Package: seqcomp (via r-universe)

June 30, 2026

Type Package

Title Sequential Comparison of Probabilistic Forecasts

Version 0.1.0

Description Implements tools for sequential comparison of probabilistic forecasts, including binary and categorical scoring rules, anytime-valid confidence sequences, e-processes, Winkler-score comparisons, lag handling, and predictable-bound e-processes.

License MIT + file LICENSE

URL https://github.com/alasgarliakbar/seqcomp

BugReports https://github.com/alasgarliakbar/seqcomp/issues

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports lamW

Suggests scoringRules, VGAM, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

VignetteBuilder knitr

NeedsCompilation no

Author Akbar Alasgarli [aut, cre]

Maintainer Akbar Alasgarli <alasgarliakbar@gmail.com>

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

Date/Publication 2026-06-30 12:20:02 UTC

RemoteUrl https://github.com/cran/seqcomp

RemoteRef HEAD

RemoteSha 55934cbcba1621f4eb1dab587e24ef9cefebdd1c

Contents

seqcomp-package . . . 2
brier_score . . . 4
calibrate_p_to_e . . . 5
cm_boundary . . . 6
compare_forecasts . . . 6
crps_empirical . . . 9
crps_normal . . . 10
crps_std . . . 10
cs_asymptotic . . . 11
cs_bernstein . . . 12
cs_hoeffding . . . 13
eprocess . . . 15
eprocess_lag . . . 16
eprocess_predictable . . . 18
eprocess_rejections . . . 19
ge_boundary . . . 20
log_score . . . 21
predictable_rejections . . . 22
ps_boundary . . . 23
qlike_score . . . 24
rho_from_vopt . . . 25
score_bounds . . . 26
score_diff_scales . . . 27
spherical_score . . . 28
split_streams . . . 29
tick_loss . . . 29
unroll_stream . . . 30
winkler_compare . . . 31
winkler_cs . . . 32
winkler_etest . . . 33
winkler_score . . . 34

Index **36**

`seqcomp-package` *seqcomp: Sequential Comparison of Probabilistic Forecasts*

Description

`seqcomp` provides tools for comparing probabilistic forecasters sequentially, following the anytime-valid framework of Choe and Ramdas (2024).

Details

The package is built around the score difference

$$\hat{\delta}_t = S(p_t, y_t) - S(q_t, y_t),$$

where scores are positively oriented, so larger values are better. Positive score differences favour forecaster p; negative score differences favour forecaster q.

Main workflow

For most applications, start with `compare_forecasts()`. It computes pointwise scores, running mean score differences, confidence sequences, and e-processes in one call.

Scoring rules

The package includes positively oriented scoring rules such as `brier_score()`, `log_score()`, `spherical_score()`, `tick_loss()`, `qlike_score()`, `winkler_score()`, `crps_normal()`, `crps_empirical()`, and `crps_std()`.

Confidence sequences

Use `cs_hoeffding()` for Hoeffding-style confidence sequences, `cs_bernstein()` for empirical Bernstein confidence sequences, and `cs_asymptotic()` for asymptotic confidence sequences when finite-sample boundedness is not available.

E-processes

Use `eprocess()` for the main sub-exponential mixture e-process and `eprocess_rejections()` to extract first rejection times. For multi-step forecasts, see `eprocess_lag()`. For predictable time-varying bounds, see `eprocess_predictable()`.

Winkler scores

For binary probability forecasts with unbounded base scores, use `winkler_score()`, `winkler_cs()`, `winkler_etest()`, or `winkler_compare()`.

References

Choe, Y. J. and Ramdas, A. (2024). Comparing Sequential Forecasters. Operations Research, 72(4), 1368-1387.

Howard, S. R., Ramdas, A., McAuliffe, J. and Sekhon, J. (2021). Time-uniform, nonparametric, nonasymptotic confidence sequences. The Annals of Statistics, 49(2).

`brier_score` *Brier score for binary and categorical forecasts*

Description

Computes the positively oriented Brier/quadratic score. Vector probability input is treated as binary; matrix probability input is treated as categorical.

Usage

```
brier_score(p, y)
```

Arguments

`p`	Numeric vector in `[0, 1]` for binary forecasts, or a numeric matrix whose rows are probability vectors for categorical forecasts.
`y`	For binary vector input, numeric vector in `{0, 1}`. For categorical matrix input, integer vector in `{1, ..., K}`, where `K = ncol(p)`.

