Package ‘EpiNow2’

October 20, 2020

Type Package

Title Estimate Real-Time Case Counts and Time-Varying Epidemiological Parameters

Version 1.2.1

Description Estimates the time-varying reproduction number, rate of spread, and doubling time using a range of open-source tools (Abbott et al. (2020) <doi:10.12688/wellcomeopenres.16006.1>), and current best practices (Gostic et al. (2020) <doi:10.1101/2020.06.18.20134858>). It aims to help users avoid some of the limitations of naive implementations in a framework that is informed by community feedback and is under active development.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports cowplot, data.table, futile.logger (>= 1.4), future, future.apply, ggplot2, lubridate, methods, patchwork, progressr, purrr, R.utils (>= 2.0.0), Rcpp (>= 0.12.0), rlang (>= 0.4.7), rstan (>= 2.18.1), scales, stats, truncnorm,

Suggests countrycode, dplyr, EpiSoon, forecastHybrid, here, kableExtra, knitr, magrittr, rmarkdown, rnaturalearth, rstantools, spelling, tidyr, testthat

Additional_repositories https://epiforecasts.io/drat/

RoxygenNote 7.1.1

Biarch true

Depends R (>= 3.4.0)

LinkingTo BH (>= 1.66.0), Rcpp (>= 0.12.0), RcppEigen (>= 0.3.3.3.0), rstan (>= 2.21.1), StanHeaders (>= 2.21.0-5)

Language en-GB

NeedsCompilation yes

Author Sam Abbott [aut, cre] (<https://orcid.org/0000-0001-8057-8037>), Joel Hellewell [aut] (<https://orcid.org/0000-0003-2683-0849>), Robin Thompson [aut],

Katelyn Gostic [aut],
Katharine Sherratt [aut],
Sophie Meakin [aut],
James Munday [aut],
Nikos Bosse [aut],
Joe Hickson [aut],
Paul Mee [ctb],
Peter Ellis [ctb],
Hamada S. Badr [ctb] (<https://orcid.org/0000-0002-9808-2344>),
Michael DeWitt [ctb] (<https://orcid.org/0000-0001-8940-1967>),
EpiForecasts [aut],
Sebastian Funk [aut]

Maintainer Sam Abbott <`sam.abbott@lshtm.ac.uk`>

Repository CRAN

Date/Publication 2020-10-20 14:50:09 UTC

R topics documented:

adjust_infection_to_report *Adjust from Case Counts by Infection Date to Date of Report*

Description

Maps from cases by date of infection to date of report via date of onset.

Usage

```
adjust_infection_to_report(
  infections,
  delay_defs,
  reporting_model,
  reporting_effect,
  type = "sample",
  truncate_future = TRUE
)
```

Arguments

infections	`data.table` containing a `date` variable and a numeric `cases` variable.
delay_defs	A list of single row data.tables that each defines a delay distribution (model, parameters and maximum delay for each model). See `lognorm_dist_def` for an example of the structure.
reporting_model	A function that takes a single numeric vector as an argument and returns a single numeric vector. Can be used to apply stochastic reporting effects. See the examples for details.
reporting_effect	A numeric vector of length 7 that allows the scaling of reported cases by the day on which they report (1 = Monday, 7 = Sunday). By default no scaling occurs.
type	Character string indicating the method to use to transform counts. Supports either "sample" which approximates sampling or "median" would shift by the median of the distribution.
truncate_future	Logical, should cases be truncated if they occur after the first date reported in the data. Defaults to `TRUE`.

Value

A `data.table` containing a `date` variable (date of report) and a `cases` variable. If `return_onset = TRUE` there will be a third variable `reference` which indicates what the date variable refers to.

Examples

```
# define example cases
cases <- EpiNow2::example_confirmed[, `:=`(cases = as.integer(confirm))]

# define a single report delay distribution
delay_def <- EpiNow2::lognorm_dist_def(mean = 5,
                                      mean_sd = 1,
                                      sd = 3,
                                      sd_sd = 1,
                                      max_value = 30,
                                      samples = 1,
                                      to_log = TRUE)

# define a single incubation period
incubation_def <- EpiNow2::lognorm_dist_def(mean = EpiNow2::incubation_periods[1, ]$mean,
                                     mean_sd = EpiNow2::incubation_periods[1, ]$mean_sd,
                                           sd = EpiNow2::incubation_periods[1, ]$sd,
                                         sd_sd = EpiNow2::incubation_periods[1, ]$sd_sd,
                                           max_value = 30, samples = 1)

# simple mapping
report <- adjust_infection_to_report(cases, delay_defs = list(incubation_def, delay_def))
print(report)

# mapping with a weekly reporting effect
report_weekly <- adjust_infection_to_report(
                      cases, delay_defs = list(incubation_def, delay_def),
                      reporting_effect = c(1.1, rep(1, 4), 0.95, 0.95))
print(report_weekly)

# map using a deterministic median shift for both delays
report_median <- adjust_infection_to_report(cases, delay_defs = list(incubation_def, delay_def),
                                            type = "median")
print(report_median)

# map with a weekly reporting effect and stochastic reporting model
report_stochastic <- adjust_infection_to_report(
                      cases, delay_defs = list(incubation_def, delay_def),
                      reporting_effect = c(1.1, rep(1, 4), 0.95, 0.95),
                      reporting_model = function(n) {
                      out <- suppressWarnings(rnbinom(length(n), as.integer(n), 0.5))
                      out <- ifelse(is.na(out), 0, out)
                      })
print(report_stochastic)
```

`allocate_delays` *Allocate Delays into Required Stan Format*

Description

Allocate Delays into Required Stan Format

Usage

```
allocate_delays(delay_var, no_delays)
```

Arguments

`delay_var`	List of numeric delays
`no_delays`	Numeric, number of delays

Value

A numeric array

`bootstrapped_dist_fit` *Fit a Subsampled Bootstrap to Integer Values and Summarise Distribution Parameters*

Description

Fit a Subsampled Bootstrap to Integer Values and Summarise Distribution Parameters

Usage

```
bootstrapped_dist_fit(
  values,
  dist = "lognormal",
  samples = 2000,
  bootstraps = 10,
  bootstrap_samples = 250,
  max_value,
  verbose = FALSE
)
```

Arguments

`values`	Numeric vector of integer values.
`dist`	Character string, which distribution to fit. Defaults to lognormal (`"lognormal"`) but gamma (`"gamma"`) is also supported.
`samples`	Numeric, number of samples to take overall from the bootstrapped posteriors.
`bootstraps`	Numeric, defaults to 1. The number of bootstrap samples (with replacement) of the delay distribution to take.
`bootstrap_samples`	Numeric, defaults to 100. The number of samples to take in each bootstrap. When the sample size of the supplied delay distribution is less than 100 this is used instead.
`max_value`	Numeric, defaults to the maximum value in the observed data. Maximum delay to allow (added to output but does impact fitting).
`verbose`	Logical, defaults to `FALSE`. Should progress messages be printed

Value

A list summarising the bootstrapped distribution

Examples

```
# lognormal
delays <- rlnorm(500, log(5), 1)
out <- bootstrapped_dist_fit(delays, samples = 1000, bootstraps = 10,
                             dist = "lognormal")
out
```

calc_CrI *Calculate Credible Interval*

Description

Adds symmetric a credible interval based on quantiles.

Usage

```
calc_CrI(samples, summarise_by = c(), CrI = 0.9)
```

Arguments

samples	A data.table containing at least a value variable
summarise_by	A character vector of variables to group by.
CrI	Numeric between 0 and 1. The credible interval for which to return values. Defaults to 0.9.

Value

A data.table containing the upper and lower bounds for the specified credible interval

Examples

```
samples <- data.frame(value = 1:10, type = "car")
# add 90% credible interval
calc_CrI(samples)
# add 90% credible interval grouped by type
calc_CrI(samples, summarise_by = "type")
```

`calc_CrIs` *Calculate Credible Intervals*

Description

Adds symmetric credible intervals based on quantiles.

Usage

```
calc_CrIs(samples, summarise_by = c(), CrIs = c(0.2, 0.5, 0.9))
```

Arguments

`samples`	A data.table containing at least a value variable
`summarise_by`	A character vector of variables to group by.
`CrIs`	Numeric vector of credible intervals to calculate.

Value

A data.table containing the `summarise_by` variables and the specified lower and upper credible intervals

Examples

```
samples <- data.frame(value = 1:10, type = "car")
# add credible intervals
calc_CrIs(samples)
# add 90% credible interval grouped by type
calc_CrIs(samples, summarise_by = "type")
```

`calc_summary_measures` *Calculate All Summary Measures*

Description

Calculate summary statistics and credible intervals from a data frame by group.

Usage

```
calc_summary_measures(
  samples,
  summarise_by = NULL,
  order_by = NULL,
  CrIs = c(0.2, 0.5, 0.9)
)
```

Arguments

`samples`	A data.table containing at least a value variable
`summarise_by`	A character vector of variables to group by.
`order_by`	A character vector of parameters to order by, defaults to all `summarise_by` variables.
`CrIs`	Numeric vector of credible intervals to calculate.

Value

A data.table containing summary statistics by group.

Examples

```
samples <- data.frame(value = 1:10, type = "car")
# default
calc_summary_measures(samples)
#  by type
calc_summary_measures(samples, summarise_by = "type")
```

`calc_summary_stats` *Calculate Summary Statistics*

Description

Calculate summary statistics from a data frame by group. Currently supports the mean, median and standard deviation.

Usage

```
calc_summary_stats(samples, summarise_by = c())
```

Arguments

`samples`	A data.table containing at least a value variable
`summarise_by`	A character vector of variables to group by.

Value

A data.table containing the upper and lower bounds for the specified credible interval

Examples

```
samples <- data.frame(value = 1:10, type = "car")
# default
calc_summary_stats(samples)
#  by type
calc_summary_stats(samples, summarise_by = "type")
```

clean_nowcasts *Clean Nowcasts for a Supplied Date*

Description

This function removes nowcasts in the format produced by `EpiNow2` from a target directory for the date supplied.

