Package: lineagefreq (via r-universe)

May 19, 2026

Title Lineage Frequency Dynamics from Genomic Surveillance Counts

Version 0.6.0

Description Models pathogen lineage frequency dynamics from genomic surveillance count data. Provides a unified interface for multinomial logistic regression, hierarchical partial-pooling models, and the Piantham approximation for relative reproduction number estimation. Features include rolling-origin backtesting, standardized forecast scoring, Compositional Adaptive Prediction Sets (CAPS) for horizon-aware calibrated forecasting, lineage collapsing, emergence detection, and sequencing power analysis. Designed for real-time public health surveillance of any variant-resolved pathogen. Methods described in Abousamra, Figgins, and Bedford (2024) <doi:10.1371/journal.pcbi.1012443>.

License MIT + file LICENSE

URL https://github.com/CuiweiG/lineagefreq, https://cuiweig.github.io/lineagefreq

BugReports https://github.com/CuiweiG/lineagefreq/issues

Depends R (>= 4.1.0)

Imports cli (>= 3.4.0), dplyr (>= 1.1.0), grDevices, ggplot2 (>= 3.4.0), MASS, numDeriv, rlang (>= 1.1.0), stats, tibble (>= 3.1.0), tidyr (>= 1.3.0)

Suggests broom, cmdstanr, covr, knitr, posterior, rmarkdown, scales, testthat (>= 3.0.0), withr

Additional_repositories https://stan-dev.r-universe.dev

VignetteBuilder knitr

Config/testthat/edition 3

Language en-US

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libicu-dev

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

Date/Publication 2026-04-19 04:22:42 UTC

RemoteUrl https://github.com/cuiweig/lineagefreq

RemoteRef HEAD

RemoteSha a08fca034b66ac388d6c4cf4937becd531c82833

Contents

`adaptive_design` *Adaptive sequencing allocation via Thompson sampling*

Description

Reallocates sequencing resources across regions in real time, directing effort toward strata where uncertainty reduction has the highest decision value. Unlike static Neyman allocation (which optimises for a single time point), this function adapts to evolving variant dynamics across multiple surveillance rounds.

Usage

```
adaptive_design(
  data,
  capacity,
  n_rounds = NULL,
  strategy = c("thompson", "ucb"),
```

```
  target_lineage = NULL,
  exploration = 2,
  seed = NULL
)
```

Arguments

`data`	An `lfq_data` object with a location column.
`capacity`	Total sequencing capacity per round (integer).
`n_rounds`	Number of allocation rounds to simulate. Default `NULL` uses the number of time points in the data.
`strategy`	Allocation strategy: `"thompson"` (default) for Thompson sampling, or `"ucb"` for Upper Confidence Bound.
`target_lineage`	Character; lineage to optimise detection for. Default `NULL` optimises for overall frequency estimation.
`exploration`	Exploration parameter for UCB. Default 2.0. Larger values explore more; smaller values exploit current best regions.
`seed`	Random seed. Default `NULL`.

Details

Thompson sampling draws from the posterior distribution of frequency estimates and allocates proportional to the posterior variance. Regions with high uncertainty receive more sequences, but the stochastic draws naturally balance exploration (sampling uncertain regions) and exploitation (sampling where variants are most prevalent).

UCB allocates proportional to the upper confidence bound of the estimation error: $\text{score}_r = \hat{\sigma}_r + c\sqrt{2\log(t)/n_r}$ where $\hat{\sigma}_r$ is the current estimation uncertainty in region r, n_r is the cumulative allocation, t is the round, and c is the exploration parameter.

Value

An `adaptive_allocation` S3 class with components:

allocations Tibble with `round`, `region`, `n_allocated`, `uncertainty`, `frequency`.

summary Tibble with per-region totals and mean allocation.

strategy Character; strategy used.

capacity Integer; per-round capacity.

See Also

`surveillance_value` for EVOI analysis.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.3, "B" = 0.9),
  n_timepoints = 12, seed = 1)
ad <- adaptive_design(sim, capacity = 200, n_rounds = 8)
ad
```

`alert_threshold` *Sequential detection of emerging variants*

Description

Applies a sequential probability ratio test (SPRT) or CUSUM procedure to lineage frequency data, determining when accumulated evidence is sufficient to declare a variant "emerging" rather than sampling noise. Controls the false alarm rate while minimising detection delay.

Usage

```
alert_threshold(
  data,
  method = c("sprt", "cusum"),
  alpha = 0.05,
  beta = 0.1,
  delta_0 = 0,
  delta_1 = 0.03,
  threshold = 5
)
```

Arguments

`data`	An `lfq_data` object.
`method`	Detection method: `"sprt"` (default) for the sequential probability ratio test, or `"cusum"` for the cumulative sum control chart.
`alpha`	False alarm probability. Default 0.05.
`beta`	Missed detection probability. Default 0.10.
`delta_0`	Null hypothesis growth rate (no emergence). Default 0 (frequency is stable).
`delta_1`	Alternative hypothesis growth rate (emergence). Default 0.03 (3 percent per-week increase on logit scale).
`threshold`	CUSUM decision threshold. Default 5.0. Only used when `method = "cusum"`.

Details

SPRT (Wald, 1945) computes the log-likelihood ratio between the alternative (lineage is growing at rate δ_1) and the null (frequency is stable). The test stops when the cumulative log-ratio crosses the upper boundary $B = \log((1-\beta)/\alpha)$ (declare emerging) or the lower boundary $A = \log(\beta/(1-\alpha))$ (declare stable).

CUSUM accumulates deviations from expected frequency under the null: $S_t = \max(0, S_{t-1} + (x_t - k))$ where x_t is the observed frequency change and k is the allowance (half the shift to detect). An alert is raised when $S_t > h$.

Value

A tibble with columns `lineage`, `date`, `statistic` (log-likelihood ratio or CUSUM value), `alert` (logical), `direction` (emerging/declining/ stable), and `confidence` (1 - alpha).

References

Wald A (1945). Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2), 117–186.

See Also

`summarize_emerging` for non-sequential trend tests, `detection_horizon` for prospective power analysis.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.5, "B" = 0.8),
  n_timepoints = 15, seed = 1)
alerts <- alert_threshold(sim)
alerts
```

`as.data.frame.lfq_data` *Convert lfq_data to long-format tibble*

Description

Convert lfq_data to long-format tibble

Usage

```
## S3 method for class 'lfq_data'
as.data.frame(x, ...)
```

Arguments

x	An lfq_data object.
...	Ignored.

Value

A tibble with all columns.

Examples

```
data(sarscov2_us_2022)
x <- lfq_data(sarscov2_us_2022, lineage = variant,
              date = date, count = count, total = total)
as.data.frame(x)
```

---

as_lfq_data *Coerce to lfq_data*

Description

Generic function to convert various data formats into lfq_data objects. Methods can be defined for specific input classes to enable seamless interoperability with other genomic surveillance packages.

Usage

```
as_lfq_data(x, ...)

## S3 method for class 'lfq_data'
as_lfq_data(x, ...)

## S3 method for class 'data.frame'
as_lfq_data(x, ...)
```

Arguments

x	An object to coerce.
...	Additional arguments passed to methods. For the `data.frame` method, these are passed to `lfq_data()`.

Value

An lfq_data object.

An lfq_data object.

An lfq_data object.

See Also

`lfq_data()` for the primary constructor.

Examples

```
df <- data.frame(
  date    = rep(as.Date("2024-01-01") + c(0, 7), each = 2),
  lineage = rep(c("A", "B"), 2),
  count   = c(80, 20, 60, 40)
)
x <- as_lfq_data(df, lineage = lineage, date = date, count = count)
x
```

`augment.lfq_fit` *Augment data with fitted values from an lfq_fit object*

Description

Augment data with fitted values from an lfq_fit object

Usage

```
augment.lfq_fit(x, ...)
```

Arguments

`x`	An `lfq_fit` object.
`...`	Ignored.

Value

A tibble with columns: `.date`, `.lineage`, `.fitted_freq`, `.observed`, `.pearson_resid`.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
augment.lfq_fit(fit)
```

`autoplot.lfq_fit` *Plot lineage frequency model results*

Description

Plot lineage frequency model results

Usage

```
## S3 method for class 'lfq_fit'
autoplot(
  object,
  type = c("frequency", "advantage", "trajectory", "residuals"),
  generation_time = NULL,
  ...
)
```

Arguments

`object`	An `lfq_fit` object.
`type`	Plot type: `"frequency"` (default), `"advantage"`, `"trajectory"`, or `"residuals"`.
`generation_time`	Required when `type = "advantage"`.
`...`	Ignored.

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
autoplot(fit)
autoplot(fit, type = "advantage", generation_time = 5)
```

autoplot.lfq_forecast *Plot a lineage frequency forecast*

Description

Plot a lineage frequency forecast

Usage

```
## S3 method for class 'lfq_forecast'
autoplot(object, ...)
```

Arguments

object	An lfq_forecast object.
...	Ignored.