Details

For binary forecasts, this computes

$$S(p, y) = -(p - y)^2.$$

For categorical forecasts, this computes

$$S(\mathbf{p}, y) = -\frac{1}{2}\|\mathbf{p} - e_y\|_2^2,$$

where `e_y` is the one-hot vector of the realised category.

With the convention that category 2 corresponds to the binary event `y = 1`, the categorical formula recovers the binary formula exactly when `K = 2`.

Value

Numeric vector of scores in `[-1, 0]`. Higher is better.

Bounds

Score differences lie in `[-1, 1]`, so use `c = 1` for Theorem 1 and `c = 2` for Theorems 2 and 3.

Examples

```
p <- c(0.2, 0.7, 0.9)
y <- c(0, 1, 1)
brier_score(p, y)
```

`calibrate_p_to_e` *P-to-e calibrator*

Description

Converts anytime-valid p-values to e-values using the mixture or simple calibrator, as used in CR23 Section 4.4.

Usage

```
calibrate_p_to_e(p, strategy = "mixture", eps = 1e-16)
```

Arguments

`p`	Numeric vector of p-values in (0, 1].
`strategy`	Character. `"mixture"` (default, from Vovk & Wang 2021) or `"simple"`. See Details for the formulas.
`eps`	Numeric. Numerical guard for log(0). Default: 1e-16.

Details

Mixture calibrator (default, matches Python comparecast behaviour):

$$f(p) = \frac{1 - p + p\log(p)}{p\,(\log p)^2}$$

Simple calibrator (`strategy = "simple"`):

$$f(p) = \frac{1}{2\sqrt{p}}$$

Value

Numeric vector of e-values >= 0.

Examples

```
p <- c(0.5, 0.1, 0.01)
calibrate_p_to_e(p)
calibrate_p_to_e(p, strategy = "simple")
```

cm_boundary *Normal mixture (CM) boundary*

Description

Normal mixture (CM) boundary

Usage

```
cm_boundary(v, alpha, rho)
```

Arguments

v	Numeric vector >= 0. Intrinsic time values (V_t or t).
alpha	Numeric in (0,1). ONE-SIDED significance level.
rho	Numeric > 0. Tuning parameter. Obtain via rho_from_vopt().

Value

Numeric vector of boundary values (cumulative-sum scale, before /t).

See Also

`rho_from_vopt()` to compute `rho` from a target `v_opt`.

Examples

```
rho <- rho_from_vopt(v_opt = 10, alpha = 0.025)
u   <- cm_boundary(v = 1:500, alpha = 0.025, rho = rho)
```

compare_forecasts *Compare Two Sequential Forecasters*

Description

Computes pointwise scores for two probabilistic forecasters and compares them sequentially using confidence sequences and, when valid finite-sample bounds are available, e-processes.

Usage

```
compare_forecasts(
  p,
  q,
  y,
  scoring_rule = c("brier", "spherical", "log"),
  alpha = 0.05,
  cs_type = NULL,
  compute_cs = TRUE,
  compute_e = TRUE,
  v_opt = 10,
  boundary = "mixture",
  lcb_only = FALSE,
  ucb_only = FALSE,
  eps = 1e-15,
  clip_max = 1e+07
)
```

Arguments

p	Forecasts from forecaster 1. For binary outcomes, a numeric vector of probabilities for event y = 1. For categorical outcomes, a numeric matrix whose rows are probability vectors.
q	Forecasts from forecaster 2, in the same format as p.
y	Outcomes. For binary vector forecasts, a numeric vector in {0, 1}. For categorical matrix forecasts, integer class labels in {1, ..., K}.
scoring_rule	Character. Scoring rule used to compare forecasts. Currently supports "brier", "spherical", and "log".
alpha	Numeric in (0, 1). Significance level. Default is 0.05.
cs_type	Character or NULL. Confidence sequence type: "bernstein", "hoeffding", "asymptotic", or "none". If NULL, the wrapper uses "bernstein" for bounded scoring rules ("brier" and "spherical") and "asymptotic" for "log".
compute_cs	Logical. If TRUE, compute a confidence sequence. Default is TRUE.
compute_e	Logical. If TRUE, compute two one-sided e-processes. Default is TRUE. This is only allowed for bounded score differences under the current wrapper, namely "brier" and "spherical".
v_opt	Numeric > 0. Intrinsic time at which the mixture boundary or e-process is tuned to be tightest. Default is 10.
boundary	Character. Boundary type passed to cs_hoeffding() or cs_bernstein(). Default is "mixture".
lcb_only	Logical. If TRUE, compute a lower one-sided empirical Bernstein CS. Only used when cs_type = "bernstein".
ucb_only	Logical. If TRUE, compute an upper one-sided empirical Bernstein CS. Only used when cs_type = "bernstein".