Usage

```
clean_nowcasts(date = NULL, nowcast_dir = ".")
```

Arguments

`date`	Date object. Defaults to today's date
`nowcast_dir`	Character string giving the filepath to the nowcast results directory. Defaults to the current directory.

clean_regions *Clean Regions*

Description

Clean Regions

Usage

```
clean_regions(reported_cases, non_zero_points)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date), and region (`region`).
`non_zero_points`	Numeric, the minimum number of time points with non-zero cases in a region required for that region to be evaluated. Defaults to 2.

Value

A dataframe of cleaned regional data

`construct_output` *Construct Output*

Description

Construct Output

Usage

```
construct_output(
  estimates,
  forecast = NULL,
  estimated_reported_cases,
  plots = NULL,
  summary = NULL,
  samples = TRUE
)
```

Arguments

`estimates`	List of data frames as output by `estimate_infections`
`forecast`	A list of data frames as output by `forecast_infections`
`estimated_reported_cases`	A list of dataframes as produced by `estimates_by_report_date`.
`plots`	A list of plots as produced by `report_plots`
`summary`	A list of summary output as produced by `report_summary`
`samples`	Logical, defaults to TRUE. Should samples be saved

Value

A list of output as returned by `epinow`

`copy_results_to_latest` *Copy Results From Dated Folder to Latest*

Description

Copy Results From Dated Folder to Latest

Usage

```
copy_results_to_latest(target_folder = NULL, latest_folder = NULL)
```

Arguments

`target_folder`	Character string specifying where to save results (will create if not present).
`latest_folder`	Character string containing the path to the latest target folder. As produced by `setup_target_folder`.

`country_map` *Generate a country map for a single variable.*

Description

This general purpose function can be used to generate a country map for a single variable. It has few defaults but the data supplied must contain a `region_code` variable for linking to mapping data. This function requires the installation of the `rnaturalearth` package.

Usage

```
country_map(
  data = NULL,
  country = NULL,
  variable = NULL,
  variable_label = NULL,
  trans = "identity",
  fill_labels = NULL,
  scale_fill = NULL,
  region_col_ne = "provnum_ne",
  ...
)
```

Arguments

`data`	Dataframe containing variables to be mapped. Must contain a `region_code` variable.
`country`	Character string indicating the name of the country to be mapped.
`variable`	A character string indicating the variable to map data for. This must be supplied.
`variable_label`	A character string indicating the variable label to use. If not supplied then the underlying variable name is used.
`trans`	A character string specifying the transform to use on the specified metric. Defaults to no transform ("identity"). Other options include log scaling ("log") and log base 10 scaling ("log10"). For a complete list of options see `ggplot2::continous_scale`.
`fill_labels`	A function to use to allocate legend labels. An example (used below) is `scales::percent`, which can be used for percentage data.
`scale_fill`	Function to use for scaling the fill. Defaults to a custom `ggplot2::scale_fill_manual`, which expects the possible values to be "Increasing", "Likely increasing", "Likely decreasing", "Decreasing" or "Unsure".

`region_col_ne`	Character string indicating the name of a column in the data returned by `rnaturalearth::ne_states()` that `data$region_code` corresponds to. Possibilities include `provnum_ne`, `name`, `fips` and others and will depend on which country you are mapping.
`...`	Additional arguments passed to the `scale_fill` function

Value

A `ggplot2` object containing a country map.

Examples

```
if(requireNamespace("rnaturalearth") & requireNamespace("scales")){
# Example 1
# if you know the provnum_ne codes you can use them directly
eg_data <- data.table::data.table(variable = c("Increasing",
                                               "Decreasing",
                                               "Unsure",
                                               "Likely decreasing",
                                               "Likely increasing"),
                                  region_code = c(5, 7, 6, 8, 9))
# make variable a factor so the ordering is sensible
eg_data$variable <- factor(eg_data$variable, levels = c("Decreasing", "Likely decreasing",
                                                        "Unsure", "Likely increasing",
                                                        "Increasing"))

country_map(data = eg_data, country = "Australia", variable = "variable")


# Example 2
# sometimes it will be more convenient to join your data by name than provnum_ne code:
us_data <- data.table::data.table(variable = c("Increasing",
                                               "Decreasing",
                                               "Unsure",
                                               "Likely decreasing",
                                               "Likely increasing"),
                                  region_code = c("California",
                                                  "Texas",
                                                  "Florida",
                                                  "Arizona",
                                                  "New York"))
# make variable a factor so the ordering is sensible in the legend
us_data$variable <- factor(us_data$variable, levels = c("Decreasing", "Likely decreasing",
                                                        "Unsure", "Likely increasing",
                                                        "Increasing"))

country_map(data = us_data, country = "United States of America",
            variable = "variable", region_col_ne = "name")
}
```

`create_clean_reported_cases` *Create Clean Reported Cases*

Description

Create Clean Reported Cases

Usage

```
create_clean_reported_cases(reported_cases, horizon, zero_threshold = 50)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.
`zero_threshold`	Numeric defaults to 50. Indicates if detected zero cases are meaningful by using a threshold of 50 cases on average over the last 7 days. If the average is above this thresold then the zero is replaced with the

Value

A cleaned data frame of reported cases

`create_future_rt` *Construct the Required Future Rt assumption*

Description

Construct the Required Future Rt assumption

Usage

```
create_future_rt(future_rt = "project", delay = 0)
```

Arguments

`future_rt`	A character string or integer. This argument indicates how to set future Rt values. Supported options are to project using the Rt model ("project"), to use the latest estimate based on partial data ("latest"), to use the latest estimate based on data that is over 50% complete ("estimate"). If an integer is supplied then the Rt estimate from this many days into the future (or past if negative) past will be used forwards in time.
`delay`	Numeric mean delay

Value

A list containing a logical called fixed and an integer called from

---

`create_initial_conditions` *Create Initial Conditions Generating Function*

Description

Create Initial Conditions Generating Function

Usage

```
create_initial_conditions(data, delays, rt_prior, generation_time, mean_shift)
```

Arguments

`data`	A list of data as produced by `create_stan_data`.
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.
`rt_prior`	A list contain the mean and standard deviation (sd) of the lognormally distributed prior for Rt. By default this is assumed to be mean 1 with a standard deviation of 1 (note in model these will be mapped to log space). To infer infections only using non-parametric backcalculation set this to `list()`.
`generation_time`	A list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the generation time (assuming a gamma distribution).
`mean_shift`	Numeric, mean delay shift

Value

An initial condition generating function

`create_shifted_cases` *Create a Data Frame of Mean Delay Shifted Cases*

Description

Create a Data Frame of Mean Delay Shifted Cases

Usage

```
create_shifted_cases(
  reported_cases,
  mean_shift,
  prior_smoothing_window,
  horizon
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`mean_shift`	Numeric, mean delay shift
`prior_smoothing_window`	Numeric defaults to 7. The number of days over which to take a rolling average for the prior based on reported cases.
`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.

Value

A dataframe for shifted reported cases

`create_stan_args` *Create a List of Stan Arguments*

Description

Generates a list of arguments as required by `rstan::sampling` (when method = "exact) or `rstan::vb` (when method = "approximate). See `create_stan_args()` for the defaults and the relevant `rstan` functions for additional options.

Usage

```
create_stan_args(
  model,
  data = NULL,
  init = "random",
  samples = 1000,
  stan_args = NULL,
  method = "exact",
  verbose = FALSE
)
```

Arguments

`model`	A stan model object, defaults to packaged model if not supplied.
`data`	A list of stan data as created by `create_stan_data`
`init`	Initial conditions passed to `rstan`. Defaults to "random" but can also be a function (as supplied by `create_intitial_conditions`).
`samples`	Numeric, defaults to 1000. The overall number of posterior samples to return (Note: not the number of samples per chain as is the default in stan).
`stan_args`	A list of stan arguments to be passed to `rstan::sampling` or `rstan::vb` (when using the "exact" or "approximate" method). For `method = approximate` an additional argument `trials` indicates the number of attempts to make using variational inference before returning an error (as stochastic failure is possible). The default for this is 5.
`method`	A character string defaults to "exact". Also accepts "approximate". Indicates the fitting method to be used this can either be "exact" (NUTs sampling) or "approximate" (variational inference). The exact approach returns samples from the posterior whilst the approximate method returns approximate samples. The approximate method is likely to return results several order of magnitudes faster than the exact method.
`verbose`	Logical, defaults to `FALSE`. Should verbose progress messages be returned.

Value

A list of stan arguments

Examples

```
# default settings
create_stan_args()

# approximate settings
create_stan_args(method = "approximate")
# increasing warmup
create_stan_args(stan_args = list(warmup = 1000))
```

create_stan_data *Create Stan Data Required for estimate_infections*

Description

Create Stan Data Required for estimate_infections

Usage

```
create_stan_data(
  reported_cases,
  shifted_reported_cases,
  horizon,
  no_delays,
  mean_shift,
  generation_time,
  rt_prior,
  estimate_rt,
  week_effect,
  stationary,
  fixed,
  break_no,
  future_rt,
  gp,
  family,
  delays
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`shifted_reported_cases`	A dataframe of delay shifted reported cases
`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.
`no_delays`	Numeric, number of delays
`mean_shift`	Numeric, mean delay shift
`generation_time`	A list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the generation time (assuming a gamma distribution).
`rt_prior`	A list contain the mean and standard deviation (sd) of the lognormally distributed prior for Rt. By default this is assumed to be mean 1 with a standard deviation of 1 (note in model these will be mapped to log space). To infer infections only using non-parametric backcalculation set this to `list()`.

`estimate_rt`	Logical, should Rt be estimated.
`week_effect`	Logical, defaults TRUE. Should weekly reporting effects be estimated.
`stationary`	Logical, defaults to FALSE. Should Rt be estimated with a global mean. When estimating Rt this should substantially improve run times but will revert to the global average for real time and forecasted estimates. This setting is most appropriate when estimating historic Rt or when combined with breakpoints.
`fixed`	Logical, should the gaussian process be used for non-parametric change over time.
`break_no`	Numeric, number of breakpoints
`future_rt`	A character string or integer. This argument indicates how to set future Rt values. Supported options are to project using the Rt model ("project"), to use the latest estimate based on partial data ("latest"), to use the latest estimate based on data that is over 50% complete ("estimate"). If an integer is supplied then the Rt estimate from this many days into the future (or past if negative) past will be used forwards in time.
`gp`	List controlling the Gaussian process approximation if set to `list()` then `Rt` is assumed to be constant unless other settings introduce variation. If set must contain the `basis_prop` (number of basis functions based on scaling the time points) which defaults to 0.3 and must be between 0 and 1 (increasing this increases the accuracy of the approximation and the cost of additional compute. Must also contain the `boundary_scale` (multiplied by half the range of the input time series). Increasing this increases the accuracy of the approximation at the cost of additional compute. See here: https://arxiv.org/abs/2004.11408 for more information on setting these parameters. Must also contain the `lengthscale_alpha` and `lengthscale_beta`. These tune the prior of the lengthscale. Principled values can be obtained using `tune_inv_gamma` which optimises based on the desired truncation (which should be based on the scale of the observed data). The default is tuned to have 98% of the density of the distribution between 2 and 21 days. Finally the list must contain `alpha_sd` the standard deviation for the alpha parameter of the gaussian process. This defaults to 0.1.
`family`	A character string indicating the reporting model to use. Defaults to negative binomial ("negbin") with poisson ("poisson") also supported.
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.