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
fc <- forecast(fit, horizon = 14)
autoplot(fc)
```

backtest *Rolling-origin backtesting of lineage frequency models*

Description

Evaluates forecast accuracy by repeatedly fitting models on historical data and comparing predictions to held-out observations. This implements the evaluation framework described in Abousamra et al. (2024).

Usage

```
backtest(
  data,
  engines = "mlr",
  origins = "weekly",
  horizons = c(7L, 14L, 21L, 28L),
  min_train = 42L,
  ...
)
```

Arguments

`data`	An lfq_data object.
`engines`	Character vector of engine names to compare. Default `"mlr"`.
`origins`	How to select forecast origins: • `"weekly"` (default): one origin per unique date, starting after `min_train` days. • An integer: use every Nth date as an origin. • A Date vector: use these specific dates as origins.
`horizons`	Integer vector of forecast horizons in days. Default `c(7, 14, 21, 28)`.
`min_train`	Minimum training window in days. Origins earlier than `min(date) + min_train` are skipped. Default 42 (6 weeks).
`...`	Additional arguments passed to `fit_model()` (e.g., `generation_time` for the Piantham engine).

Details

Implements the rolling-origin evaluation framework described in Abousamra et al. (2024), Section 2.4. At each origin date, the model is fit on data up to that date and forecasts are compared to held-out future observations. This avoids look-ahead bias and provides an honest assessment of real-time forecast accuracy.

Value

An `lfq_backtest` object (tibble subclass) with columns:

origin_date Date used as the training cutoff.

target_date Date being predicted.

horizon Forecast horizon in days.

engine Engine name.

lineage Lineage name.

predicted Predicted frequency (median).

lower Lower prediction bound.

upper Upper prediction bound.

observed Observed frequency at target_date.

References

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

See Also

`score_forecasts()` to compute accuracy metrics, `compare_models()` to rank engines.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
bt
```

`calibrate` *Calibration diagnostics for lineage frequency forecasts*

Description

Assesses whether prediction intervals from backtesting are well-calibrated by computing PIT (Probability Integral Transform) values, reliability diagrams, and uniformity tests. A perfectly calibrated forecaster produces PIT values that are uniformly distributed on $[0, 1]$.

Usage

```
calibrate(x, observed = NULL, n_bins = 10L)

## S3 method for class 'lfq_backtest'
calibrate(x, observed = NULL, n_bins = 10L)

## S3 method for class 'lfq_forecast'
calibrate(x, observed = NULL, n_bins = 10L)
```

Arguments

`x`	An `lfq_backtest` object from `backtest`, or an `lfq_forecast` object from `forecast` together with observed data.
`observed`	For `lfq_forecast` objects: a numeric vector of observed frequencies corresponding to the forecast rows. Ignored for `lfq_backtest` objects (which carry their own observed values).
`n_bins`	Number of bins for the PIT histogram and reliability diagram. Default 10.

Details

The PIT value for a single forecast-observation pair is defined as the quantile of the observation within the forecast distribution. Under the assumption of Gaussian prediction intervals (which the parametric simulation in `forecast` approximates), the PIT is computed as $\Phi((y - \hat{y})/\hat{\sigma})$ where $\hat{y}$ is the predicted median, $\hat{\sigma}$ is derived from the prediction interval width, and Φ is the standard normal CDF.

The reliability diagram plots observed coverage against nominal coverage at levels 10 through 90 percent. Perfect calibration lies on the diagonal.

Value

A `calibration_report` object (S3 class) with components:

pit_values Numeric vector of PIT values in $[0, 1]$.

pit_histogram Tibble with `bin`, `count`, `density`, `expected` columns.

reliability Tibble with `nominal`, `observed` coverage at each level.

ks_test List with `statistic` and `p_value` from the Kolmogorov-Smirnov test for uniformity.

n Integer; number of forecast-observation pairs used.

References

Gneiting T, Balabdaoui F, Raftery AE (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B*, 69(2), 243–268. doi:10.1111/j.14679868.2007.00587.x

See Also

`recalibrate` for post-hoc recalibration, `score_forecasts` for proper scoring rules.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
cal <- calibrate(bt)
cal
```

`calibrate_joint` *Joint Calibration Diagnostics*

Description

Assess calibration of multivariate forecasts using the energy score, joint coverage, and multivariate rank histograms.

Usage

```
calibrate_joint(bt, n_bins = 10L)
```

Arguments

`bt`	An `lfq_backtest` object.
`n_bins`	Integer; number of rank histogram bins (default 10).

Details

The energy score is a multivariate proper scoring rule that generalises the CRPS (Gneiting & Raftery, 2007, Section 5). For a deterministic forecast f and observation y:

$$ES = ||f - y||^2$$

where the norm is the Aitchison distance on the simplex.

Joint coverage at level $1 - \alpha$: the fraction of observed composition vectors falling within Aitchison distance q_α of the forecast, where q_α is derived from the empirical distribution of distances.

The multivariate rank histogram uses the pre-rank approach: the rank of the observation among an ensemble is computed using the Aitchison distance to the ensemble mean.

Value

A `joint_calibration_report` S3 object (list) with:

energy_scores Tibble with origin_date, horizon, energy_score.

mean_energy_score Overall mean energy score.

joint_coverage Tibble with nominal level, observed coverage, using Aitchison distance from backtest results.

rank_histogram Tibble with bin, count, density, expected.

n Number of forecast-observation vectors.

`cdc_ba2_transition` *CDC SARS-CoV-2 variant proportions: BA.1 to BA.2 transition (US, 2022)*

Description

Real surveillance data covering the Omicron BA.1 to BA.2 variant replacement in the United States, December 2021 through June 2022. This is one of the best-documented variant replacement events and serves as an independent validation dataset.

Usage

```
cdc_ba2_transition
```

Format

A data frame with 150 rows and 4 columns:

date Biweek ending date (Date).

lineage Lineage name: BA.1, BA.2, BA.2.12.1, BA.4/5, Other.

count Approximate sequence count per biweek (integer).

proportion CDC weighted proportion estimate (numeric).

Source

CDC COVID Data Tracker (data.cdc.gov, public domain).

References

Lyngse FP, et al. (2022). Household transmission of SARS-CoV-2 Omicron variant of concern subvariants BA.1 and BA.2 in Denmark. *Nature Communications*, 13:5760. doi:10.1038/s41467-022334980

Examples

```
data(cdc_ba2_transition)
vd <- lfq_data(cdc_ba2_transition,
               date = date, lineage = lineage, count = count)
fit <- fit_model(vd, engine = "mlr", pivot = "BA.1")
# BA.2 Rt ~ 1.34 (consistent with published estimates)
growth_advantage(fit, type = "relative_Rt", generation_time = 3.2)
```

cdc_sarscov2_jn1 *CDC SARS-CoV-2 variant proportions: JN.1 emergence (US, 2023-2024)*

Description

Real surveillance data from the CDC's national genomic surveillance program covering the emergence and dominance of the SARS-CoV-2 JN.1 lineage in the United States, October 2023 through June 2024.

Usage

```
cdc_sarscov2_jn1
```

Format

A data frame with 171 rows and 4 columns:

date Biweek ending date (Date).

lineage Lineage name (character): JN.1, XBB.1.5, EG.5.1, HV.1, HK.3, BA.2.86, KP.2, KP.3, JN.1.11.1, Other.

count Approximate sequence count per biweek (integer).

proportion CDC weighted proportion estimate (numeric).

Details

Derived from CDC's published weighted variant proportion estimates. Approximate biweekly sequence counts were reconstructed from proportions using a reference total of 8,000 sequences per period. The original proportions are retained in the `proportion` column.

Source

CDC COVID Data Tracker, SARS-CoV-2 Variant Proportions. Dataset ID: jr58-6ysp. https://data.cdc.gov/Laboratory-Surveillance/SARS-CoV-2-Variant-Proportions/jr58-6ysp

Public domain (U.S. Government Work, 17 USC 105).

References

Ma KC, et al. (2024). Genomic Surveillance for SARS-CoV-2 Variants. *MMWR*, 73(42):941–948. doi:10.15585/mmwr.mm7342a1

Examples

```
data(cdc_sarscov2_jn1)
vd <- lfq_data(cdc_sarscov2_jn1,
               date = date, lineage = lineage, count = count)
fit <- fit_model(vd, engine = "mlr")
growth_advantage(fit, type = "relative_Rt", generation_time = 5)
```

coef.lfq_fit *Extract coefficients from a lineage frequency model*

Description

Extract coefficients from a lineage frequency model

Usage

```
## S3 method for class 'lfq_fit'
coef(object, type = c("growth_rate", "all"), ...)
```

Arguments

object	An lfq_fit object.
type	What to return: "growth_rate" (default) for growth rates only, or "all" for intercepts and growth rates.
...	Ignored.

Value

Named numeric vector.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
coef(fit)
```

collapse_lineages *Collapse rare lineages into an aggregate group*

Description

Merges lineages that never exceed a frequency or count threshold into a single group (default "Other"). Useful for reducing noise from dozens of low-frequency lineages.

