (in days) Minimally grouped by: boot (and other groupings allowed).
a1	the end time of the infectivity profile probability estimate (defaults to 0.5,1.5,2.5,...).
a0	the start time of the infectivity profile probability estimate (defaults to 0,0.5,1.5,...).

Value

a reproduction number estimate based on r

Examples

```
# using a probability vector.
wallinga_lipsitch(r=seq(-0.1,0.1,length.out=9), y=stats::dgamma(1:50, 5,2))

# using an infectivity profile
wallinga_lipsitch(r=seq(-0.1,0.1,length.out=9), y=example_ip())
```

wedge	<i>Wedge distribution</i>
-------	---------------------------

Description

The wedge distribution has a support of 0 to 1 and has a linear probability density function over that support.

Arguments

n	number of observations
x	vector of quantiles
q	vector of quantiles
p	vector of probabilities
log	logical; if TRUE, probabilities p are given as log(p).
log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$ otherwise $P[X > x]$.
a	a gradient from -2 (left skewed) to 2 (right skewed)

Details

The `rwedge` can be combined with quantile functions to skew standard distributions, or introduce correlation or down weight certain parts of the distribution.

Value

a vector of probabilities, quantiles, densities or samples.

Examples

```
pwedge(seq(0,1,0.1), a=1)
dwedge(seq(0,1,0.1), a=1)
qwedge(c(0.25,0.5,0.75), a=-1)

stats::cor(
  stats::qnorm(rwedge(1000, a=2)),
  stats::qnorm(rwedge(1000, a=-2))
)
```

weekdays.time_period *Extract Parts of a POSIXt or Date Object*

Description

Extract the weekday, month or quarter, or the Julian time (days since some origin). These are generic functions: the methods for the internal date-time classes are documented here.

Usage

```
## S3 method for class 'time_period'
weekdays(x, abbreviate = FALSE)
```

Arguments

x an object inheriting from class "POSIXt" or "Date".
abbreviate logical vector (possibly recycled). Should the names be abbreviated?

Value

weekdays and months return a character vector of names in the locale in use, i.e., `Sys.getlocale("LC_TIME")`.
quarters returns a character vector of "Q1" to "Q4".
julian returns the number of days (possibly fractional) since the origin, with the origin as a "origin" attribute. All time calculations in R are done ignoring leap-seconds.

Note

Other components such as the day of the month or the year are very easy to compute: just use `as.POSIXlt` and extract the relevant component. Alternatively (especially if the components are desired as character strings), use `strftime`.

See Also

`DateTimeClasses`, `Date`; `Sys.getlocale("LC_TIME")` crucially for `months()` and `weekdays()`.

Examples

```
## first two are locale dependent:
weekdays(.leap.seconds)
months (.leap.seconds)
quarters(.leap.seconds)

## Show how easily you get month, day, year, day (of {month, week, yr}), ... :
## (remember to count from 0 (!): mon = 0..11, wday = 0..6, etc !!)

##' Transform (Time-)Date vector to convenient data frame :
dt2df <- function(dt, dName = deparse(substitute(dt))) {
  DF <- as.data.frame(unclass(as.POSIXlt( dt )))
```

```

    `names<-`(cbind(dt, DF, deparse.level=0L), c(dName, names(DF)))
  }
  ## e.g.,
  dt2df(.leap.seconds)    # date+time
  dt2df(Sys.Date() + 0:9) # date

  ##' Even simpler: Date -> Matrix - dropping time info {sec,min,hour, isdst}
  d2mat <- function(x) simplify2array(unclass(as.POSIXlt(x))[4:7])
  ## e.g.,
  d2mat(seq(as.Date("2000-02-02"), by=1, length.out=30)) # has R 1.0.0's release date

  ## Julian Day Number (JDN, https://en.wikipedia.org/wiki/Julian\_day)
  ## is the number of days since noon UTC on the first day of 4317 BCE.
  ## in the proleptic Julian calendar. To more recently, in
  ## 'Terrestrial Time' which differs from UTC by a few seconds
  ## See https://en.wikipedia.org/wiki/Terrestrial\_Time
  julian(Sys.Date(), -2440588) # from a day
  floor(as.numeric(julian(Sys.time()))) + 2440587.5 # from a date-time

```

Index

* **delay_distribution**

- [format_ip, 30](#)
- [make_empirical_ip, 53](#)
- [make_fixed_ip, 54](#)
- [make_gamma_ip, 54](#)
- [make_posterior_ip, 56](#)
- [make_resampled_ip, 58](#)
- [omega_matrix, 66](#)
- [summarise_ip, 172](#)