Value

A list of stan data

dist_fit *Fit an Integer Adjusted Exponential, Gamma or Lognormal distributions*

Description

Fit an Integer Adjusted Exponential, Gamma or Lognormal distributions

Usage

```
dist_fit(
  values = NULL,
  samples = NULL,
  cores = 1,
  chains = 2,
  dist = "exp",
  verbose = FALSE
)
```

Arguments

values	Numeric vector of values
samples	Numeric, number of samples to take
cores	Numeric, defaults to 1. Number of CPU cores to use (no effect if greater than the number of chains).
chains	Numeric, defaults to 2. Number of MCMC chains to use. More is better with the minimum being two.
dist	Character string, which distribution to fit. Defaults to exponential (`"exp"`) but gamma (`"gamma"`) and lognormal (`"lognorma"`) are also supported.
verbose	Logical, defaults to FALSE. Should verbose progress messages be printed.

Value

A `stan` fit of an interval censored distribution

Examples

```
# integer adjusted exponential model
dist_fit(rexp(1:100, 2), samples = 1000, dist = "exp",
         cores = ifelse(interactive(), 4, 1), verbose = TRUE)


# integer adjusted gamma model
dist_fit(rgamma(1:100, 5, 5), samples = 1000, dist = "gamma",
         cores = ifelse(interactive(), 4, 1), verbose = TRUE)
```

```
# integer adjusted lognormal model
dist_fit(rlnorm(1:100, log(5), 0.2), samples = 1000, dist = "lognormal",
         cores = ifelse(interactive(), 4, 1), verbose = TRUE)
```

`dist_skel` *Distribution Skeleton*

Description

This function acts as a skeleton for a truncated distribution defined by model type, maximum value and model parameters. It is designed to be used with the output from `get_dist`.

Usage

```
dist_skel(n, dist = FALSE, cum = TRUE, model, params, max_value = 120)
```

Arguments

`n`	Numeric vector, number of samples to take (or days for the probability density).
`dist`	Logical, defaults to `FALSE`. Should the probability density be returned rather than a number of samples.
`cum`	Logical, defaults to `TRUE`. If `dist = TRUE` should the returned distribution be cumulative.
`model`	Character string, defining the model to be used. Supported options are exponential ("exp"), gamma ("gamma"), and log normal ("lognorm")
`params`	A list of parameters values (by name) required for each model. For the exponential model this is a rate parameter and for the gamma model this is alpha and beta.
`max_value`	Numeric, the maximum value to allow. Defaults to 120. Samples outside of this range are resampled.

Value

A vector of samples or a probability distribution.

Examples

```
## Exponential model
# sample
dist_skel(10, model = "exp", params = list(rate = 1))

# cumulative prob density
dist_skel(1:10, model = "exp", dist = TRUE, params = list(rate = 1))

# probability density
```

```
dist_skel(1:10, model = "exp", dist = TRUE,
          cum = FALSE, params = list(rate = 1))

## Gamma model
# sample
dist_skel(10, model = "gamma", params = list(alpha = 1, beta = 2))

# cumulative prob density
dist_skel(0:10, model = "gamma", dist = TRUE,
          params = list(alpha = 1, beta = 2))

# probability density
dist_skel(0:10, model = "gamma", dist = TRUE,
          cum = FALSE, params = list(alpha = 2, beta = 2))

## Log normal model
# sample
dist_skel(10, model = "lognorm", params = list(mean = log(5), sd = log(2)))

# cumulative prob density
dist_skel(0:10, model = "lognorm", dist = TRUE,
          params = list(mean = log(5), sd = log(2)))

# probability density
dist_skel(0:10, model = "lognorm", dist = TRUE, cum = FALSE,
          params = list(mean = log(5), sd = log(2)))
```

epinow *Real-time Rt Estimation, Forecasting and Reporting*

Description

This function wraps the functionality of `estimate_infections` and `forecast_infections` in order to estimate Rt and cases by date of infection, forecast into these infections into the future. It also contains additional functionality to convert forecasts to date of report and produce summary output useful for reporting results and interpreting them.

Usage

```
epinow(
  reported_cases,
  samples = 1000,
  horizon = 7,
  generation_time,
  delays = list(),
  CrIs = c(0.2, 0.5, 0.9),
  return_output = FALSE,
  output = c("samples", "plots", "latest"),
  target_folder = NULL,
  target_date,
  forecast_args = NULL,
  logs = tempdir(),
  id = "epinow",
  verbose = FALSE,
  ...
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`samples`	Numeric, defaults to 1000. Number of samples post warmup.
`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.
`generation_time`	A list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the generation time (assuming a gamma distribution).
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.
`CrIs`	Numeric vector of credible intervals to calculate.
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.
`output`	A character vector of optional output to return. Supported options are samples ("samples"), plots ("plots"), the run time ("timing"), copying the dated folder into a latest folder (if `target_folder` is not null • set using "latest"), and the stan fit ("fit"). The default is to return samples and plots alongside summarised estimates and summary statistics. This argument uses partial matching so for example passing "sam" will lead to samples being reported.
`target_folder`	Character string specifying where to save results (will create if not present).
`target_date`	Date, defaults to maximum found in the data if not specified.
`forecast_args`	A list of arguments to pass to `forecast_infections`. Unless at a minimum a `forecast_model` is passed tin his list then `forecast_infections` will be bypassed.
`logs`	Character path indicating the target folder in which to store log information. Defaults to the temporary directory if not specified. Default logging can be disabled if `logs` is set to NULL. If specifying a custom logging setup then the code for `setup_default_logging` and the `setup_logging` function are a sensible place to start.

id	A character string used to assign logging information on error. Used by `regional_epinow` to assign `epinow` errors to regions.
verbose	Logical, defaults to `FALSE`. Should verbose debug progress messages be printed. Corresponds to the "DEBUG" level from `futile.logger`. See `setup_logging` for more detailed logging options.
...	Additional arguments passed to `estimate_infections`. See that functions documentation for options.

Value

A list of output from estimate_infections, forecast_infections, report_cases, and report_summary.

Examples

```
# construct example distributions
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# example case data
reported_cases <- EpiNow2::example_confirmed[1:40]

# estimate Rt and nowcast/forecast cases by date of infection
out <- epinow(reported_cases = reported_cases, generation_time = generation_time,
              delays = list(incubation_period, reporting_delay),
              stan_args = list(cores = ifelse(interactive(), 4, 1),
              control = list(adapt_delta = 0.95)))
out

# optional forecasting using EpiSoon plug-in
if(requireNamespace("EpiSoon")){
   if(requireNamespace("forecastHybrid")){
   # report Rt along with forecasts
   out <- epinow(reported_cases = reported_cases, samples = 200,
                 generation_time = generation_time,
                 delays = list(incubation_period, reporting_delay),
                 forecast_args = list(
                     forecast_model = function(y, ...){
                     EpiSoon::forecastHybrid_model(
                          y = y[max(1, length(y) - 21):length(y)],
                          model_params = list(models = "aefz", weights = "equal"),
                          forecast_params = list(PI.combination = "mean"), ...)}
                          ),
                 stan_args = list(warmup = 200, cores = ifelse(interactive(), 4, 1)))
    out
   }
}
```

estimates_by_report_date

Estimate Cases by Report Date

Description

Estimate Cases by Report Date

Usage

```
estimates_by_report_date(
  estimates,
  forecast,
  delays,
  CrIs = c(0.2, 0.5, 0.9),
  target_folder = NULL,
  samples = TRUE
)
```

Arguments

`estimates`	List of data frames as output by `estimate_infections`
`forecast`	A list of data frames as output by `forecast_infections`
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.
`CrIs`	Numeric vector of credible intervals to calculate.
`target_folder`	Character string specifying where to save results (will create if not present).
`samples`	Logical, defaults to TRUE. Should samples be saved

Value

A list of samples and summarised estimates of estimated cases by date of report `estimate_infections` *Estimate Infections, the Time-Varying Reproduction Number and the Rate of Growth*

Description

This function uses a non-parametric approach to reconstruct cases by date of infection from reported cases. It can optionally then estimate the time-varying reproduction number and the rate of growth.

Usage

```
estimate_infections(
  reported_cases,
  model = NULL,
  samples = 1000,
  stan_args = NULL,
  method = "exact",
  family = "negbin",
  generation_time,
  CrIs = c(0.2, 0.5, 0.9),
  delays = list(),
  horizon = 7,
  gp = list(basis_prop = 0.3, boundary_scale = 2, lengthscale_alpha = 4.5,
    lengthscale_beta = 21.5, alpha_sd = 0.1),
  rt_prior = list(mean = 1, sd = 1),
  week_effect = TRUE,
  use_breakpoints = TRUE,
  stationary = FALSE,
  future_rt = "project",
  burn_in = 0,
  prior_smoothing_window = 7,
  future = FALSE,
  max_execution_time = Inf,
  return_fit = FALSE,
  verbose = FALSE
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`model`	A compiled stan model. By default uses the internal package model.
`samples`	Numeric, defaults to 1000. Number of samples post warmup.
`stan_args`	A list of stan arguments to be passed to `rstan::sampling` or `rstan::vb` (when using the "exact" or "approximate" method). For `method = approximate` an additional argument `trials` indicates the number of attempts to make using

	variational inference before returning an error (as stochastic failure is possible). The default for this is 5.
`method`	A character string defaults to "exact". Also accepts "approximate". Indicates the fitting method to be used this can either be "exact" (NUTs sampling) or "approximate" (variational inference). The exact approach returns samples from the posterior whilst the approximate method returns approximate samples. The approximate method is likely to return results several order of magnitudes faster than the exact method.
`family`	A character string indicating the reporting model to use. Defaults to negative binomial ("negbin") with poisson ("poisson") also supported.
`generation_time`	A list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the generation time (assuming a gamma distribution).
`CrIs`	Numeric vector of credible intervals to calculate.
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.
`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.
`gp`	List controlling the Gaussian process approximation if set to `list()` then `Rt` is assumed to be constant unless other settings introduce variation. If set must contain the `basis_prop` (number of basis functions based on scaling the time points) which defaults to 0.3 and must be between 0 and 1 (increasing this increases the accuracy of the approximation and the cost of additional compute. Must also contain the `boundary_scale` (multiplied by half the range of the input time series). Increasing this increases the accuracy of the approximation at the cost of additional compute. See here: https://arxiv.org/abs/2004.11408 for more information on setting these parameters. Must also contain the `lengthscale_alpha` and `lengthscale_beta`. These tune the prior of the lengthscale. Principled values can be obtained using `tune_inv_gamma` which optimises based on the desired truncation (which should be based on the scale of the observed data). The default is tuned to have 98% of the density of the distribution between 2 and 21 days. Finally the list must contain `alpha_sd` the standard deviation for the alpha parameter of the gaussian process. This defaults to 0.1.
`rt_prior`	A list contain the mean and standard deviation (sd) of the lognormally distributed prior for Rt. By default this is assumed to be mean 1 with a standard deviation of 1 (note in model these will be mapped to log space). To infer infections only using non-parametric backcalculation set this to `list()`.
`week_effect`	Logical, defaults TRUE. Should weekly reporting effects be estimated.
`use_breakpoints`	Logical, defaults to TRUE but only active if a `breakpoint` variable is present in the input data. Breakpoints should be defined as 1 if present and otherwise 0. By default breakpoints are fit jointly with a global non-parametric effect and so