Usage

```
collapse_lineages(
  x,
  min_freq = 0.01,
  min_count = 10L,
  other_label = "Other",
  mapping = NULL
)
```

Arguments

x	An lfq_data object.
min_freq	Minimum peak frequency a lineage must reach at any time point to be kept. Default 0.01 (1%).
min_count	Minimum total count across all time points to be kept. Default 10.
other_label	Label for the collapsed group. Default "Other".
mapping	Optional named character vector for custom grouping. Names = original lineage names, values = group names. If provided, min_freq and min_count are ignored.

Value

An lfq_data object with rare lineages merged.

Examples

```
sim <- simulate_dynamics(n_lineages = 6,
  advantages = c(A = 1.3, B = 1.1, C = 0.95, D = 0.5, E = 0.3),
  n_timepoints = 12, seed = 1)
collapsed <- collapse_lineages(sim, min_freq = 0.05)
attr(collapsed, "lineages")
```

---

compare_models *Compare model engines from backtest scores*

Description

Summarises and ranks engines across horizons based on forecast accuracy scores.

Usage

```
compare_models(scores, by = "engine")
```

Arguments

scores	Output of score_forecasts().
by	Grouping variable(s). Default "engine".

Value

A tibble with average scores per group, sorted by MAE.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
sc <- score_forecasts(bt)
compare_models(sc)
```

`conformal_forecast` *Conformal prediction intervals for lineage frequencies*

Description

Produces distribution-free prediction intervals with finite-sample coverage guarantees using split conformal inference. Unlike the parametric intervals from `forecast`, conformal intervals require no distributional assumptions on the residuals and are valid under exchangeability.

Usage

```
conformal_forecast(
  fit,
  data,
  horizon = 28L,
  ci_level = 0.95,
  method = c("split", "aci"),
  cal_fraction = 0.3,
  gamma = 0.05,
  seed = NULL
)
```

Arguments

`fit`	An `lfq_fit` object (any engine).
`data`	The `lfq_data` object used to fit the model.
`horizon`	Number of days to forecast. Default 28.
`ci_level`	Target coverage level. Default 0.95.
`method`	Conformal method: `"split"` (default) for split conformal prediction, or `"aci"` for adaptive conformal inference with online coverage correction.
`cal_fraction`	Fraction of the data reserved for the calibration set (split conformal only). Default 0.3.
`gamma`	Learning rate for adaptive conformal inference. Default 0.05. Controls how quickly the coverage target adjusts in response to observed miscoverage.
`seed`	Random seed for the calibration split. Default `NULL`.

Details

Split conformal prediction partitions the training data into a proper training set and a calibration set. The model is refit on the training set, and conformity scores (absolute residuals) are computed on the calibration set. The prediction interval at a new point is the point forecast plus or minus the $(1-\alpha)(1+1/n_{\text{cal}})$ quantile of the calibration scores. This provides exact $1-\alpha$ marginal coverage under exchangeability (Vovk et al. 2005).

Adaptive conformal inference (ACI) (Gibbs and Candes, 2021) adjusts the miscoverage level online to maintain long-run coverage even when the data distribution shifts over time, as is typical in surveillance data during variant replacement waves.

Value

An `lfq_forecast` object with conformal prediction intervals. The object is fully compatible with `autoplot` and other forecast methods.

References

Vovk V, Gammerman A, Shafer G (2005). *Algorithmic Learning in a Random World*. Springer.

Gibbs I, Candes E (2021). Adaptive conformal inference under distribution shift. *Advances in Neural Information Processing Systems*, 34.

See Also

`forecast` for parametric prediction intervals, `calibrate` for calibration diagnostics.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
fit <- fit_model(sim, engine = "mlr")
fc_conf <- conformal_forecast(fit, sim, horizon = 21)
fc_conf
```

---

conformal_forecast_joint

Joint Conformal Prediction on the Simplex

Description

Produces a compositional prediction region that respects the simplex constraint (frequencies sum to 1) using Aitchison geometry. Unlike marginal conformal prediction, joint prediction guarantees that the *entire* frequency vector is covered, not just individual lineages.

Usage

```
conformal_forecast_joint(
  fit,
  data,
  horizon = 28L,
  ci_level = 0.95,
  cal_fraction = 0.3,
  seed = NULL
)
```

Arguments

fit	An lfq_fit object.
data	An lfq_data object (same data used to fit the model).
horizon	Integer; forecast horizon in days (default 28).
ci_level	Numeric in (0,1); coverage target (default 0.95).
cal_fraction	Numeric in (0,1); fraction of dates for calibration (default 0.3).
seed	Optional integer for reproducibility.

Details

The nonconformity score is the Aitchison distance between predicted and observed compositions. The Aitchison distance equals the Euclidean distance in the isometric log-ratio (ILR) transformed space, which respects the geometry of the simplex (Aitchison, 1986).

The prediction region is the set of all compositions within Aitchison distance r of the point forecast, where r is the $(1-\alpha)(1+1/n)$ quantile of calibration distances. Marginal intervals are obtained by projecting this region onto each coordinate axis, then intersecting with $[0, 1]$.

Value

An lfq_conformal_joint S3 object (list) with:

forecast The point forecast (lfq_forecast).

radius Conformal radius in Aitchison distance.

marginal_intervals Tibble with .date, .lineage, .lower_joint, .upper_joint — marginal bounds projected from the joint region.

marginal_only Tibble with .lower_marginal, .upper_marginal from standard marginal conformal prediction.

comparison Tibble indicating whether joint intervals are wider or narrower than marginal intervals per lineage.

calibration_scores Aitchison distances on calibration set.

ci_level Nominal coverage level.

n_cal Number of calibration compositions.

References

Aitchison J (1986). *The Statistical Analysis of Compositional Data*. Chapman & Hall.

Vovk V, Gammerman A, Shafer G (2005). *Algorithmic Learning in a Random World*. Springer.

`detection_horizon` *Detection horizon for an emerging variant*

Description

Given current sequencing capacity and a hypothetical variant at initial prevalence p_0 growing at rate δ, estimates the number of surveillance periods (weeks) until the probability of detection exceeds a specified threshold.

Usage

```
detection_horizon(
  initial_prev = 0.001,
  growth_rate = 1.3,
  n_per_period = 500L,
  n_periods = 26L,
  detection_threshold = 1L,
  confidence = 0.95
)
```

Arguments

`initial_prev`	Initial prevalence of the emerging variant. Default 0.001 (0.1 percent).
`growth_rate`	Per-week multiplicative growth rate on the frequency scale. Default 1.3 (30 percent per week).
`n_per_period`	Sequences collected per surveillance period. Default 500.
`n_periods`	Maximum number of periods to evaluate. Default 26 (6 months of weekly data).
`detection_threshold`	Minimum number of sequences of the target lineage required to declare detection. Default 1.
`confidence`	Detection probability threshold. Default 0.95.

Details

At each period t, the variant prevalence is modelled as logistic growth: $p(t) = p_0 \cdot r^t / (1 - p_0 + p_0 \cdot r^t)$ where r is the per-period growth rate. The probability of detecting at least k sequences in n draws at prevalence p is $1 - F_{\text{Binom}}(k-1; n, p)$. Cumulative detection is $1 - \prod_{\tau=1}^{t}(1 - P_\tau)$.

Value

A tibble with columns `period`, `prevalence`, `detection_prob` (cumulative detection probability), and `detected` (logical). An attribute `weeks_to_detection` contains the first period where detection probability exceeds the confidence threshold, or `NA` if not reached.

See Also

sequencing_power for static sample size calculations, alert_threshold for sequential detection.

Examples

```
dh <- detection_horizon(initial_prev = 0.005, growth_rate = 1.2,
  n_per_period = 300)
dh
```

---

evaluate_prospective *Pseudo-Prospective Evaluation of Conformal Prediction*

Description

Simulates real-time deployment: at each origin, all preceding origins form the conformal calibration set, and the forecast at the next origin is evaluated. No future data is ever used for calibration — this is a genuine expanding-window evaluation.