eps	Numeric. Probability floor passed to log_score() when scoring_rule = "log". Default is 1e-15.
clip_max	Numeric. Maximum e-process value before clipping. Passed to eprocess(). Default is 1e7.

Details

This is a convenience wrapper around brier_score(), spherical_score(), log_score(), cs_hoeffding(), cs_bernstein(), cs_asymptotic(), and eprocess(). It is designed for the common workflow where the user has two forecast streams p and q, an outcome stream y, and wants a single tidy output object.

All scoring rules in seqcomp are positively oriented: higher scores are better. Therefore

$$\hat{\delta}_t = S(p_t, y_t) - S(q_t, y_t)$$

is positive when forecaster p performs better than forecaster q at time t.

For "brier" and "spherical", score differences are bounded in [-1, 1]. The wrapper therefore uses c = 1 for Hoeffding-style confidence sequences and c = 2 for empirical Bernstein confidence sequences and e-processes.

For "log", score differences are unbounded. The wrapper therefore defaults to cs_asymptotic() and refuses to compute finite-sample e-processes. For binary log-score comparisons where the Winkler construction is appropriate, use winkler_compare() instead.

Value

A data.frame with one row per time point and columns:

t Time index.

score_p Pointwise score of forecaster p.

score_q Pointwise score of forecaster q.

delta Pointwise score difference, score_p - score_q.

estimate Running mean score difference. Positive values favour forecaster p; negative values favour forecaster q.

lower, upper Confidence sequence bounds. These are NA if compute_cs = FALSE or cs_type = "none".

e_pq, e_qp One-sided e-processes. e_pq tests whether forecaster p outperforms q; e_qp tests the reverse direction. These are NA if compute_e = FALSE.

Interpretation

The confidence sequence estimates the running average score advantage of p over q. If the whole interval lies above zero, the data favour p; if the whole interval lies below zero, the data favour q.

The e-processes are evidence processes for one-sided null hypotheses. At level alpha, the two-sided rejection threshold used by eprocess() is 2 / alpha.

Examples

```
set.seed(1)
y <- rbinom(200, 1, 0.5)
p <- rep(0.5, 200)
q <- runif(200)

out <- compare_forecasts(p, q, y, scoring_rule = "brier")
tail(out)
```

`crps_empirical` *Negated CRPS for empirical predictive distributions*

Description

Wrapper over `scoringRules::crps_sample` using method = "edf" (empirical distribution function, O(n log n) via quantile decomposition of Laio & Tamea, 2007). Positively oriented: higher = better.

Usage

```
crps_empirical(ensemble, y)
```

Arguments

`ensemble`	Matrix. T x n matrix of forecast draws. Each row corresponds to one observation in y and comprises n simulation draws from the predictive distribution. For Historical Simulation: each row is the past WINDOW returns.
`y`	Numeric vector of length T. Realised observations.

Details

Requires `nrow(ensemble) == length(y)`. Passes `dat = ensemble` directly to `crps_sample` which handles vectorisation over rows natively. `show_messages` is suppressed as the "edf" method requires no bandwidth selection messages.

Value

Numeric vector of length T of CRPS values in `(-Inf, 0]` (negated loss).

Examples

```
if (requireNamespace("scoringRules", quietly = TRUE)) {
  ensemble <- matrix(c(0.1, 0.2, 0.3, 1.0, 1.1, 1.2), nrow = 2, byrow = TRUE)
  crps_empirical(ensemble, y = c(0.25, 1.05))
}
```

crps_normal — *Negated CRPS for normal predictive distributions*

Description

Computes the Continuous Ranked Probability Score for a normal predictive distribution using `scoringRules::crps_norm()` and negates it so that higher values are better.

Usage

```
crps_normal(mu, sigma, x)
```

Arguments

`mu`	Numeric vector. Location parameters (conditional means).
`sigma`	Numeric vector. Scale parameters (conditional SDs, > 0).
`x`	Numeric vector. Realised observations.