	represent a conservative estimate of breakpoint changes. To specify a random walk define breakpoints every n days (so every 7 days for a weekly random walk) and disable the gaussian process using `gp = list()`.
`stationary`	Logical, defaults to FALSE. Should Rt be estimated with a global mean. When estimating Rt this should substantially improve run times but will revert to the global average for real time and forecasted estimates. This setting is most appropriate when estimating historic Rt or when combined with breakpoints.
`future_rt`	A character string or integer. This argument indicates how to set future Rt values. Supported options are to project using the Rt model ("project"), to use the latest estimate based on partial data ("latest"), to use the latest estimate based on data that is over 50% complete ("estimate"). If an integer is supplied then the Rt estimate from this many days into the future (or past if negative) past will be used forwards in time.
`burn_in`	Numeric, defaults to 0. The number of initial Rt estimates to discard. This argument may be used to reduce spurious findings when running `estimate_infections` on a partial timeseries (as the earliest estimates will not be informed by all cases that occurred only those supplied to `estimate_infections`). The combined delays used will inform the appropriate length of this burn in but 7 days is likely a sensible starting point.
`prior_smoothing_window`	Numeric defaults to 7. The number of days over which to take a rolling average for the prior based on reported cases.
`future`	Logical, defaults to `FALSE`. Should stan chains be run in parallel using `future`. This allows users to have chains fail gracefully (i.e when combined with `max_execution_time`). Should be combined with a call to `future::plan`
`max_execution_time`	Numeric, defaults to Inf (seconds). If set will kill off processing of each chain if not finished within the specified timeout. When more than 2 chains finish successfully estimates will still be returned. If less than 2 chains return within the allowed time then estimation will fail with an informative error.
`return_fit`	Logical, defaults to FALSE. Should the fitted stan model be returned.
`verbose`	Logical, defaults to `FALSE`. Should verbose debug progress messages be printed. Corresponds to the "DEBUG" level from `futile.logger`. See `setup_logging` for more detailed logging options.

Examples

```
# get example case counts
reported_cases <- EpiNow2::example_confirmed[1:50]

# add a dummy breakpoint (used only when optionally estimating breakpoints)
reported_cases_bp <- data.table::copy(reported_cases)[,
              breakpoint := data.table::fifelse(date == as.Date("2020-03-16"), 1, 0)]
# set up example generation time
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
# set delays between infection and case report
```

```
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- list(mean = log(5), mean_sd = log(2),
                        sd = log(2), sd_sd = log(1.5), max = 30)

# Note: all examples below have been tuned to reduce the runtimes of examples
# these settings are not suggesed for real world use.
# run model with default setting
def <- estimate_infections(reported_cases, generation_time = generation_time,
                           delays = list(incubation_period, reporting_delay),
                        stan_args = list(warmup = 200, control = list(adapt_delta = 0.8),
                                            cores = ifelse(interactive(), 4, 1)))

plots <- report_plots(summarised_estimates = def$summarised, reported = reported_cases)
plots$summary

# run the model using the approximate method (variational inference)
approx <- estimate_infections(reported_cases, generation_time = generation_time,
                              delays = list(incubation_period, reporting_delay),
                              method = "approximate")

plots <- report_plots(summarised_estimates = approx$summarised, reported = reported_cases)
plots$summary

# run the model with default settings using the future backend
## (combine with a call to future::plan to make this parallel).
def_future <- estimate_infections(reported_cases, generation_time = generation_time,
                                  delays = list(incubation_period, reporting_delay),
                                  stan_args = list(warmup = 200,
                                                   control = list(adapt_delta = 0.9),
                                                   cores = ifelse(interactive(), 4, 1)))

plots <- report_plots(summarised_estimates = def_future$summarised, reported = reported_cases)
plots$summary

# run model with Rt fixed into the future using the latest estimate
fixed_rt <- estimate_infections(reported_cases, generation_time = generation_time,
                                delays = list(incubation_period, reporting_delay),
                                stan_args = list(warmup = 200,
                                                 control = list(adapt_delta = 0.9),
                                                 cores = ifelse(interactive(), 4, 1)),
                                future_rt = "latest")

plots <- report_plots(summarised_estimates = fixed_rt$summarised, reported = reported_cases)
plots$summary

# run the model with default settings on a later snapshot of
# data (use burn_in here to remove the first week of estimates that may
# be impacted by this most).
snapshot_cases <- EpiNow2::example_confirmed[80:130]
snapshot <- estimate_infections(reported_cases, generation_time = generation_time,
                                delays = list(incubation_period, reporting_delay),
                                stan_args = list(warmup = 200,
                                                 control = list(adapt_delta = 0.9),
                                                 cores = ifelse(interactive(), 4, 1)),
                                burn_in = 7)

plots <- report_plots(summarised_estimates = snapshot$summarised, reported = snapshot_cases)
plots$summary

# run model with stationary Rt assumption (likely to provide biased real-time estimates)
stat <- estimate_infections(reported_cases, generation_time = generation_time,
                            delays = list(incubation_period, reporting_delay),
                       stan_args = list(warmup = 200, cores = ifelse(interactive(), 4, 1),
                                             control = list(adapt_delta = 0.9)),
                            stationary = TRUE)

plots <- report_plots(summarised_estimates = stat$summarised, reported = reported_cases)
plots$summary

# run model with fixed Rt assumption
fixed <- estimate_infections(reported_cases, generation_time = generation_time,
                             delays = list(incubation_period, reporting_delay),
                       stan_args = list(warmup = 200, cores = ifelse(interactive(), 4, 1),
                                              control = list(adapt_delta = 0.9)),
                             gp = list())

plots <- report_plots(summarised_estimates = fixed$summarised, reported = reported_cases)
plots$summary

# run model with no delays
no_delay <- estimate_infections(reported_cases, generation_time = generation_time,
                                stan_args = list(warmup = 200,
                                                 cores = ifelse(interactive(), 4, 1),
                                                 control = list(adapt_delta = 0.9)))

plots <- report_plots(summarised_estimates = no_delay$summarised, reported = reported_cases)
plots$summary

# run model with breakpoints
bkp <- estimate_infections(reported_cases_bp, generation_time = generation_time,
                           delays = list(incubation_period, reporting_delay),
                           stan_args = list(warmup = 200,
                                            cores = ifelse(interactive(), 4, 1),
                                            control = list(adapt_delta = 0.9)))

plots <- report_plots(summarised_estimates = bkp$summarised, reported = reported_cases)
plots$summary

# run model with breakpoints but with constrained non-linear change over time
# rhis formulation may increase the apparent effect of the breakpoint but needs to be tested using
# model fit criteria (i.e LFO).
cbkp <- estimate_infections(reported_cases_bp, generation_time = generation_time,
                           delays = list(incubation_period, reporting_delay),
                           gp = list(basis_prop = 0.3, boundary_scale = 2,
                                     lengthscale_mean = 20, lengthscale_sd = 1),
                           stan_args = list(warmup = 200,
                           cores = ifelse(interactive(), 4, 1),
                           control = list(adapt_delta = 0.9)))

plots <- report_plots(summarised_estimates = cbkp$summarised, reported = reported_cases)
plots$summary
# breakpoint effect
cbkp$summarised[variable == "breakpoints"]

# run model with breakpoints but otherwise static Rt
# This formulation may increase the apparent effect of the breakpoint but needs to be tested using
# model fit criteria (i.e LFO).
fbkp <- estimate_infections(reported_cases_bp, generation_time = generation_time,
                            delays = list(incubation_period, reporting_delay),
                            stan_args = list(warmup = 200,
                                             cores = ifelse(interactive(), 4, 1),
                                             control = list(adapt_delta = 0.9)),
                            gp = list())

plots <- report_plots(summarised_estimates = fbkp$summarised, reported = reported_cases)
plots$summary
# breakpoint effect
fbkp$summarised[variable == "breakpoints"]

# run model without Rt estimation (just backcalculation)
backcalc <- estimate_infections(reported_cases, generation_time = generation_time,
                               delays = list(incubation_period, reporting_delay),
                               stan_args = list(warmup = 200,
                                                cores = ifelse(interactive(), 4, 1),
                                                control = list(adapt_delta = 0.9)),
                               rt_prior = list())

# plot just infections as report_plots does not support the backcalculation only model
plot_estimates(estimate = backcalc$summarised[variable == "infections"],
               reported = reported_cases, ylab = "Cases")
```

example_confirmed *Example Confirmed Case Data Set*

Description

An example data frame of observed cases

Usage

```
example_confirmed
```

Format

A data frame containing cases reported on each date.

`extract_CrIs` *Extract Credible Intervals Present*

Description

Extract Credible Intervals Present

Usage

```
extract_CrIs(summarised)
```

Arguments

`summarised`	A data frame as processed by `calc_CrIs`

Value

A numeric vector of credible intervals detected in the data frame.

Examples

```
samples <- data.frame(value = 1:10, type = "car")
summarised <- calc_CrIs(samples, summarise_by = "type",
                        CrIs = c(seq(0.05, 0.95, 0.05)))
extract_CrIs(summarised)
```

`extract_parameter` *Extract Samples for a Parameter from a Stan model*

Description

Extract Samples for a Parameter from a Stan model

Usage

```
extract_parameter(param, samples, dates)
```

Arguments

`param`	Character string indicating the parameter to extract
`samples`	Extracted stan model (using `rstan::extract`)
`dates`	A vector identifying the dimensionality of the parameter to extract. Generally this will be a date

Value

A data frame containing the parameter name, date, sample id and sample value

`extract_parameter_samples` *Extract Parameter Samples from a Stan Model*

Description

Extract Parameter Samples from a Stan Model

Usage

```
extract_parameter_samples(stan_fit, data, reported_dates, reported_inf_dates)
```

Arguments

`stan_fit`	A fit Stan model as returned by `rstan:sampling`
`data`	A list of the data supplied to the `rstan::sampling` call.
`reported_dates`	A vector of dates to report estimates for.
`reported_inf_dates`	A vector of dates to report infection estimates for.

Value

A list of dataframes each containing the posterior of a parameter

`extract_static_parameter` *Extract Samples from a Parameter with a Single Dimension*

Description

Extract Samples from a Parameter with a Single Dimension

Usage

```
extract_static_parameter(param, samples)
```

Arguments

`param`	Character string indicating the parameter to extract
`samples`	Extracted stan model (using `rstan::extract`)

Value

A data frame containing the parameter name, sample id and sample value

`fit_model_with_nuts` *Fit a Stan Model using the NUTs sampler*

Description

Fit a Stan Model using the NUTs sampler

Usage

```
fit_model_with_nuts(
  args,
  future = FALSE,
  max_execution_time = Inf,
  verbose = FALSE
)
```

Arguments

`args`	List of stan arguments
`future`	Logical, defaults to `FALSE`. Should `future` be used to run stan chains in parallel.
`max_execution_time`	Numeric, defaults to Inf. What is the maximum execution time per chain in seconds. Results will still be returned as long as at least 2 chains complete successfully within the timelimit.
`verbose`	Logical, defaults to `FALSE`. Should verbose progress information be returned.