Usage

```
evaluate_prospective(
  data,
  engine = "mlr",
  horizons = c(7L, 14L),
  min_train = 42L,
  min_cal = 5L,
  ci_level = 0.95,
  gamma = 0.05,
  ...
)
```

Arguments

data	An lfq_data object.
engine	Character; estimation engine (default "mlr").
horizons	Integer vector; forecast horizons in days (default c(7L, 14L)).
min_train	Integer; minimum training window in days (default 42).
min_cal	Integer; minimum calibration origins before conformal intervals are computed (default 5).
ci_level	Numeric in (0,1); nominal coverage (default 0.95).
gamma	Numeric; ACI learning rate (default 0.05).
...	Passed to fit_model().

Details

At each origin t_i with $i \geq$ `min_cal`:

1. Fit the model on data up to t_i.
2. Compute residuals at all previous origins $t_1, \ldots, t_{i-1}$ (the calibration set).
3. Static conformal: radius = $(1-\alpha)(1+1/n)$ quantile of absolute calibration residuals.
4. ACI: radius adjusted by online update of α_t.
5. Evaluate coverage at t_{i+1}.

This differs from `conformal_forecast()` which uses a fixed temporal split. Here, the calibration set expands over time, mimicking a surveillance agency accumulating validation data.

Value

An `lfq_prospective` S3 object (list) with:

results Tibble with origin_date, target_date, horizon, lineage, predicted, observed, lower_param, upper_param, lower_static, upper_static, lower_aci, upper_aci, radius_static, radius_aci, alpha_aci.

coverage Tibble summarising cumulative coverage by method (parametric, static conformal, ACI) at each step.

summary Tibble with method, overall coverage, mean width, Winkler score.

n_origins Total number of evaluation origins.

`filter_sparse` *Filter sparse time points and lineages*

Description

Removes time points with very low total counts and lineages observed at too few time points.

Usage

```
filter_sparse(x, min_total = 10L, min_timepoints = 3L)
```

Arguments

`x`	An lfq_data object.
`min_total`	Minimum total sequences per time point. Default 10.
`min_timepoints`	Minimum number of time points a lineage must appear to be retained. Default 3.

Value

An lfq_data object with sparse entries removed.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
filtered <- filter_sparse(sim, min_total = 100)
```

`fit_dms_prior` *Fit model with Deep Mutational Scanning priors*

Description

A penalised multinomial logistic regression engine that incorporates Deep Mutational Scanning (DMS) escape scores as informative priors on variant fitness. This is valuable for early-emergence scenarios where a new lineage has few observed sequences but laboratory-measured phenotypic data (e.g., ACE2 binding affinity, antibody escape) are available.

Usage

```
fit_dms_prior(data, dms_scores, lambda = 1, pivot = NULL, ci_level = 0.95)
```

Arguments

`data`	An `lfq_data` object.
`dms_scores`	Named numeric vector of DMS-derived fitness priors. Names correspond to lineage identifiers. Values are on the log growth rate scale (positive = fitter than average). Lineages not in the vector receive a prior of 0.
`lambda`	Regularisation strength (penalty weight). Default 1.0. Larger values pull estimates more strongly toward the DMS prior. At `lambda = 0`, the result is identical to the standard MLR engine.
`pivot`	Reference lineage name. Default `NULL` (automatic selection).
`ci_level`	Confidence level. Default 0.95.

Details

The approach uses penalised maximum likelihood where the penalty is proportional to the squared difference between the estimated growth rate and the DMS-derived prior. This implements an empirical Bayes shrinkage: with abundant data, the penalty has little effect; with sparse data, estimates are pulled toward the DMS prior.

The penalised log-likelihood is:

$$\ell_{\text{pen}}(\alpha, \delta) = \ell(\alpha, \delta) - \frac{\lambda}{2} \sum_{v} (\delta_v - \mu_v)^2$$

where ℓ is the standard multinomial log-likelihood, δ_v is the growth rate for lineage v, and μ_v is the DMS prior. The Hessian is adjusted accordingly, ensuring correct confidence interval widths.

Value

An `lfq_fit` object compatible with all downstream functions (`forecast`, `growth_advantage`, etc.).

References

Dadonaite B, Crawford KHD, Radford CE, et al. (2023). A pseudovirus system enables deep mutational scanning of the full SARS-CoV-2 spike. *Cell*, 186(6), 1263–1278. doi:10.1016/j.cell.2023.02.001

Bloom JD, Neher RA (2023). Fitness effects of mutations to SARS-CoV-2 proteins. *Virus Evolution*, 9(2), vead055. doi:10.1093/ve/vead055

See Also

`fit_model` for the standard MLR engine.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.3, "B" = 0.9),
  n_timepoints = 8, total_per_tp = 100, seed = 1)
# DMS suggests lineage A has fitness advantage
dms <- c("A" = 0.04, "B" = -0.02)
fit_dms <- fit_dms_prior(sim, dms_scores = dms, lambda = 2)
growth_advantage(fit_dms)
```

`fit_model` *Fit a lineage frequency model*

Description

Unified interface for modeling lineage frequency dynamics. Supports multiple engines that share the same input/output contract.

Usage

```
fit_model(
  data,
  engine = c("mlr", "hier_mlr", "piantham", "fga", "garw"),
  pivot = NULL,
  horizon = 28L,
  ci_level = 0.95,
  ...
)
```

Arguments

`data`	An lfq_data object.
`engine`	Model engine to use: • `"mlr"` (default): Multinomial logistic regression. Fast, frequentist, no external dependencies. • `"hier_mlr"`: Hierarchical MLR with partial pooling across locations. Requires `.location` column in data. • `"piantham"`: Piantham approximation converting MLR growth rates to relative reproduction numbers. Requires `generation_time` argument. • `"fga"`: Fixed growth advantage model (Bayesian via CmdStan). Requires 'CmdStan'; check with `lfq_stan_available()`. • `"garw"`: Growth advantage random walk model (Bayesian via CmdStan). Allows fitness to change over time.
`pivot`	Reference lineage name. Growth rates are reported relative to this lineage (fixed at 0). Default: the lineage with the highest count at the earliest time point.
`horizon`	Forecast horizon in days (stored for later use by `forecast()`). Default 28.
`ci_level`	Confidence level for intervals. Default 0.95.
`...`	Engine-specific arguments passed to the internal engine function. For `engine = "mlr"`: `window`, `ci_method`, `laplace_smooth`. For `engine = "piantham"`: `generation_time` (required). For `engine = "hier_mlr"`: `shrinkage_method`.

Details

The MLR engine models the frequency of lineage v at time t as:

$$p_v(t) = \frac{\exp(\alpha_v + \delta_v t)}{\sum_k \exp(\alpha_k + \delta_k t)}$$

where α_v is the intercept, δ_v is the growth rate per `time_scale` days (default 7), and the pivot lineage has $\alpha = \delta = 0$. Parameters are estimated by maximum likelihood via `optim(method = "BFGS")` with the Hessian used for asymptotic Wald confidence intervals.

The constant growth rate assumption is appropriate for monotonic variant replacement periods (typically 2–4 months). For longer periods or non-monotonic dynamics, use the `window` argument to restrict the estimation window, or consider the `"garw"` engine which allows time-varying growth advantages.

Value

An `lfq_fit` object (S3 class), a list containing:

engine Engine name (character).

growth_rates Named numeric vector of growth rates per `time_scale` days (pivot = 0).

intercepts Named numeric vector of intercepts.

pivot Name of pivot lineage.

lineages Character vector of all lineage names.

fitted_values Tibble of fitted frequencies.

residuals Tibble with observed, fitted, Pearson residuals.

vcov_matrix Variance-covariance matrix.

loglik, aic, bic Model fit statistics.

nobs, n_timepoints, df Sample and model sizes.

ci_level, horizon As specified.

call The matched call.

References

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

Piantham C, Linton NM, Nishiura H (2022). Predicting the trajectory of replacements of SARS-CoV-2 variants using relative reproduction numbers. *Viruses*, 14(11):2556. doi:10.3390/v14112556

See Also

`growth_advantage()` to extract fitness estimates, `forecast()` for frequency prediction, `backtest()` for rolling-origin evaluation.

Examples

```
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42
)
fit <- fit_model(sim, engine = "mlr")
fit
```

`fitness_decomposition` *Decompose variant fitness into transmissibility and immune escape*

Description

Partitions the observed growth advantage of each lineage into two components: intrinsic transmissibility (fitness in a fully naive population) and immune escape (additional fitness gained from evading population immunity). This decomposition follows the framework of Figgins and Bedford (2025), where the effective fitness of lineage v is modelled as $f_v(t) = \beta_v \cdot (1 - \pi_v(t))$ with β_v representing intrinsic transmissibility and $\pi_v(t)$ the proportion of the population with neutralising immunity against v.

Usage

```
fitness_decomposition(fit, landscape, generation_time)
```

Arguments

`fit`	An `lfq_fit` object from `fit_model`.
`landscape`	An `immune_landscape` object from `immune_landscape`.
`generation_time`	Generation time in days (required for converting growth rates to reproduction numbers).