* **distributions**

- [dbeta2, 19](#)
- [dgamma, 20](#)
- [dgamma2, 21](#)
- [dlnorm2, 22](#)
- [dlogitnorm, 23](#)
- [dlogitnorm2, 24](#)
- [dnbinom2, 25](#)
- [dnull, 26](#)
- [dwedge, 28](#)
- [pbeta2, 66](#)
- [pcgamma, 67](#)
- [pgamma2, 68](#)
- [plnorm2, 69](#)
- [plogitnorm, 70](#)
- [plogitnorm2, 71](#)
- [pnbinom2, 92](#)
- [pnull, 94](#)
- [pwedge, 105](#)
- [qbeta2, 106](#)
- [qgamma, 107](#)
- [qgamma2, 108](#)
- [qlnorm2, 109](#)
- [qlogitnorm, 110](#)
- [qlogitnorm2, 111](#)
- [qnbinom2, 111](#)
- [qnull, 113](#)
- [qwedge, 116](#)
- [rbern, 117](#)
- [rbeta2, 117](#)

- [rcategorical, 118](#)

- [rcgamma, 119](#)

- [rdiscgamma, 120](#)

- [rexpgrowth, 123](#)

- [rexpgrowthI0, 124](#)

- [rgamma2, 125](#)

- [rlnorm2, 126](#)

- [rlogitnorm, 127](#)

- [rlogitnorm2, 128](#)

- [rnbinom2, 128](#)

- [rnull, 130](#)

- [rwedge, 146](#)

- [wedge, 180](#)

* **models**

- [doubling_time, 27](#)

- [gam_delayed_reporting, 30](#)

- [gam_knots, 31](#)

- [gam_nb_model_fn, 32](#)

- [gam_poisson_model_fn, 33](#)

- [growth_rate_from_incidence, 37](#)

- [growth_rate_from_prevalence, 38](#)

- [growth_rate_from_proportion, 40](#)

- [infer_population, 41](#)

- [infer_prevalence, 42](#)

- [infer_rate_ratio, 44](#)

- [infer_risk_ratio, 45](#)

- [inv_wallinga_lipsitch, 47](#)

- [linelist, 51](#)

- [multinomial_nnet_model, 61](#)

- [normalise_count, 63](#)

- [normalise_incidence, 64](#)

- [poisson_gam_model, 94](#)

- [poisson_glm_model, 97](#)

- [poisson_locfit_model, 99](#)

- [proportion_glm_model, 101](#)

- [proportion_locfit_model, 103](#)

- [rescale_model, 122](#)

- [rt_cori, 130](#)

- [rt_epiestim, 132](#)

- rt_from_growth_rate, 134
- rt_from_incidence, 136
- rt_from_renewal, 138
- rt_incidence_reference_implementation, 140
- rt_incidence_timeseries_implementation, 143
- timeseries, 175
- wallinga_lipsitch, 179
- * test**
 - cfg_beta_prob_rng, 9
 - cfg_gamma_ip_fn, 10
 - cfg_ip_sampler_rng, 10
 - cfg_linear_fn, 11
 - cfg_step_fn, 12
 - cfg_transition_fn, 12
 - cfg_weekly_gamma_rng, 13
 - cfg_weekly_ip_fn, 14
 - cfg_weekly_proportion_rng, 14
 - quantify_lag, 114
 - score_estimate, 148
 - sim_apply_ascertainment, 152
 - sim_apply_delay, 153
 - sim_branching_process, 155
 - sim_convolution, 157
 - sim_delay, 159
 - sim_delayed_observation, 160
 - sim_events, 161
 - sim_geom_function, 162
 - sim_multinomial, 164
 - sim_poisson_model, 165
 - sim_poisson_Rt_model, 166
 - sim_seir_model, 168
 - sim_summarise_linelist, 169
 - sim_test_data, 171
- * time_period**
 - as.time_period, 5
 - cut_date, 15
 - date_seq, 16
 - date_seq.Date, 17
 - date_seq.numeric, 18
 - date_seq.time_period, 18
 - fdmy, 29
 - is.Date, 48
 - julian.time_period, 49
 - labels.time_period, 50
 - max_date, 59
 - min_date, 59
 - months.time_period, 60
 - quarters.time_period, 115
 - set_defaults, 150
 - time_aggregate, 172
 - time_summarise, 173
 - weekdays.time_period, 181
- * vis**
 - breaks_log1p, 8
 - geom_events, 34
 - integer_breaks, 47
 - logit_trans, 52
 - plot_cases, 72
 - plot_counts, 73
 - plot_growth_phase, 75
 - plot_growth_rate, 79
 - plot_incidence, 81
 - plot_ip, 84
 - plot_multinomial, 84
 - plot_prevalence, 86
 - plot_proportion, 87
 - plot_proportions_data, 89
 - plot_rt, 90
 - scale_x_log1p, 146
 - scale_x_logit, 147
 - scale_y_log1p, 147
 - scale_y_logit, 148
- * wrangling**
 - reband_discrete, 120
- aes(), 162
- annotation_borders(), 164
- as.POSIXlt, 49, 60, 115, 181
- as.time_period, 5, 36, 52, 80, 83, 91, 176
- breaks_log1p, 8
- cfg_beta_prob_rng, 9
- cfg_gamma_ip_fn, 10
- cfg_gamma_ip_fn(), 158, 161
- cfg_ip_sampler_rng, 10
- cfg_linear_fn, 11
- cfg_step_fn, 12
- cfg_transition_fn, 12
- cfg_transition_fn(), 156
- cfg_weekly_gamma_rng, 13
- cfg_weekly_ip_fn, 14
- cfg_weekly_ip_fn(), 158, 161
- cfg_weekly_proportion_rng, 14
- cfg_weekly_proportion_rng(), 153