Value

A stan model object

`fit_model_with_vb` *Fit a Stan Model using Variational Inference*

Description

Fit a Stan Model using Variational Inference

Usage

```
fit_model_with_vb(args, future = FALSE, verbose = FALSE)
```

Arguments

`args`	List of stan arguments
`future`	Logical, defaults to `FALSE`. Should `future` be used to run stan chains in parallel.
`verbose`	Logical, defaults to `FALSE`. Should verbose progress information be returned.

Value

A stan model object

forecast_infections *Forecast Infections and the Time-Varying Reproduction Number*

Description

This function provides optional tools for forecasting cases and Rt estimates using the timeseries methods (via the EpiSoon package). It requires the Episoon package. Installation instructions for the EpiSoon package are available here.

Usage

```
forecast_infections(
  infections,
  rts,
  gt_mean,
  gt_sd,
  gt_max = 30,
  ensemble_type = "mean",
  forecast_model,
  CrIs = c(0.2, 0.5, 0.9),
  horizon = 14,
  samples = 1000
)
```

Arguments

infections	A data frame of cases by date of infection containing the following variables: date, mean, sd
rts	A data frame of Rt estimates by date of infection containing the following variables: date, mean, sd
gt_mean	Numeric, the mean of the gamma distributed generation time.
gt_sd	Numeric, the standard deviation of the gamma distributed generation time.
gt_max	Numeric, the maximum allowed value of the gamma distributed generation time.
ensemble_type	Character string indicating the type of ensemble to use. By default this is an unweighted ensemble ("mean") with no other types currently supported.
forecast_model	An uninitialised forecast model function to be passed to `EpiSoon::forecast_rt`. Used for forecasting future Rt and case co An example of the required structure is: `function(ss,y){bsts::AddSemilocalLinearTrend(ss,y = y)}`.
CrIs	Numeric vector of credible intervals to calculate.
horizon	Numeric, defaults to 14. The horizon over which to forecast Rts and cases.
samples	Numeric, the number of forecast samples to take.

Value

A list of data.tables. The first entry ("samples") contains raw forecast samples and the second entry ("summarised") contains summarised forecasts.

Examples

```
if(requireNamespace("EpiSoon")){
   if(requireNamespace("forecastHybrid")){
# example case data
reported_cases <- EpiNow2::example_confirmed[1:40]

generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 15)

# estimate Rt and infections from data
out <- estimate_infections(reported_cases, generation_time = generation_time,
                           delays = list(incubation_period, reporting_delay),
                           gp = list(), samples = 100,
                           stan_args = list(cores = ifelse(interactive(), 4, 1),
                                            warmup = 100, chains = 4))

# forecast Rt and infections from estimates
forecast <- forecast_infections(
    infections = out$summarised[variable == "infections"],
    rts = out$summarised[variable == "R"],
     gt_mean = out$summarised[variable == "gt_mean"]$mean,
     gt_sd = out$summarised[variable == "gt_sd"]$mean,
     gt_max = 30,
     forecast_model = function(y, ...){
       EpiSoon::forecastHybrid_model(y = y[max(1, length(y) - 21):length(y)],
       model_params = list(models = "aefz", weights = "equal"),
       forecast_params = list(PI.combination = "mean"), ...)},
     horizon = 14,
     samples = 1000)

forecast$summarised
  }
 }
```

---

format_fit *Format Posterior Samples*

Description

Format Posterior Samples

Usage

```
format_fit(posterior_samples, horizon, shift, burn_in, start_date, CrIs)
```

Arguments

`posterior_samples`	A list of posterior samples as returned by `extract_parameter_samples`
`horizon`	Numeric, forecast horizon
`shift`	Numeric, the shift to apply to estimates
`burn_in`	Numeric, number of days to discard estimates for
`start_date`	Date, earliest date with data
`CrIs`	Numeric vector of credible intervals to calculate.

Value

A list of samples and summarised posterior parameter estimates

---

`gamma_dist_def` *Generate a Gamma Distribution Definition Based on Parameter Estimates*

Description

Generates a distribution definition when only parameter estimates are available for gamma distributed parameters. See `rgamma` for distribution information.

Usage

```
gamma_dist_def(
  shape,
  shape_sd,
  scale,
  scale_sd,
  mean,
  mean_sd,
  sd,
  sd_sd,
  max_value,
  samples
)
```

Arguments

shape	Numeric, shape parameter of the gamma distribution.
shape_sd	Numeric, standard deviation of the shape parameter.
scale	Numeric, scale parameter of the gamma distribution.
scale_sd	Numeric, standard deviation of the scale parameter.
mean	Numeric, log mean parameter of the gamma distribution.
mean_sd	Numeric, standard deviation of the log mean parameter.
sd	Numeric, log sd parameter of the gamma distribution.
sd_sd	Numeric, standard deviation of the log sd parameter.
max_value	Numeric, the maximum value to allow. Defaults to 120. Samples outside of this range are resampled.
samples	Numeric, number of sample distributions to generate.

Value

A data.table defining the distribution as used by `dist_skel`

Examples

```
# using estimated shape and scale
def <- gamma_dist_def(shape = 5.807, shape_sd = 0.2,
               scale = 0.9, scale_sd = 0.05,
               max_value = 20, samples = 10)
print(def)
def$params[[1]]

# using mean and sd
def <- gamma_dist_def(mean = 3, mean_sd = 0.5,
               sd = 3, sd_sd = 0.1,
               max_value = 20, samples = 10)
print(def)
def$params[[1]]
```

`generation_times` *Literature Estimates of Generation Times*

Description

Generation time estimates. See here for details: https://github.com/epiforecasts/EpiNow2/blob/master/data-raw/generation-time.R

Usage

```
generation_times
```

Format

A `data.table` of summarising the distribution

`get_dist` *Get a Literature Distribution*

Description

Get a Literature Distribution

Usage

```
get_dist(data, disease, source, max_value = 30)
```

Arguments

`data`	A `data.table` in the format of `generation_times`.
`disease`	A character string indicating the disease of interest.
`source`	A character string indicating the source of interest.
`max_value`	Numeric, the maximum value to allow. Defaults to 30 days.

Value

A list defining a distribution

Examples

```
get_dist(EpiNow2::generation_times, disease = "SARS-CoV-2", source = "ganyani")
```

`get_generation_time` *Get a Literature Distribution for the Generation Time*

Description

Extracts a literature distribution from `generation_times`

Usage

```
get_generation_time(disease, source, max_value = 30)
```

Arguments

`disease`	A character string indicating the disease of interest.
`source`	A character string indicating the source of interest.
`max_value`	Numeric, the maximum value to allow. Defaults to 30 days.

Value

A list defining a distribution

Examples

```
get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
```

`get_incubation_period` *Get a Literature Distribution for the Incubation Period*

Description

Extracts a literature distribution from `incubation_periods`

Usage

```
get_incubation_period(disease, source, max_value = 30)
```

Arguments

`disease`	A character string indicating the disease of interest.
`source`	A character string indicating the source of interest.
`max_value`	Numeric, the maximum value to allow. Defaults to 30 days.

Value

A list defining a distribution

Examples

```
get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
```

`get_raw_result` *Get a Single Raw Result*

Description

Get a Single Raw Result

Usage

```
get_raw_result(file, region, date, result_dir)
```

Arguments

`file`	Character string giving the result files name.
`region`	Character string giving the region of interest.
`date`	Target date (in the format "yyyy-mm-dd).
`result_dir`	Character string giving the location of the target directory

Value

An R object read in from the targeted .rds file

`get_regional_results` *Get Combined Regional Results*

Description

Get Combined Regional Results

Usage

```
get_regional_results(
  regional_output,
  results_dir,
  date,
  samples = TRUE,
  forecast = FALSE
)
```

Arguments

`regional_output`	A list of output as produced by `regional_epinow` and stored in the `regional` list.
`results_dir`	A character string indicating the folder containing the `EpiNow2` results to extract.
`date`	A Character string (in the format "yyyy-mm-dd") indicating the date to extract data for. Defaults to "latest" which finds the latest results available.
`samples`	Logical, defaults to `TRUE`. Should samples be returned.
`forecast`	Logical, defaults to `FALSE`. Should forecast results be returned.

Value

A list of estimates, forecasts and estimated cases by date of report.

Examples

```
# construct example distributions
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# example case vector from EpiSoon
cases <- EpiNow2::example_confirmed[1:30]
cases <- data.table::rbindlist(list(
  data.table::copy(cases)[, region := "testland"],
  cases[, region := "realland"]))

# run multiregion estimates
regional_out <- regional_epinow(reported_cases = cases,
                                samples = 100,
                                generation_time = generation_time,
                                delays = list(incubation_period, reporting_delay),
                                stan_args = list(warmup = 100,
                                                 cores = ifelse(interactive(), 4, 1)),
                                output = c("regions"))

summary_only <- get_regional_results(regional_out$regional, forecast = FALSE, samples = FALSE)
names(summary_only)

all <- get_regional_results(regional_out$regional, forecast = TRUE)
names(all)
```

`get_regions` *Get Folders with Nowcast Results*

Description

Get Folders with Nowcast Results

Usage

```
get_regions(results_dir)
```

Arguments

`results_dir`	A character string giving the directory in which results are stored (as produced by `regional_rt_pipeline`).

Value

A named character vector containing the results to plot.

get_regions_with_most_reports *Get Regions with Most Reported Cases*

Description

Get Regions with Most Reported Cases

Usage

```
get_regions_with_most_reports(reported_cases, time_window = 7, no_regions = 6)
```

Arguments

reported_cases	A data frame of confirmed cases (confirm) by date (date), and region (region).
time_window	Numeric, number of days to include from latest date in data. Defaults to 7 days.
no_regions	Numeric, number of regions to return. Defaults to 6.

Value

A character vector of regions with the highest reported cases

global_map *Generate a global map for a single variable.*

Description

This general purpose function can be used to generate a global map for a single variable. It has few defaults but the data supplied must contain a country variable for linking to mapping data. This function requires the installation of the rnaturalearth package.

Usage

```
global_map(
  data = NULL,
  variable = NULL,
  variable_label = NULL,
  trans = "identity",
  fill_labels = NULL,
  scale_fill = NULL,
  ...
)
```

Arguments

`data`	Dataframe containing variables to be mapped. Must contain a `country` variable.
`variable`	A character string indicating the variable to map data for. This must be supplied.
`variable_label`	A character string indicating the variable label to use. If not supplied then the underlying variable name is used.
`trans`	A character string specifying the transform to use on the specified metric. Defaults to no transform ("identity"). Other options include log scaling ("log") and log base 10 scaling ("log10"). For a complete list of options see `ggplot2::continous_scale`.
`fill_labels`	A function to use to allocate legend labels. An example (used below) is `scales::percent`, which can be used for percentage data.
`scale_fill`	Function to use for scaling the fill. Defaults to a custom `ggplot2::scale_fill_manual`, which expects the possible values to be "Increasing", "Likely increasing", "Likely decreasing", "Decreasing" or "Unsure".
`...`	Additional arguments passed to the `scale_fill` function

Value

A `ggplot2` object containing a global map.