Details

The decomposition proceeds as follows. For each non-pivot lineage, the observed growth rate δ_v from the MLR fit is taken as the total fitness difference relative to the reference. The immune escape component is estimated from the immunity differential:

$$\text{escape}_v = -\log(1 - \bar{\pi}_v) + \log(1 - \bar{\pi}_{\text{ref}})$$

where $\bar{\pi}_v$ is the mean population immunity against lineage v over the fitted period. The intrinsic transmissibility component is the residual:

$$\beta_v = \delta_v - \text{escape}_v$$

When cross-immunity data are available in the landscape object, effective immunity against each lineage is computed as the weighted sum of immunity sources, discounted by the cross-immunity matrix.

Value

A `fitness_decomposition` object (S3 class) with components:

decomposition Tibble with columns: `lineage`, `observed_advantage` (total growth rate), `beta` (intrinsic transmissibility), `escape_contribution` (immune escape component), `transmissibility_fraction` (proportion due to intrinsic fitness), `escape_fraction` (proportion due to immune escape).

fit The input lfq_fit object.

landscape The input immune_landscape.

generation_time Generation time used.

References

Figgins MD, Bedford T (2025). Jointly modeling variant frequencies and case counts to estimate relative variant severity. *medRxiv*. doi:10.1101/2024.12.02.24318334

See Also

`immune_landscape` for constructing the immunity input, `selective_pressure` for population-level early warning signals.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.3, "B" = 0.9),
  n_timepoints = 15, seed = 1)
fit <- fit_model(sim, engine = "mlr")

imm_data <- data.frame(
  date = rep(unique(sim$.date), each = 3),
  lineage = rep(c("A", "B", "ref"), length(unique(sim$.date))),
  immunity = c(rep(c(0.2, 0.5, 0.4),
    length(unique(sim$.date))))
)
il <- immune_landscape(imm_data, date, lineage, immunity)
fd <- fitness_decomposition(fit, il, generation_time = 5)
fd
```

---

forecast *Forecast lineage frequencies (generic)*

Description

Forecast lineage frequencies (generic)

Usage

```
forecast(object, ...)
```

Arguments

object	A model object.
...	Additional arguments passed to methods.

Value

A forecast object.

`forecast.lfq_fit` *Forecast lineage frequencies*

Description

Projects lineage frequencies forward in time using the fitted model. Prediction uncertainty is quantified by parametric simulation from the estimated parameter distribution.

Usage

```
## S3 method for class 'lfq_fit'
forecast(
  object,
  horizon = 28L,
  ci_level = 0.95,
  n_sim = 1000L,
  sampling_noise = TRUE,
  effective_n = 100L,
  ...
)
```

Arguments

`object`	An `lfq_fit` object.
`horizon`	Number of days to forecast. Default 28 (4 weeks).
`ci_level`	Confidence level for prediction intervals. Default 0.95.
`n_sim`	Number of parameter draws for prediction intervals. Default 1000.
`sampling_noise`	Logical; add multinomial sampling noise to prediction intervals? Default `TRUE`. When `TRUE`, each draw includes both parameter uncertainty (from the MLE covariance) and observation-level multinomial sampling variability.
`effective_n`	Effective sample size for multinomial sampling noise. Default 100, corresponding to a typical weekly sequencing volume per reporting unit. Smaller values produce wider (more conservative) prediction intervals.
`...`	Unused.

Value

An `lfq_forecast` object (tibble subclass) with columns:

.date Date.

.lineage Lineage name.

.median Median predicted frequency.

.lower Lower prediction bound.

.upper Upper prediction bound.

.type "fitted" or "forecast".

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim, engine = "mlr")
fc <- forecast(fit, horizon = 21)
fc
```

`glance.lfq_fit` *Glance at an lfq_fit object*

Description

Returns a single-row tibble of model-level summary statistics.

Usage

```
glance.lfq_fit(x, ...)
```

Arguments

`x`	An `lfq_fit` object.
`...`	Ignored.

Value

A single-row tibble with columns: `engine`, `n_lineages`, `n_timepoints`, `nobs`, `df`, `logLik`, `AIC`, `BIC`, `pivot`, `convergence`.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
glance.lfq_fit(fit)
```

`growth_advantage` *Extract growth advantage estimates*

Description

Computes relative fitness of each lineage from a fitted model. Supports four output formats for different use cases.

Usage

```
growth_advantage(
  fit,
  type = c("growth_rate", "relative_Rt", "selection_coefficient", "doubling_time"),
  generation_time = NULL,
  ci_level = NULL
)
```

Arguments

`fit`	An `lfq_fit` object returned by `fit_model()`.
`type`	Output type: • `"growth_rate"` (default): raw growth rate delta per `time_scale` days (typically per week). • `"relative_Rt"`: relative effective reproduction number. Requires `generation_time`. • `"selection_coefficient"`: relative Rt minus 1. Requires `generation_time`. • `"doubling_time"`: days for frequency ratio vs pivot to double (positive = growing) or halve (negative = declining).
`generation_time`	Mean generation time in days. Required for `type = "relative_Rt"` and `"selection_coefficient"`. Common values: SARS-CoV-2 approximately 5 days, influenza approximately 3 days.
`ci_level`	Confidence level for intervals. Default uses the level from the fitted model.

Details

The `"relative_Rt"` and `"selection_coefficient"` types use the Piantham approximation (Piantham et al. 2022, doi:10.3390/v14112556), which assumes:

1. Variants are in their exponential growth phase (not saturating due to population-level immunity).
2. All variants share the same generation time distribution.
3. Growth advantage reflects transmissibility differences; immune escape is not separately identified.

Confidence intervals for `"relative_Rt"` are computed in log-space (Wald intervals on log-Rt), which is more accurate than the linear delta method for ratios.

Value

A tibble with columns:

lineage Lineage name.

estimate Point estimate.

lower Lower confidence bound.

upper Upper confidence bound.

type Type of estimate.

pivot Name of pivot (reference) lineage.

References

Piantham C, Linton NM, Nishiura H (2022). Predicting the trajectory of replacements of SARS-CoV-2 variants using relative reproduction numbers. *Viruses*, 14(11):2556. doi:10.3390/v14112556

Abousamra E, Figgins M, Bedford T (2024). Fitness models provide accurate short-term forecasts of SARS-CoV-2 variant frequency. *PLoS Computational Biology*, 20(9):e1012443. doi:10.1371/journal.pcbi.1012443

Examples

```
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42
)
fit <- fit_model(sim, engine = "mlr")

# Growth rates per week
growth_advantage(fit, type = "growth_rate")

# Relative Rt (needs generation time)
growth_advantage(fit, type = "relative_Rt", generation_time = 5)
```

---

immune_landscape *Construct a population immunity landscape*

Description

Assembles time-varying population immunity estimates against each circulating lineage from seroprevalence surveys, vaccination records, or infection history data. The resulting object serves as input to `fitness_decomposition` for disentangling intrinsic transmissibility from immune escape.

Usage

```
immune_landscape(
  data,
  date,
  lineage,
  immunity,
  type = "combined",
  cross_immunity = NULL
)
```

Arguments

`data`	A data frame with columns for date, lineage, and immunity level.
`date`	Column name (unquoted or string) containing dates.
`lineage`	Column name containing lineage/variant identifiers.
`immunity`	Column name containing population-level immunity estimates (proportion, 0–1 scale) against each lineage.
`type`	Character vector specifying immunity source(s): `"infection"`, `"vaccine"`, `"hybrid"`, or `"combined"` (default). If the data contain a `type` column, per-source estimates are preserved; otherwise all estimates are treated as combined.
`cross_immunity`	Optional numeric matrix of cross-immunity between lineages. Rows and columns correspond to lineages; entry (i, j) is the degree to which immunity against lineage i protects against lineage j (0 to 1). Default `NULL` assumes no cross-protection data.

Details

Immunity estimates may come from multiple sources. Seroprevalence surveys provide direct measurements but are infrequent and geographically sparse. Vaccination coverage data are more widely available but do not capture waning or variant-specific escape. Model-based reconstructions (e.g., from case and death data) can fill gaps but introduce model dependence.

`immune_landscape()` accepts any of these as input. The critical requirement is that immunity is expressed on a 0–1 scale representing the proportion of the population with neutralising protection against each lineage at each time point.

Value

An `immune_landscape` object (S3 class) with components:

estimates Tibble with columns `date`, `lineage`, `immunity`, and optionally `type`.

lineages Character vector of lineage names.

date_range Two-element Date vector.

cross_immunity Matrix or NULL.

See Also

`fitness_decomposition` for downstream analysis.

Examples