- `coord_cartesian()`, 36
- `cut_date`, 15
- `Date`, 49, 60, 115, 181
- `date_seq`, 16
- `date_seq.Date`, 17
- `date_seq.numeric`, 18
- `date_seq.time_period`, 18
- `date_to_time(as.time_period)`, 5
- `DateTimeClasses`, 49, 60, 115, 181
- `dbeta2`, 19
- `dbinom`, 26, 93, 113, 129, 130
- `dgamma`, 20
- `dgamma2`, 21
- `dgeom`, 26, 93, 113, 130
- `Distributions`, 23, 26, 70, 93, 110, 113, 127, 130
- `dlnorm2`, 22
- `dlogitnorm`, 23
- `dlogitnorm2`, 24
- `dnbinom2`, 25
- `dnorm`, 23, 70, 110, 127
- `dnull`, 26
- `double`, 26, 93, 112, 129
- `doubling_time`, 27
- `dpois`, 26, 93, 113, 130
- `dwedge`, 28
-
- `expansion()`, 36
-
- `fdmy`, 29
- `format_ip`, 30
-
- `gam_delayed_reporting`, 30
- `gam_delayed_reporting()`, 95
- `gam_knots`, 31, 31–33
- `gam_nb_model_fn`, 32
- `gam_poisson_model_fn`, 33
- `gam_poisson_model_fn()`, 95
- `geom_events`, 34, 72, 74, 77, 80, 82, 85, 87–89, 91
- `geom_truth`, 80, 83, 91
- `ggplot2::facet_wrap`, 77
- `ggplot2::geom_function`, 162
- `ggplot2::scale_x_date`, 35
- `grid::arrow()`, 163
- `growth_rate_from_incidence`, 37
- `growth_rate_from_prevalence`, 38
- `growth_rate_from_proportion`, 40
-
- `guides()`, 36
-
- `infer_population`, 41
- `infer_prevalence`, 42
- `infer_rate_ratio`, 44
- `infer_risk_ratio`, 45
- `integer`, 26, 93, 112, 129
- `integer_breaks`, 47
- `inv_wallinga_lipsitch`, 47
- `is.Date`, 48
- `is.time_period(as.time_period)`, 5
-
- `julian.time_period`, 49
-
- `key_glyphs`, 163
-
- `label_parsed()`, 78
- `label_value()`, 78
- `labeller()`, 78
- `labels.time_period`, 50
- `lambda`, 35, 36
- `layer geom`, 164
- `layer position`, 163
- `layer stat`, 162
- `layer()`, 163
- `linelist`, 51
- `logit_trans`, 52
-
- `make_empirical_ip`, 53
- `make_fixed_ip`, 54
- `make_gamma_ip`, 54
- `make_posterior_ip`, 56
- `make_resampled_ip`, 58
- `max_date`, 59
- `min_date`, 59
- `months.time_period`, 60
- `multinomial_nnet_model`, 61
-
- `normalise_count`, 63
- `normalise_count()`, 73
- `normalise_incidence`, 64
-
- `omega_matrix`, 66
-
- `pbeta`, 26, 93, 113, 129
- `pbeta2`, 66
- `pcgamma`, 67
- `pgamma`, 25, 93, 112, 129
- `pgamma2`, 68
- `plnorm2`, 69