Examples

```
if(requireNamespace("rnaturalearth") & requireNamespace("scales")){
# Example 1 - categorical data
# If values are "Increasing", "Likely increasing" etc (see ?EpiNow2::theme_map),
# then the default fill scale works
eg_data <- data.table::data.table(variable = c("Increasing",
                                               "Decreasing",
                                               "Unsure",
                                               "Likely decreasing",
                                               "Likely increasing"),
                                  country = c("France",
                                              "Germany",
                                              "United Kingdom",
                                              "Spain",
                                              "Australia") )
# make variable a factor so the ordering is sensible in the legend
eg_data$variable <- factor(eg_data$variable, levels = c("Decreasing", "Likely decreasing",
                                                        "Unsure", "Likely increasing",
                                                        "Increasing"))
global_map(eg_data, variable = "variable", variable_label = "Direction\nof change")


# Example 2 - numeric data
# numeric data requires scale_fill and a global viridis_palette specified
eg_data$second_variable <- runif(nrow(eg_data))
viridis_palette <- "A"
global_map(eg_data, variable = "second_variable", scale_fill = scale_fill_viridis_c)
}
```

growth_to_R *Convert Growth Rates to Reproduction numbers.*

Description

See here for justification.

Usage

```
growth_to_R(r, gamma_mean, gamma_sd)
```

Arguments

`r`	Numeric, rate of growth estimates
`gamma_mean`	Numeric, mean of the gamma distribution
`gamma_sd`	Numeric, standard deviation of the gamma distribution

Value

Numeric vector of reproduction number estimates

Examples

```
growth_to_R(0.2, 4, 1)
```

incubation_periods *Literature Estimates of Incubation Periods*

Description

Incubation period estimates. See here for details: https://github.com/epiforecasts/EpiNow2/blob/master/data-raw/incubation-period.R

Usage

```
incubation_periods
```

Format

A `data.table` of summarising the distribution

lognorm_dist_def *Generate a Log Normal Distribution Definition Based on Parameter Estimates*

Description

Generates a distribution definition when only parameter estimates are available for log normal distributed parameters. See `rlnorm` for distribution information.

Usage

```
lognorm_dist_def(mean, mean_sd, sd, sd_sd, max_value, samples, to_log = FALSE)
```

Arguments

`mean`	Numeric, log mean parameter of the gamma distribution.
`mean_sd`	Numeric, standard deviation of the log mean parameter.
`sd`	Numeric, log sd parameter of the gamma distribution.
`sd_sd`	Numeric, standard deviation of the log sd parameter.
`max_value`	Numeric, the maximum value to allow. Defaults to 120. Samples outside of this range are resampled.
`samples`	Numeric, number of sample distributions to generate.
`to_log`	Logical, should parameters be logged before use.

Value

A data.table defining the distribution as used by `dist_skel`

Examples

```
def <- lognorm_dist_def(mean = 1.621, mean_sd = 0.0640,
                        sd = 0.418, sd_sd = 0.0691,
                        max_value = 20, samples = 10)
print(def)
def$params[[1]]

def <- lognorm_dist_def(mean = 5, mean_sd = 1,
                        sd = 3, sd_sd = 1,
                        max_value = 20, samples = 10,
                        to_log = TRUE)
print(def)
def$params[[1]]
```

`make_conf` *Format Credible Intervals*

Description

Format Credible Intervals

Usage

```
make_conf(value, CrI = 90, reverse = FALSE)
```

Arguments

`value`	List of value to map into a string. Requires, `point`, `lower`, and `upper`.
`CrI`	Numeric, credible interval to report. Defaults to 90
`reverse`	Logical, defaults to FALSE. Should the reported credible interval be switched.

Value

A character vector formatted for reporting

Examples

```
value <- list(median = 2, lower_90 = 1, upper_90 = 3)
make_conf(value)
```

`map_prob_change` *Categorise the Probability of Change for Rt*

Description

Categorises a numeric variable into "Increasing" (< 0.05), "Likely increasing" (<0.2), "Unsure" (< 0.8), "Likely decreasing" (< 0.95), "Decreasing" (<= 1)

Usage

```
map_prob_change(var)
```

Arguments

`var`	Numeric variable to be categorised

Value

A character variable.

Examples

```
var <- seq(0.01, 1, 0.01)
var

map_prob_change(var)
```

---

match_output_arguments

Match Input Output Arguments with Supported Options

Description

Match Input Output Arguments with Supported Options

Usage

```
match_output_arguments(
  input_args = c(),
  supported_args = c(),
  logger = NULL,
  level = "info"
)
```

Arguments

`input_args`	A character vector of input arguments (can be partial).
`supported_args`	A character vector of supported output arguments.
`logger`	A character vector indicating the logger to target messages at. Defaults to no logging.
`level`	Character string defaulting to "info". Logging level see documentation of futile.logger for details. Supported options are "info" and "debug"

Value

A logical vector of named output arguments

Examples

```
# select nothing
match_output_arguments(supported_args = c("fit", "plots", "samples"))

# select just plots
match_output_arguments("plots", supported_args = c("fit", "plots", "samples"))

# select plots and samples
match_output_arguments(c("plots", "samples"),
                       supported_args = c("fit", "plots", "samples"))
```

```
# lazily select arguments
match_output_arguments("p",
                       supported_args = c("fit", "plots", "samples"))
```

plot_estimates *Plot Estimates*

Description

Plot Estimates

Usage

```
plot_estimates(
  estimate,
  reported,
  ylab = "Cases",
  hline,
  obs_as_col = TRUE,
  max_plot = 10
)
```

Arguments

`estimate`	A data.table of estimates containing the following variables: date, type (must contain "estimate", "estimate based on partial data" and optionally "forecast"),
`reported`	A data.table of reported cases with the following variables: date, confirm.
`ylab`	Character string, defaulting to "Cases". Title for the plot y axis.
`hline`	Numeric, if supplied gives the horizontal intercept for a indicator line.
`obs_as_col`	Logical, defaults to `TRUE`. Should observed data, if supplied, be plotted using columns or as points (linked using a line).
`max_plot`	Numeric, defaults to 10. A multiplicative upper bound on the number of cases shown on the plot. Based on the maximum number of reported cases.

Value

A `ggplot2` object

Examples

```
# define example cases
cases <- EpiNow2::example_confirmed[1:40]

# set up example delays
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
```

```
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)


# run model
out <- EpiNow2::estimate_infections(cases, generation_time = generation_time,
                                    delays = list(incubation_period, reporting_delay),
                                  stan_args = list(cores = ifelse(interactive(), 4, 1)))
# plot infections
plot_estimates(
  estimate = out$summarised[variable == "infections"],
  reported = cases,
  ylab = "Cases", max_plot = 2) + ggplot2::facet_wrap(~type, scales = "free_y")

# plot reported cases estimated via Rt
plot_estimates(estimate = out$summarised[variable == "reported_cases"],
               reported = cases,
               ylab = "Cases")

# plot Rt estimates
plot_estimates(estimate = out$summarised[variable == "R"],
               ylab = "Effective Reproduction No.",
               hline = 1)
```

plot_summary *Plot a Summary of the Latest Results*

Description

Plot a Summary of the Latest Results

Usage

```
plot_summary(summary_results, x_lab = "Region", log_cases = FALSE, max_cases)
```

Arguments

`summary_results`	A data.table as returned by `summarise_results` (the `data` object).
`x_lab`	A character string giving the label for the x axis, defaults to region.
`log_cases`	Logical, should cases be shown on a logged scale. Defaults to `FALSE`
`max_cases`	Numeric, no default. The maximum number of cases to plot.

Value

A `ggplot2` object

process_region *Process regional estimate*

Description

Process regional estimate

Usage

```
process_region(
  out,
  target_region,
  timing,
  return_output = TRUE,
  return_timing = TRUE,
  complete_logger = "EpiNow2.epinow"
)
```

Arguments

`out`	List of output returned by `epinow`
`target_region`	Character string indicating the region being evaluated
`timing`	Output from `Sys.time`
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.
`return_timing`	Logical, should runtime be returned
`complete_logger`	Character string indicating the logger to output the completion of estimation to.

Value

A list of processed output

process_regions *Process all Region Estimates*

Description

Process all Region Estimates

Usage

```
process_regions(regional_out, regions)
```

Arguments

`regional_out`	A list of output from multiple runs of `regional_epinow`
`regions`	A character vector identifying the regions that have been run

Value

A list of all regional estimates and successful regional estimates

---

`regional_epinow` *Real-time Rt Estimation, Forecasting and Reporting by Region*

Description

Estimates Rt by region. See the documentation for `epinow` for further information. The progress of producing estimates across multiple regions is tracked using the `progressr` package. Modify this behaviour using progressr::handlers and enable it in batch by setting `R_PROGRESSR_ENABLE=TRUE` as an environment variable.

Usage

```
regional_epinow(
  reported_cases,
  target_folder = NULL,
  target_date,
  non_zero_points = 2,
  output = c("regions", "summary", "samples", "plots", "latest"),
  return_output = FALSE,
  summary_args = list(),
  logs = tempdir(),
  ...
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date), and region (`region`).
`target_folder`	Character string specifying where to save results (will create if not present).
`target_date`	Date, defaults to maximum found in the data if not specified.
`non_zero_points`	Numeric, the minimum number of time points with non-zero cases in a region required for that region to be evaluated. Defaults to 2.
`output`	A character vector of optional output to return. Supported options are the individual regional estimates ("regions"), samples ("samples"), plots ("plots"), copying the individual region dated folder into a latest folder (if `target_folder` is not null - set using "latest"), the stan fit of the underlying model ("fit"), and an overall summary across regions ("summary"). The default is to return

	samples and plots alongside summarised estimates and summary statistics. If `target_folder` is not NULL then the default is also to copy all results into a latest folder.
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.
`summary_args`	A list of arguments passed to `regional_summary`. See the `regional_summary` documentation for details.
`logs`	Character path indicating the target folder in which to store log information. Defaults to the temporary directory if not specified. Default logging can be disabled if `logs` is set to NULL. If specifying a custom logging setup then the code for `setup_default_logging` and the `setup_logging` function are a sensible place to start.
`...`	Pass additional arguments to `epinow`. See the documentation for `epinow` for details.

Value

A list of output stratified at the top level into regional output and across region output summary output

Examples

```
# construct example distributions
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- list(mean = log(10), mean_sd = log(2),
                        sd = log(2), sd_sd = log(1.1), max = 30)

# uses example case vector
cases <- EpiNow2::example_confirmed[1:40]
cases <- data.table::rbindlist(list(
  data.table::copy(cases)[, region := "testland"],
  cases[, region := "realland"]))