```
# Simulated immunity data
imm_data <- data.frame(
  date = rep(seq(as.Date("2024-01-01"), by = "week",
                 length.out = 10), each = 3),
  lineage = rep(c("BA.5", "XBB.1.5", "JN.1"), 10),
  immunity = c(
    rep(c(0.6, 0.4, 0.1), 5),
    rep(c(0.55, 0.45, 0.25), 5))
)
il <- immune_landscape(imm_data, date = date,
  lineage = lineage, immunity = immunity)
il
```

---

influenza_h3n2 *Simulated influenza A/H3N2 clade frequency data*

Description

A simulated dataset of weekly influenza A/H3N2 clade sequence counts over a single Northern Hemisphere season (24 weeks).

Usage

```
influenza_h3n2
```

Format

A data frame with 96 rows and 4 columns:

date Collection date (Date, weekly).

clade Clade name (character).

count Number of sequences (integer).

total Total sequences for this week (integer).

Source

Simulated based on 'Nextstrain' influenza clade dynamics.

Examples

```
data(influenza_h3n2)
x <- lfq_data(influenza_h3n2, lineage = clade,
              date = date, count = count, total = total)
x
```

is_lfq_data *Test if an object is an lfq_data object*

Description

Test if an object is an lfq_data object

Usage

```
is_lfq_data(x)
```

Arguments

x	Object to test.

Value

Logical scalar.

Examples

```
d <- data.frame(date = Sys.Date(), lineage = "A", count = 10)
x <- lfq_data(d, lineage = lineage, date = date, count = count)
is_lfq_data(x)
is_lfq_data(d)
```

lfq_advantage *Pipe-friendly growth advantage extraction*

Description

Pipe-friendly growth advantage extraction

Usage

```
lfq_advantage(fit, type = "relative_Rt", generation_time = NULL, ...)
```

Arguments

fit	An lfq_fit object.
type	Output type. Default "relative_Rt".
generation_time	Mean generation time in days.
...	Passed to growth_advantage().

Value

A tibble of growth advantages.

`lfq_data` *Create a lineage frequency data object*

Description

Validates, structures, and annotates lineage count data for downstream modeling and analysis. This is the entry point for all lineagefreq workflows.

Usage

```
lfq_data(
  data,
  lineage,
  date,
  count,
  total = NULL,
  location = NULL,
  min_total = 10L
)
```

Arguments

`data`	A data frame containing at minimum columns for lineage identity, date, and count.
`lineage`	<tidy-select> Column containing lineage/variant identifiers (character or factor).
`date`	<tidy-select> Column containing collection dates (`Date` class or parseable character).
`count`	<tidy-select> Column containing sequence counts (non-negative integers).
`total`	<tidy-select> Optional column of total sequences per date-location. If `NULL`, computed as the sum of `count` per group.
`location`	<tidy-select> Optional column for geographic stratification.
`min_total`	Minimum total count per time point. Time points below this are flagged as unreliable. Default 10.

Details

Performs the following validation and processing:

1. Checks that all required columns exist and have correct types.
2. Coerces character dates to Date via ISO 8601 parsing.
3. Ensures counts are non-negative integers.

4. Replaces NA counts with 0 (with warning).
5. Aggregates duplicate lineage-date rows by summing (with warning).
6. Computes per-time-point totals and frequencies.
7. Flags time points below `min_total` as unreliable.
8. Sorts by date ascending, then lineage alphabetically.

Value

An `lfq_data` object (a tibble subclass) with standardized columns:

`.lineage` Lineage identifier (character).

`.date` Collection date (Date).

`.count` Sequence count (integer).

`.total` Total sequences at this time point (integer).

`.freq` Observed frequency (numeric).

`.reliable` Logical; `TRUE` if `.total` >= `min_total`.

`.location` Location, if provided (character).

All original columns are preserved.

Examples

```
d <- data.frame(
  date = rep(seq(as.Date("2024-01-01"), by = "week",
                 length.out = 8), each = 3),
  lineage = rep(c("JN.1", "KP.3", "Other"), 8),
  n = c(5, 2, 93, 12, 5, 83, 28, 11, 61, 50, 20, 30,
        68, 18, 14, 80, 12, 8, 88, 8, 4, 92, 5, 3)
)
x <- lfq_data(d, lineage = lineage, date = date, count = n)
x
```

---

lfq_engines *List available modeling engines*

Description

Returns information about all modeling engines available in lineagefreq. Core engines (mlr, hier_mlr, piantham) are always available. Bayesian engines (fga, garw) require 'CmdStan'.

Usage

```
lfq_engines()
```

Value

A tibble with columns: `engine`, `type`, `time_varying`, `available`, `description`.

Examples

```
lfq_engines()
```

`lfq_fit` *Pipe-friendly model fitting*

Description

Enables tidyverse-style chaining: `data |> lfq_fit("mlr") |> lfq_forecast(28) |> lfq_score()`

Usage

```
lfq_fit(data, engine = "mlr", ...)
```

Arguments

`data`	An lfq_data object.
`engine`	Engine name. Default `"mlr"`.
`...`	Passed to `fit_model()`.

Value

An `lfq_fit` object.

Examples

```
data(sarscov2_us_2022)
sarscov2_us_2022 |>
  lfq_data(lineage = variant, date = date, count = count, total = total) |>
  lfq_fit("mlr") |>
  lfq_advantage(generation_time = 5)
```

lfq_forecast *Pipe-friendly forecasting*

Description

Pipe-friendly forecasting

Usage

```
lfq_forecast(fit, horizon = 28L, ...)
```

Arguments

fit	An lfq_fit object.
horizon	Forecast horizon in days. Default 28.
...	Passed to forecast().

Value

An lfq_forecast object.

lfq_score *Pipe-friendly backtesting + scoring*

Description

Runs backtest and returns scores in one step.

Usage

```
lfq_score(
  data,
  engines = "mlr",
  horizons = c(14, 28),
  metrics = c("mae", "coverage"),
  ...
)
```

Arguments

data	An lfq_data object.
engines	Character vector of engine names.
horizons	Forecast horizons in days.
metrics	Score metrics.
...	Passed to backtest().

Value

A tibble of scores.

`lfq_stan_available` *Check if 'CmdStan' backend is available*

Description

Returns `TRUE` if 'CmdStanR' and 'CmdStan' are installed. Bayesian engines require this.

Usage

```
lfq_stan_available()
```

Value

Logical scalar.

Examples

```
lfq_stan_available()
```

`lfq_summary` *Convert lfq_fit results to a summary tibble*

Description

Returns a one-row-per-lineage summary with growth rates, fitted frequencies at first and last time points, and growth advantage in multiple scales.

Usage

```
lfq_summary(fit, generation_time = NULL)
```

Arguments

`fit`	An `lfq_fit` object.
`generation_time`	Generation time for Rt calculation.

Value

A tibble.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
lfq_summary(fit, generation_time = 5)
```

`lfq_version` *Package version and system information*

Description

Reports lineagefreq version and availability of optional backends. Useful for reproducibility and bug reports.

Usage

```
lfq_version()
```

Value

A list with components: `version`, `r_version`, `stan_available`, `engines`.

Examples

```
lfq_version()
```

`plot.adaptive_allocation` *Plot adaptive allocation*

Description

Plot adaptive allocation

Usage

```
## S3 method for class 'adaptive_allocation'
plot(x, ...)
```

Arguments

`x`	An `adaptive_allocation` object.
`...`	Unused.

Value

A ggplot object.

---

plot.calibration_report *Plot calibration diagnostics*

Description

Produces either a reliability diagram (default) or a PIT histogram.

Usage

```
## S3 method for class 'calibration_report'
plot(x, type = c("reliability", "pit"), ...)
```

Arguments

x	A calibration_report object.
type	Which panel to display: "reliability" (default) or "pit" for the PIT histogram.
...	Unused.

Value

A ggplot object.

---

plot.evoi *Plot EVOI curve*

Description

Plot EVOI curve

Usage

```
## S3 method for class 'evoi'
plot(x, ...)
```

Arguments

x	An evoi object.
...	Unused.

Value

A ggplot object.

plot.fitness_decomposition *Plot fitness decomposition*

Description

Stacked bar plot showing the partition of growth advantage into intrinsic transmissibility and immune escape components.

Usage

```
## S3 method for class 'fitness_decomposition'
plot(x, ...)
```

Arguments

x	A fitness_decomposition object.
...	Unused.

Value

A ggplot object.

plot.immune_landscape *Plot population immunity landscape*

Description

Plot population immunity landscape

Usage

```
## S3 method for class 'immune_landscape'
plot(x, ...)
```

Arguments

x	An immune_landscape object.
...	Unused.