- plogitnorm, 70
- plogitnorm2, 71
- plot_cases, 72
- plot_counts, 73
- plot_growth_phase, 75
- plot_growth_rate, 79
- plot_incidence, 81
- plot_ip, 84
- plot_multinomial, 84
- plot_prevalence, 86
- plot_proportion, 87
- plot_proportions_data, 89
- plot_rt, 90
- pnbinom2, 92
- pnull, 94
- poisson_gam_model, 94
- poisson_gam_model(), 137
- poisson_gam_model.censored, 95
- poisson_gam_model.incidence, 95
- poisson_glm_model, 97
- poisson_locfit_model, 82, 99
- poisson_locfit_model(), 44, 137
- proportion_glm_model, 101
- proportion_locfit_model, 103
- proportion_locfit_model(), 46
- pwedge, 105

- qbeta2, 106
- qcgamma, 107
- qgamma2, 108
- qlnorm2, 109
- qlogitnorm, 110
- qlogitnorm2, 111
- qnbinom2, 111
- qnull, 113
- quantify_lag, 114
- quantify_lag(), 149
- quarters.time_period, 115
- qwedge, 116

- rbern, 117
- rbeta2, 117
- rbeta2(), 9
- rcategorical, 118
- rcategorical(), 156
- rcgamma, 119
- rdisgamma, 120
- reband_discrete, 120
- rescale_model, 122
- rescale_model(), 136
- rexpgrowth, 123
- rexpgrowthI0, 124
- rgamma2, 125
- rlang::as_function(), 164
- rlnorm2, 126
- rlogitnorm, 127
- rlogitnorm2, 128
- rnbinom2, 128
- rnull, 130
- rt_cori, 130
- rt_epiestim, 132
- rt_from_growth_rate, 134
- rt_from_incidence, 136
- rt_from_renewal, 138
- rt_incidence_reference_implementation, 140
- rt_incidence_timeseries_implementation, 143
- rt_incidence_timeseries_implementation(), 137, 140
- rwedge, 146

- scale_x_log1p, 146
- scale_x_logit, 147
- scale_y_log1p, 147
- scale_y_logit, 148
- scales::censor(), 36
- scales::squish(), 36
- scales::squish_infinite(), 36
- score_estimate, 148
- seq.time_period(as.time_period), 5
- set_default_start(set_defaults), 150
- set_default_unit(set_defaults), 150
- set_defaults, 150
- set_defaults(), 6, 36, 52, 80, 83, 91, 176
- sim_apply_ascertainment, 152
- sim_apply_delay, 153
- sim_apply_delay(), 170
- sim_apply_delay.count_data, 154
- sim_apply_delay.linelist, 154
- sim_branching_process, 155
- sim_branching_process(), 12, 154, 159, 170
- sim_convolution, 157
- sim_delay, 159
- sim_delay(), 170
- sim_delayed_observation, 160
- sim_events, 161

`sim_geom_function`, 162
`sim_multinomial`, 164
`sim_poisson_model`, 165, 165
`sim_poisson_model()`, 152, 157, 161
`sim_poisson_Rt_model`, 166
`sim_seir_model`, 168
`sim_summarise_linelist`, 169
`sim_summarise_linelist()`, 152, 157, 161
`sim_test_data`, 171
`stats::dbeta()`, 20
`stats::dgamma()`, 21, 22
`stats::pbeta()`, 67
`stats::pgamma()`, 68, 69
`stats::qbeta()`, 106
`stats::qgamma()`, 107, 108
`stats::rbeta()`, 118
`stats::rgamma()`, 119, 120, 125
`strftime`, 49, 60, 115, 181
`strftime()`, 35
`summarise_ip`, 172
`Sys.getlocale`, 49, 60, 115, 181

`time_aggregate`, 172
`time_summarise`, 173
`time_summarise()`, 172
`time_to_date` (`as.time_period`), 5
`timeseries`, 175

`vars()`, 78
`vcov_from_residuals`, 178

`wallinga_lipsitch`, 179
`wedge`, 180
`weekdays.time_period`, 181
`with_defaults` (`set_defaults`), 150