# run epinow across multiple regions and generate summaries
# samples and warmup have been reduced for this example
out <- regional_epinow(reported_cases = cases,
                       samples = 100,
                       generation_time = generation_time,
                       delays = list(incubation_period, reporting_delay),
                       stan_args = list(warmup = 100,
                                        cores = ifelse(interactive(), 4, 1)))
```

`regional_runtimes` *Summarise Regional Runtimes*

Description

Summarise Regional Runtimes

Usage

```
regional_runtimes(
  regional_output = NULL,
  target_folder = NULL,
  target_date = NULL,
  return_output = FALSE
)
```

Arguments

`regional_output`	A list of output as produced by `regional_epinow` and stored in the `regional` list.
`target_folder`	Character string specifying where to save results (will create if not present).
`target_date`	A character string giving the target date for which to extract results (in the format "yyyy-mm-dd"). Defaults to latest available estimates.
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.

Value

A data.table of region run times

Examples

```
# example delays
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# example case vector from EpiSoon
cases <- EpiNow2::example_confirmed[1:30]
cases <- data.table::rbindlist(list(
  data.table::copy(cases)[, region := "testland"],
  cases[, region := "realland"]))

# run basic nowcasting pipeline
regional_out <- regional_epinow(reported_cases = cases,
                                generation_time = generation_time,
                                delays = list(incubation_period, reporting_delay),
                                samples = 100, stan_args = list(warmup = 100),
                                output = c("region", "timing"))

regional_runtimes(regional_output = regional_out$regional)
```

`regional_summary` *Generate Regional Summary Output*

Description

Generate Regional Summary Output

Usage

```
regional_summary(
  regional_output = NULL,
  reported_cases,
  results_dir = NULL,
  summary_dir = NULL,
  target_date = NULL,
  region_scale = "Region",
  all_regions = TRUE,
  return_output = FALSE,
  max_plot = 10
)
```

Arguments

`regional_output`	A list of output as produced by `regional_epinow` and stored in the `regional` list.
`reported_cases`	A data frame of confirmed cases (confirm) by date (date), and region (`region`).
`results_dir`	An optional character string indicating the location of the results directory to extract results from.
`summary_dir`	A character string giving the directory in which to store summary of results.
`target_date`	A character string giving the target date for which to extract results (in the format "yyyy-mm-dd"). Defaults to latest available estimates.
`region_scale`	A character string indicating the name to give the regions being summarised.
`all_regions`	Logical, defaults to `TRUE`. Should summary plots for all regions be returned rather than just regions of interest.
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.
`max_plot`	Numeric, defaults to 10. A multiplicative upper bound on the number of cases shown on the plot. Based on the maximum number of reported cases.

Value

A list of summary measures and plots

Examples

```
# example delays
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# example case vector from EpiSoon
cases <- EpiNow2::example_confirmed[1:30]
cases <- data.table::rbindlist(list(
  data.table::copy(cases)[, region := "testland"],
  cases[, region := "realland"]))

# run basic nowcasting pipeline
out <- regional_epinow(reported_cases = cases,
                       generation_time = generation_time,
                       delays = list(incubation_period, reporting_delay),
                       samples = 50, output = "region",
                       stan_args = list(warmup = 50,
                                        control = list(adapt_delta = 0.95)),
                                        logs = NULL)

regional_summary(regional_output = out$regional,
                 reported_cases = cases)
```

report_cases *Report case counts by date of report*

Description

Report case counts by date of report

Usage

```
report_cases(
  case_estimates,
  case_forecast = NULL,
  delays,
  type = "sample",
  reporting_effect,
  CrIs = c(0.2, 0.5, 0.9)
)
```

Arguments

`case_estimates`	A data.table of case estimates with the following variables: date, sample, cases
`case_forecast`	A data.table of case forecasts with the following variables: date, sample, cases. If not supplied the default is not to incorporate forecasts.
`delays`	A list of delays (i.e incubation period/reporting delay) between infection and report. Each list entry must also be a list containing the mean, standard deviation of the mean (mean_sd), standard deviation (sd), standard deviation of the standard deviation and the maximum allowed value for the that delay (assuming a lognormal distribution with all parameters excepting the max allowed value on the log scale). To use no delays set this to `list()`.
`type`	Character string indicating the method to use to transform counts. Supports either "sample" which approximates sampling or "median" would shift by the median of the distribution.
`reporting_effect`	A `data.table` giving the weekly reporting effect with the following variables: `sample` (must be the same as in `nowcast`), `effect` (numeric scaling factor for each weekday), day (numeric 1 - 7 (1 = Monday and 7 = Sunday)). If not supplied then no weekly reporting effect is assumed.
`CrIs`	Numeric vector of credible intervals to calculate.

Value

A list of `data.tables`. The first entry contains the following variables `sample`, `date` and `cases` with the second being summarised across samples.

Examples

```
# define example cases
cases <- EpiNow2::example_confirmed[1:40]

# set up example delays
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# run model
out <- EpiNow2::estimate_infections(cases, samples = 100,
                                    generation_time = generation_time,
                                    delays = list(incubation_period, reporting_delay),
                                    stan_args = list(warmup = 100,
                                                     cores = ifelse(interactive(), 4, 1)),
                                    estimate_rt = FALSE)

reported_cases <- report_cases(case_estimates =
                                out$samples[variable == "infections"][,
                                cases := as.integer(value)][, value := NULL],
                               delays = list(incubation_period, reporting_delay),
                               type = "sample")
```

```
print(reported_cases)
```

report_plots *Report plots*

Description

Report plots

Usage

```
report_plots(
  summarised_estimates,
  reported,
  target_folder = NULL,
  max_plot = 10
)
```

Arguments

`summarised_estimates`	A data.table of summarised estimates containing the following variables: variable, median, bottom, and top. It should contain the following estimates: R, infections, reported_cases_rt, and r (rate of growth).
`reported`	A data.table of reported cases with the following variables: date, confirm.
`target_folder`	Character string specifying where to save results (will create if not present).
`max_plot`	Numeric, defaults to 10. A multiplicative upper bound on the number of cases shown on the plot. Based on the maximum number of reported cases.

Value

A `ggplot2` object

Examples

```
# define example cases
cases <- EpiNow2::example_confirmed[1:40]

# set up example delays
generation_time <- get_generation_time(disease = "SARS-CoV-2", source = "ganyani")
incubation_period <- get_incubation_period(disease = "SARS-CoV-2", source = "lauer")
reporting_delay <- EpiNow2::bootstrapped_dist_fit(rlnorm(100, log(6), 1), max_value = 30)

# run model
out <- EpiNow2::estimate_infections(cases, samples = 100,
                                    generation_time = generation_time,
                                        delays = list(incubation_period, reporting_delay),
                                        stan_args = list(warmup = 100, cores = 4))

# plot infections
plots <- report_plots(summarised_estimates = out$summarised,
                      reported = cases)
plots
```

`report_summary` *Provide Summary Statistics for Estimated Infections and Rt*

Description

Provide Summary Statistics for Estimated Infections and Rt

Usage

```
report_summary(summarised_estimates, rt_samples, target_folder = NULL)
```

Arguments

`summarised_estimates`	A data.table of summarised estimates containing the following variables: variable, median, bottom, and top. It should contain the following estimates: R, infections, and r (rate of growth).
`rt_samples`	A data.table containing Rt samples with the following variables: sample and value.
`target_folder`	Character string specifying where to save results (will create if not present).

Value

A data.table containing formatted and numeric summary measures

---

`run_region` *Run epinow with Regional Processing Code*

Description

Run epinow with Regional Processing Code

Usage

```
run_region(
  target_region,
  reported_cases,
  target_folder,
  target_date,
  return_output,
  output,
  complete_logger,
  progress_fn,
  ...
)
```

Arguments

`target_region`	Character string indicating the region being evaluated
`reported_cases`	A data frame of confirmed cases (confirm) by date (date), and region (`region`).
`target_folder`	Character string specifying where to save results (will create if not present).
`target_date`	Date, defaults to maximum found in the data if not specified.
`return_output`	Logical, defaults to FALSE. Should output be returned, this automatically updates to TRUE if no directory for saving is specified.
`output`	A character vector of optional output to return. Supported options are the individual regional estimates ("regions"), samples ("samples"), plots ("plots"), copying the individual region dated folder into a latest folder (if `target_folder` is not null - set using "latest"), the stan fit of the underlying model ("fit"), and an overall summary across regions ("summary"). The default is to return samples and plots alongside summarised estimates and summary statistics. If `target_folder` is not NULL then the default is also to copy all results into a latest folder.
`complete_logger`	Character string indicating the logger to output the completion of estimation to.
`progress_fn`	Function as returned by `progressr::progressor`. Allows the use of a progress bar.
`...`	Pass additional arguments to `epinow`. See the documentation for `epinow` for details.

Value

A list of processed output as produced by `process_region`

`R_to_growth` *Convert Reproduction Numbers to Growth Rates*

Description

See here for justification.

Usage

```
R_to_growth(R, gamma_mean, gamma_sd)
```

Arguments

`R`	Numeric, Reproduction number estimates
`gamma_mean`	Numeric, mean of the gamma distribution
`gamma_sd`	Numeric, standard deviation of the gamma distribution

Value

Numeric vector of reproduction number estimates

Examples

```
R_to_growth(2.18, 4, 1)
```

`sample_approx_dist` *Approximate Sampling a Distribution using Counts*

Description

Approximate Sampling a Distribution using Counts

Usage

```
sample_approx_dist(
  cases = NULL,
  dist_fn = NULL,
  max_value = 120,
  earliest_allowed_mapped = NULL,
  direction = "backwards",
  type = "sample",
  truncate_future = TRUE
)
```

Arguments

`cases`	A dataframe of cases (in date order) with the following variables: `date` and `cases`.
`dist_fn`	Function that takes two arguments with the first being numeric and the second being logical (and defined as `dist`). Should return the probability density or a sample from the defined distribution. See the examples for more.
`max_value`	Numeric, maximum value to allow. Defaults to 120 days
`earliest_allowed_mapped`	A character string representing a date ("2020-01-01"). Indicates the earliest allowed mapped value.
`direction`	Character string, defato "backwards". Direction in which to map cases. Supports either "backwards" or "forwards".
`type`	Character string indicating the method to use to transform counts. Supports either "sample" which approximates sampling or "median" would shift by the median of the distribution.
`truncate_future`	Logical, should cases be truncated if they occur after the first date reported in the data. Defaults to `TRUE`.

Value

A `data.table` of cases by date of onset

Examples

```
cases <- EpiNow2::example_confirmed
cases <- cases[, cases := as.integer(confirm)]
print(cases)

# total cases
sum(cases$cases)

delay_fn <- function(n, dist, cum) {
              if(dist) {
                pgamma(n + 0.9999, 2, 1) - pgamma(n - 1e-5, 2, 1)
               }else{
                as.integer(rgamma(n, 2, 1))
               }
             }

onsets <- sample_approx_dist(cases = cases,
                             dist_fn = delay_fn)