Value

A ggplot object.

plot.lfq_prospective *Plot Prospective Evaluation Results*

Description

Plot Prospective Evaluation Results

Usage

```
## S3 method for class 'lfq_prospective'
plot(x, type = c("coverage", "radius", "comparison"), ...)
```

Arguments

x	An lfq_prospective object.
type	Character; one of "coverage" (cumulative coverage over time), "radius" (conformal radius over time), or "comparison" (retrospective vs prospective).
...	Ignored.

Value

A ggplot object.

plot_backtest *Plot backtest scores*

Description

Creates a panel plot of forecast accuracy by engine and horizon.

Usage

```
plot_backtest(scores)
```

Arguments

scores	Output of score_forecasts().

Value

A ggplot object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
sc <- score_forecasts(bt)
plot_backtest(sc)
```

---

print.lfq_fit *Print a lineage frequency model*

Description

Print a lineage frequency model

Usage

```
## S3 method for class 'lfq_fit'
print(x, ...)
```

Arguments

x	An `lfq_fit` object.
...	Ignored.

Value

Invisibly returns `x`.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42)
fit <- fit_model(sim, engine = "mlr")
print(fit)
```

`read_lineage_counts` *Read lineage count data from a CSV file*

Description

A convenience wrapper for reading surveillance count data from CSV files into lfq_data format. Expects a file with at least columns for date, lineage, and count.

Usage

```
read_lineage_counts(
  file,
  date = "date",
  lineage = "lineage",
  count = "count",
  ...
)
```

Arguments

`file`	Path to CSV file.
`date`	Name of the date column. Default `"date"`.
`lineage`	Name of the lineage column. Default `"lineage"`.
`count`	Name of the count column. Default `"count"`.
`...`	Additional arguments passed to `lfq_data()`.

Value

An lfq_data object.

Examples

```
# Read the bundled example CSV
f <- system.file("extdata", "example_counts.csv",
                 package = "lineagefreq")
x <- read_lineage_counts(f)
x
```

`recalibrate` *Recalibrate prediction intervals*

Description

Applies post-hoc recalibration to improve the coverage properties of prediction intervals from `forecast`. Two methods are available: isotonic regression (nonparametric, monotonicity- preserving) and Platt scaling (logistic, parametric).

Usage

```
recalibrate(forecast_obj, bt, method = c("isotonic", "platt"))
```

Arguments

`forecast_obj`	An `lfq_forecast` object to recalibrate.
`bt`	An `lfq_backtest` object providing the calibration data. The mapping from nominal to empirical coverage is learned from the backtest residuals.
`method`	Recalibration method: `"isotonic"` (default) for isotonic regression on the empirical coverage function, or `"platt"` for Platt scaling via logistic regression.

Details

Isotonic regression: Learns the monotone mapping from nominal coverage levels to observed coverage using the backtest data, then inverts it to find the nominal level that achieves the desired empirical coverage. This is a nonparametric approach that requires no distributional assumptions.

Platt scaling: Fits a logistic regression of the form $P(\text{covered}) = \text{logit}^{-1}(a \cdot z + b)$ where z is the standardised residual, and uses the fitted model to adjust prediction interval widths.

Both methods require a backtest object with sufficient data to estimate the calibration mapping reliably (at least 30 forecast-observation pairs recommended).

Value

An `lfq_forecast` object with recalibrated `.lower` and `.upper` bounds. The object retains all attributes of the original forecast and can be passed to `autoplot`.

References

Platt JC (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in Large Margin Classifiers*, 61–74.

See Also

`calibrate` for calibration diagnostics.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
fit <- fit_model(sim, engine = "mlr")
fc  <- forecast(fit, horizon = 21)
bt  <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
fc_recal <- recalibrate(fc, bt, method = "isotonic")
```

`register_engine` *Register a custom modeling engine*

Description

Allows third-party packages to register custom engines with `fit_model()`. This enables an extensible plugin architecture similar to 'parsnip' engine registration.

Usage

```
register_engine(
  name,
  fit_fn,
  description = "",
  type = "frequentist",
  time_varying = FALSE
)
```

Arguments

`name`	Engine name (character scalar).
`fit_fn`	Function with signature `function(data, pivot, ci_level, ...)`. Must return a list compatible with `lfq_fit` structure.
`description`	One-line description of the engine.
`type`	`"frequentist"` or `"bayesian"`.
`time_varying`	Logical; does the engine support time-varying growth advantages?

Value

Invisibly returns the updated engine registry.

Examples

```
# Register a custom engine
my_engine <- function(data, pivot = NULL, ci_level = 0.95, ...) {
  # Custom implementation...
  .engine_mlr(data, pivot = pivot, ci_level = ci_level, ...)
}
register_engine("my_mlr", my_engine, "Custom MLR wrapper")
lfq_engines()
```

---

sarscov2_us_2022 *Simulated SARS-CoV-2 variant frequency data (US, 2022)*

Description

A simulated dataset of weekly SARS-CoV-2 variant sequence counts for the United States in 2022. Includes the BA.1 to BA.2 to BA.4/5 to BQ.1 transition dynamics. This is simulated data (not real GISAID data) to avoid license restrictions while preserving realistic statistical properties.

Usage

```
sarscov2_us_2022
```

Format

A data frame with 200 rows and 4 columns:

date Collection date (Date, weekly).

variant Variant name (character): BA.1, BA.2, BA.4/5, BQ.1, Other.

count Number of sequences assigned to this variant in this week (integer).

total Total sequences for this week (integer).

Source

Simulated based on parameters from published CDC MMWR genomic surveillance reports and 'Nextstrain' public data.

Examples

```
data(sarscov2_us_2022)
x <- lfq_data(sarscov2_us_2022, lineage = variant,
              date = date, count = count, total = total)
x
```

`score_forecasts` *Score backtest forecast accuracy*

Description

Computes standardized accuracy metrics from backtesting results.

Usage

```
score_forecasts(
  bt,
  metrics = c("mae", "rmse", "coverage", "wis", "crps", "log_score", "dss",
    "calibration")
)
```

Arguments

`bt`	An `lfq_backtest` object from `backtest()`.
`metrics`	Character vector of metrics to compute: • `"mae"`: Mean absolute error of frequency. • `"rmse"`: Root mean squared error. • `"coverage"`: Proportion within prediction intervals. • `"wis"`: Simplified weighted interval score for the single prediction interval stored in the backtest (typically 95%). • `"crps"`: Continuous Ranked Probability Score, assuming Gaussian forecast distribution (Gneiting and Raftery, 2007). • `"log_score"`: Logarithmic scoring rule evaluated at the observed value under the Gaussian forecast density. • `"dss"`: Dawid-Sebastiani Score, a proper scoring rule based on the predictive mean and variance. • `"calibration"`: Mean squared calibration error across nominal coverage levels 10\

Value

A tibble with columns: `engine`, `horizon`, `metric`, `value`.

References

Bracher J, Ray EL, Gneiting T, Reich NG (2021). Evaluating epidemic forecasts in an interval format. *PLoS Computational Biology*, 17(2):e1008618. doi:10.1371/journal.pcbi.1008618

Gneiting T, Raftery AE (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, 102(477), 359–378. doi:10.1198/016214506000001437

See Also

compare_models() to rank engines based on scores.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8),
  n_timepoints = 20, seed = 1)
bt <- backtest(sim, engines = "mlr",
  horizons = c(7, 14), min_train = 42)
score_forecasts(bt)
```

---

selective_pressure *Population-level selective pressure from variant dynamics*

Description

Computes a population-level selective pressure metric from genomic surveillance data alone, without requiring case counts or epidemiological data. The metric quantifies how rapidly the variant landscape is shifting and serves as an early warning signal for epidemic growth that is robust to case underreporting.

Usage

```
selective_pressure(fit, method = c("mean", "variance"))
```

Arguments

fit	An lfq_fit object from fit_model.
method	Aggregation method: "mean" (default) for the frequency-weighted mean growth rate, or "variance" for the frequency-weighted variance of growth rates (higher variance indicates stronger selection).

Details

The approach follows the framework of Figgins and Bedford (2025), where selective pressure is defined as the variance-weighted mean growth rate across circulating lineages:

$$S(t) = \sum_v p_v(t) \cdot \delta_v$$

This represents the expected rate at which the population-average fitness is increasing, measured entirely from sequence data.

When method = "mean", the metric is positive when fitter- than-average lineages are increasing in frequency, indicating population-level adaptation. A sustained positive value precedes epidemic

growth because it means the effective reproduction number of the average circulating virus is increasing.

When `method = "variance"`, the metric captures the heterogeneity of fitness across co-circulating lineages. High variance indicates strong directional selection; low variance indicates near-neutral drift.

This metric requires only genomic surveillance data. It does not require case counts, hospitalisations, or wastewater data, making it applicable in settings where epidemiological reporting is incomplete or delayed.

Value

A tibble with columns:

date Date.

pressure Selective pressure value.

dominant_lineage Lineage with highest frequency.

dominant_freq Frequency of dominant lineage.