# estimated onset distribution
print(onsets)

# check that sum is equal to reported cases
total_onsets <- median(
   purrr::map_dbl(1:100,
                  ~ sum(sample_approx_dist(cases = cases,
                  dist_fn = delay_fn)$cases)))
total_onsets


# map from onset cases to reported
reports <- sample_approx_dist(cases = cases,
                              dist_fn = delay_fn,
                              direction = "forwards")


# map from onset cases to reported using a mean shift
reports <- sample_approx_dist(cases = cases,
                              dist_fn = delay_fn,
                              direction = "forwards",
                              type = "median")
```

`save_estimate_infections` *Save Estimated Infections*

Description

Save Estimated Infections

Usage

```
save_estimate_infections(
  estimates,
  target_folder = NULL,
  samples = TRUE,
  return_fit = TRUE
)
```

Arguments

`estimates`	List of data frames as output by `estimate_infections`
`target_folder`	Character string specifying where to save results (will create if not present).
`samples`	Logical, defaults to TRUE. Should samples be saved
`return_fit`	Logical, defaults to FALSE. Should the fitted stan model be returned.

`save_forecast_infections` *Save Forecast Infections*

Description

Save Forecast Infections

Usage

```
save_forecast_infections(forecast, target_folder = NULL, samples = TRUE)
```

Arguments

`forecast`	A list of data frames as output by `forecast_infections`
`target_folder`	Character string specifying where to save results (will create if not present).
`samples`	Logical, defaults to TRUE. Should samples be saved

`save_input` *Save Observed Data*

Description

Save Observed Data

Usage

```
save_input(reported_cases, target_folder)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.
`target_folder`	Character string specifying where to save results (will create if not present).

`setup_default_logging` *Setup Default Logging*

Description

Setup Default Logging

Usage

```
setup_default_logging(
  logs = tempdir(),
  mirror_epinow = FALSE,
  mirror_epinow_fit = FALSE,
  target_date = NULL
)
```

Arguments

`logs`	Character path indicating the target folder in which to store log information. Defaults to the temporary directory if not specified. Default logging can be disabled if `logs` is set to NULL. If specifying a custom logging setup then the code for `setup_default_logging` and the `setup_logging` function are a sensible place to start.
`mirror_epinow`	Logical, defaults to FALSE. Should internal logging be returned from `epinow` to the console.
`mirror_epinow_fit`	Logical, defaults to FALSE. Should internal logging be returned from internal fitting functions to the console.
`target_date`	Date, defaults to maximum found in the data if not specified.

Examples

```
setup_default_logging()
```

`setup_dt` *Convert to Data Table*

Description

Convert to Data Table

Usage

```
setup_dt(reported_cases)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.

Value

A data table

`setup_future` *Set up Future Backend*

Description

A utility function that aims to streamline the set up of the required future backend with sensible defaults for most users of `regional_epinow`. More advanced users are recommended to setup their own `future` backend based on their available resources.

Usage

```
setup_future(
  reported_cases,
  strategies = c("multiprocess", "multiprocess"),
  min_cores_per_worker = 4
)
```

Arguments

`reported_cases`	A data frame of confirmed cases (confirm) by date (date), and region (`region`).
`strategies`	A vector length 1 to 2 of strategies to pass to `future::plan`. Nesting of parallisation is from the top level down. The default is to set up nesting parallisation with both using `future::multiprocess`. For single level parallisation use a single strategy or `future::plan` directly. See `?future::plan` for options.
`min_cores_per_worker`	Numeric, the minimum number of cores per worker. Defaults to 4 which assumes 4 MCMC chains are in use per region.

Value

Numeric number of cores to use per worker. If greater than 1 pass to `stan_args = list(cores = "output from setup future")` or use `future = TRUE`. If only a single strategy is used then nothing is returned.

setup_logging *Setup Logging*

Description

Sets up `futile.logger` logging, which is integrated into `EpiNow2`. See the documentation for `futile.logger` for full details. By default `EpiNow2` prints all logs at the "INFO" level and returns them to the console.

Usage

```
setup_logging(
  threshold = "INFO",
  file = NULL,
  mirror_to_console = FALSE,
  name = "EpiNow2"
)
```

Arguments

`threshold`	Character string indicating the logging level see (?futile.logger for details of the available options). Defaults to "INFO".
`file`	Character string indicating the path to save logs to. By default logs will be written to the console.
`mirror_to_console`	Logical, defaults to `FALSE`. If saving logs to a file should they also be duplicated in the console.
`name`	Character string defaulting to EpiNow2. This indicates the name of the logger to setup. The default logger for EpiNow2 is called EpiNow2. Nested options include: Epinow2.epinow which controls all logging for `epinow` and nested functions, EpiNow2.epinow.estimate_infections (logging in `estimate_infections`), and EpiNow2.epinow.estimate_infections.fit (logging in fitting functions).

Value

Nothing

Examples

```
# set up info only logs with errors only
# for logging related to epinow (or nested) calls
# (info logs are enabled by default at all levels.)
setup_logging("Info", name = "EpiNow2")
setup_logging("ERROR", name = "EpiNow2.epinow")
```

`setup_target_folder` *Setup Target Folder for Saving*

Description

Setup Target Folder for Saving

Usage

```
setup_target_folder(target_folder = NULL, target_date)
```

Arguments

`target_folder`	Character string specifying where to save results (will create if not present).
`target_date`	Date, defaults to maximum found in the data if not specified.

Value

A list containing the path to the dated folder and the latest folder

`simulate_cases` *Simulate Cases by Date of Infection, Onset and Report*

Description

Simulate Cases by Date of Infection, Onset and Report

Usage

```
simulate_cases(
  rts,
  initial_cases,
  initial_date,
  generation_interval,
  rdist = rpois,
  delay_defs,
  reporting_effect,
  reporting_model,
  truncate_future = TRUE,
  type = "sample"
)
```

Arguments

`rts`	A dataframe of containing two variables `rt` and `date` with `rt` being numeric and `date` being a date.
`initial_cases`	Integer, initial number of cases.
`initial_date`	Date, (i.e `as.Date("2020-02-01")`). Starting date of the simulation.
`generation_interval`	Numeric vector describing the generation interval probability density
`rdist`	A function to be used to sample the number of cases. Must take two arguments with the first specifying the number of samples and the second the mean. Defaults to `rpois` if not supplied
`delay_defs`	A list of single row data.tables that each defines a delay distribution (model, parameters and maximum delay for each model). See `lognorm_dist_def` for an example of the structure.
`reporting_effect`	A numeric vector of length 7 that allows the scaling of reported cases by the day on which they report (1 = Monday, 7 = Sunday). By default no scaling occurs.
`reporting_model`	A function that takes a single numeric vector as an argument and returns a single numeric vector. Can be used to apply stochastic reporting effects. See the examples for details.
`truncate_future`	Logical, should cases be truncated if they occur after the first date reported in the data. Defaults to `TRUE`.
`type`	Character string indicating the method to use to transform counts. Supports either "sample" which approximates sampling or "median" would shift by the median of the distribution.

Value

A dataframe containing three variables: `date`, `cases` and `reference`.

`stop_timeout` *Timeout Error*

Description

Timeout Error

Usage

```
stop_timeout(fit)
```

Arguments

`fit`	A stan fit object

Value

Nothing

`summarise_key_measures` *Summarise rt and cases*

Description

Summarise rt and cases

Usage

```
summarise_key_measures(
  regional_results = NULL,
  results_dir = NULL,
  summary_dir = NULL,
  type = "region",
  date = "latest"
)
```

Arguments

`regional_results`	A list of dataframes as produced by `get_regional_results`
`results_dir`	Character string indicating the directory from which to extract results.
`summary_dir`	Character string the directory into which to save results as a csv.
`type`	Character string, the region identifier to apply (defaults to region).
`date`	A Character string (in the format "yyyy-mm-dd") indicating the date to extract data for. Defaults to "latest" which finds the latest results available.

Value

A list of summarised Rt, cases by date of infection and cases by date of report

summarise_results *Summarise Real-time Results*

Description

Summarise Real-time Results

Usage

```
summarise_results(
  regions,
  summaries = NULL,
  results_dir = NULL,
  target_date = NULL,
  region_scale = "Region"
)
```

Arguments

regions	An character string containing the list of regions to extract results for (must all have results for the same target date).
summaries	A list of summary data frames as output by `epinow`
results_dir	An optional character string indicating the location of the results directory to extract results from.
target_date	A character string indicating the target date to extract results for. All regions must have results for this date.
region_scale	A character string indicating the name to give the regions being summarised.

Value

A list of summary data

theme_map *Custom Map Theme*

Description

Custom Map Theme

Usage

```
theme_map(
  map = NULL,
  continuous = FALSE,
  variable_label = NULL,
  trans = "identity",
  fill_labels = NULL,
  scale_fill = NULL,
  breaks = NULL,
  ...
)
```

Arguments

`map`	`ggplot2` map object
`continuous`	Logical defaults to `FALSE`. Is the fill variable continuous.
`variable_label`	A character string indicating the variable label to use. If not supplied then the underlying variable name is used.
`trans`	A character string specifying the transform to use on the specified metric. Defaults to no transform ("identity"). Other options include log scaling ("log") and log base 10 scaling ("log10"). For a complete list of options see `ggplot2::continous_scale`.
`fill_labels`	A function to use to allocate legend labels. An example (used below) is `scales::percent`, which can be used for percentage data.
`scale_fill`	Function to use for scaling the fill. Defaults to a custom `ggplot2::scale_fill_manual`, which expects the possible values to be "Increasing", "Likely increasing", "Likely decreasing", "Decreasing" or "Unsure".
`breaks`	Breaks to use in legend. Defaults to `ggplot2::waiver`.
`...`	Additional arguments passed to the `scale_fill` function

Value

A `ggplot2` object

`tune_inv_gamma` *Tune an Inverse Gamma to Achieve the Target Truncation*

Description

Tune an Inverse Gamma to Achieve the Target Truncation

Usage

```
tune_inv_gamma(lower = 2, upper = 21)
```

Arguments

`lower`	Numeric, defaults to 2. Lower truncation bound.
`upper`	Numeric, defaults to 21. Upper truncation bound.

Value

A list of alpha and beta values that describe a inverse gamma distribution that achieves the target truncation.

Examples

```
tune_inv_gamma(lower = 2, upper = 21)
```

`update_horizon` *Updates Forecast Horizon Based on Input Data and Target*

Description

This function makes sure that a forecast is returned for the user specified time period beyond the target date.

Usage

```
update_horizon(horizon, target_date, reported_cases)
```

Arguments

`horizon`	Numeric, defaults to 7. Number of days into the future to forecast.
`target_date`	Date, defaults to maximum found in the data if not specified.
`reported_cases`	A data frame of confirmed cases (confirm) by date (date). confirm must be integer and date must be in date format.

Value

Numeric forecast horizon adjusted for the users intention

Index