References

Figgins MD, Bedford T (2025). Jointly modeling variant frequencies and case counts to estimate relative variant severity. *medRxiv*. doi:10.1101/2024.12.02.24318334

Examples

```
sim <- simulate_dynamics(n_lineages = 4,
  advantages = c("A" = 1.4, "B" = 1.1, "C" = 0.8),
  n_timepoints = 15, seed = 1)
fit <- fit_model(sim, engine = "mlr")
sp <- selective_pressure(fit)
sp
```

---

sequencing_power *Sequencing power analysis*

Description

Estimates the minimum number of sequences needed to detect a lineage at a given frequency with specified precision.

Usage

```
sequencing_power(target_precision = 0.05, current_freq = 0.02, ci_level = 0.95)
```

Arguments

target_precision	Desired half-width of the frequency confidence interval. Default 0.05 (plus/minus 5 percentage points).
current_freq	True or assumed frequency of the target lineage. Can be a vector for multiple scenarios. Default 0.02 (2%).
ci_level	Confidence level. Default 0.95.

Details

Uses the normal approximation to the binomial:

$$n = z^2 \cdot p(1-p)/E^2$$

where z is the critical value, p is frequency, E is precision.

Value

A tibble with columns: `current_freq`, `target_precision`, `required_n`, `ci_level`.

Examples

```
# How many sequences to estimate a 2% lineage within +/-5%?
sequencing_power()

# Multiple scenarios
sequencing_power(current_freq = c(0.01, 0.02, 0.05, 0.10))
```

---

simulate_dynamics *Simulate lineage frequency dynamics*

Description

Generates synthetic lineage frequency data under a multinomial sampling model with configurable growth advantages. Useful for model validation, power analysis, and teaching.

Usage

```
simulate_dynamics(
  n_lineages = 4L,
  n_timepoints = 20L,
  total_per_tp = 500L,
  advantages = NULL,
  reference_name = "ref",
  start_date = Sys.Date(),
  interval = 7L,
  overdispersion = NULL,
  seed = NULL
)
```

Arguments

n_lineages	Number of lineages including reference. Default 4.
n_timepoints	Number of time points. Default 20.
total_per_tp	Sequences per time point. A single integer (constant across time) or a vector of length n_timepoints (variable sampling effort). Default 500.
advantages	Named numeric vector of per-week multiplicative growth advantages for non-reference lineages. Length must equal n_lineages - 1. Values > 1 mean growing faster than reference, < 1 means declining. If NULL, random values in (0.8, 1.5).
reference_name	Name of the reference lineage. Default "ref".
start_date	Start date for the time series. Default today.
interval	Days between consecutive time points. Default 7 (weekly data).
overdispersion	If NULL (default), standard multinomial sampling. If a positive number, Dirichlet-Multinomial sampling with concentration = 1 / overdispersion. Larger values = more overdispersion.
seed	Random seed for reproducibility.

Value

An lfq_data object with an additional true_freq column containing the true (pre-sampling) frequencies.

Examples

```
# JN.1 grows 1.3x per week, KP.3 declines at 0.9x
sim <- simulate_dynamics(
  n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15,
  seed = 42
)
sim
```

---

summarize_emerging *Summarize emerging lineages*

Description

Tests whether each lineage's frequency is significantly increasing over time using a binomial GLM. Useful for early warning of lineages that may warrant enhanced surveillance.

Usage

```
summarize_emerging(data, threshold = 0.01, min_obs = 3L, p_adjust = "holm")
```

Arguments

data	An lfq_data object.
threshold	Minimum current frequency to test. Default 0.01.
min_obs	Minimum time points observed. Default 3.
p_adjust	P-value adjustment method. Default "holm".

Value

A tibble with columns: lineage, first_seen, last_seen, n_timepoints, current_freq, growth_rate, p_value, p_adjusted, significant, direction.

Examples

```
sim <- simulate_dynamics(
  n_lineages = 4,
  advantages = c(emerging = 1.5, stable = 1.0, declining = 0.7),
  n_timepoints = 12, seed = 42)
summarize_emerging(sim)
```

---

summary.lfq_fit *Summarise a lineage frequency model*

Description

Summarise a lineage frequency model

Usage

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

Arguments

object	An lfq_fit object.
...	Ignored.

Value

Invisibly returns object.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("JN.1" = 1.3, "KP.3" = 0.9),
  n_timepoints = 15, seed = 42)
fit <- fit_model(sim, engine = "mlr")
summary(fit)
```

surveillance_dashboard *Comprehensive surveillance quality dashboard*

Description

Produces a multi-panel display combining calibration diagnostics, detection power, estimation quality, and current variant landscape into a single figure suitable for weekly surveillance reports. Designed for programme managers rather than statisticians.

Usage

```
surveillance_dashboard(fit, data, bt = NULL, target_prevalence = 0.01)
```

Arguments

fit	An lfq_fit object from fit_model.
data	The lfq_data object used for fitting.
bt	Optional lfq_backtest object for calibration panel. If NULL, the calibration panel is omitted.
target_prevalence	Prevalence for detection power calculation. Default 0.01 (1 percent).

Details

The dashboard contains up to four panels: (1) current frequency landscape, (2) growth advantage forest plot, (3) detection power curve, and (4) calibration reliability diagram (if backtest data are provided).

Value

A list of ggplot objects with class surveillance_dashboard. A print method renders all panels.

See Also

surveillance_value for EVOI analysis, alert_threshold for sequential detection.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.3, "B" = 0.9),
  n_timepoints = 15, seed = 1)
fit <- fit_model(sim, engine = "mlr")
panels <- surveillance_dashboard(fit, sim)
```

surveillance_value *Expected Value of Information for genomic surveillance*

Description

Quantifies the marginal decision value of sequencing additional samples from each region or stratum, based on the current posterior uncertainty of variant frequency estimates. The EVOI captures how much the expected estimation error decreases per additional sequence, enabling cost-effective resource allocation.

Usage

```
surveillance_value(
  fit,
  n_current = 100L,
  n_additional = seq(10L, 200L, by = 10L),
  target_lineage = NULL
)
```

Arguments

`fit`	An `lfq_fit` object from `fit_model`.
`n_current`	Integer vector of current sample sizes per stratum. If a single integer, assumed equal across all lineages.
`n_additional`	Integer vector of candidate additional sample sizes to evaluate. Default `seq(10, 200, by = 10)`.
`target_lineage`	Optional character; compute EVOI specifically for this lineage. Default `NULL` evaluates across all non-pivot lineages.

Details

The EVOI is computed as the expected reduction in mean squared estimation error for lineage frequency when n additional sequences are observed. Under the multinomial likelihood with Gaussian approximation to the posterior, the variance of the frequency estimate scales as $p(1-p)/n$, and additional samples reduce this by the factor $n_0/(n_0 + n_{\text{add}})$.

The marginal EVOI (the value of one additional sequence) is the derivative of the EVOI curve. It decreases monotonically with sample size, exhibiting the diminishing returns characteristic of information-theoretic quantities.

Value

An `evoi` S3 class with components:

values Tibble with `n_additional`, `evoi`, `marginal_evoi` columns.

current_uncertainty Numeric; current estimation variance (sum of growth rate SEs squared).

target_lineage Character or NULL.

See Also

adaptive_design for dynamic allocation, sequencing_power for static sample size planning.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.3, "B" = 0.9),
  n_timepoints = 15, seed = 1)
fit <- fit_model(sim, engine = "mlr")
ev <- surveillance_value(fit, n_current = 500)
ev
```

---

tidy.fitness_decomposition *Tidy a fitness decomposition*

Description

Tidy a fitness decomposition

Usage

```
tidy.fitness_decomposition(x, ...)
```

Arguments

x	A fitness_decomposition object.
...	Unused.

Value

A tibble with the decomposition results.

---

tidy.lfq_fit *Tidy an lfq_fit object*

Description

Converts model results to a tidy tibble, compatible with the broom package ecosystem.

Usage

```
tidy.lfq_fit(x, conf.int = TRUE, conf.level = 0.95, ...)
```

Arguments

x	An lfq_fit object.
conf.int	Include confidence intervals? Default TRUE.
conf.level	Confidence level. Default 0.95.
...	Ignored.

Value

A tibble with columns: lineage, term, estimate, std.error, conf.low, conf.high.

Examples

```
sim <- simulate_dynamics(n_lineages = 3,
  advantages = c("A" = 1.2, "B" = 0.8), seed = 1)
fit <- fit_model(sim)
tidy.lfq_fit(fit)
```

---

unregister_engine *Remove a registered engine*

Description

Remove a registered engine

Usage

```
unregister_engine(name)
```

Arguments

name	Engine name to remove.

Value

Invisibly returns the updated registry.

Index