Details

Calls `scoringRules::crps_norm(y = x, mean = mu, sd = sigma)` and negates. Use for GARCH(1,1)-norm forecasts where mu is the conditional mean and sigma is the conditional standard deviation.

Value

Numeric vector of CRPS values in `(-Inf, 0]` (negated loss).

Examples

```
if (requireNamespace("scoringRules", quietly = TRUE)) {
  crps_normal(mu = c(0, 1), sigma = c(1, 2), x = c(0.2, 1.3))
}
```

crps_std — *Negated CRPS for Student-t predictive distributions*

Description

Wrapper over `scoringRules::crps_t`. Positively oriented: higher = better. The dof > 2 constraint ensures finite variance, which is required for the CRPS to be well-defined for the t-distribution.

Usage

```
crps_std(mu, sigma, dof, x)
```

Arguments

`mu`	Numeric vector. Location parameters (conditional means).
`sigma`	Numeric vector. Scale parameters (conditional SDs, > 0).
`dof`	Numeric vector or scalar. Degrees of freedom (> 2). May be scalar if constant across all observations (e.g. estimated once per rolling window).
`x`	Numeric vector. Realised observations.

Details

Calls `scoringRules::crps_t(y = x, df = dof, location = mu, scale = sigma)` and negates. Use for GARCH(1,1)-std forecasts where dof is the estimated degrees-of-freedom parameter from ugarchroll.

Value

Numeric vector of CRPS values in `(-Inf, 0]` (negated loss).

Examples

```
if (requireNamespace("scoringRules", quietly = TRUE)) {
  crps_std(mu = c(0, 1), sigma = c(1, 2), dof = 5, x = c(0.2, 1.3))
}
```

`cs_asymptotic` *Asymptotic confidence sequence (Appendix C, Eq. 55, Choe & Ramdas 2023)*

Description

Asymptotic, not finite-sample: coverage `>= 1 - alpha` holds only as `t -> infinity`. Valid without requiring bounded score differences — requires only that `hat_delta_t` has finite variance — so it's appropriate for tick loss and other scoring rules where hard bounds depend on unbounded realised values. Suitable for large evaluation windows.

Usage

```
cs_asymptotic(scores1, scores2, alpha = 0.05, t_star = NULL)
```

Arguments

`scores1`	Numeric vector. Scores for forecaster 1.
`scores2`	Numeric vector. Scores for forecaster 2.
`alpha`	Numeric in (0,1). Significance level. Default: 0.05.
`t_star`	Numeric > 0. Sample size at which CS is tightest. Default: length(scores1) (tightest at end of sample).

Details

$$C_t^A = \hat{\Delta}_t \pm \sqrt{\frac{2(t\sigma_t^2\rho^2+1)}{t^2\rho^2}\log\frac{\sqrt{t\sigma_t^2\rho^2+1}}{\alpha}}$$

where $\sigma_t^2 = \frac{1}{t}\sum_{i=1}^{t}(\hat{\delta}_i - \hat{\Delta}_{i-1})^2$ and `rho` is tuned to be tightest at `t_star`:

$$\rho(t_{star}) = \sqrt{\frac{2\log(1/\alpha)+\log(1+2\log(1/\alpha))}{t_{star}}}$$

The running variance estimator uses the predictable mean `hat_Delta_{t-1}` (not the current mean `hat_Delta_t`) to maintain predictability, with `hat_Delta_0 := 0`.

Value

data.frame with columns t, estimate, lower, upper.

Examples

```
scores1 <- c(-0.4, -0.2, -0.3, -0.1, -0.2)
scores2 <- c(-0.5, -0.3, -0.4, -0.2, -0.3)
cs_asymptotic(scores1, scores2, alpha = 0.05)
```

---

cs_bernstein — *Empirical Bernstein confidence sequence (Theorem 2, Choe & Ramdas 2023)*

Description

Constructs a variance-adaptive time-uniform CS using empirical intrinsic time $\hat{V}_t = \sum_{i=1}^{t}(\hat{\delta}_i - \gamma_i)^2$. Tighter than the Hoeffding CS when score differences have low variance.

Usage

```
cs_bernstein(
  scores1,
  scores2,
  alpha = 0.05,
  c = 2,
  v_opt = 10,
  boundary = "mixture",
  gammas = NULL,
  lcb_only = FALSE,
  ucb_only = FALSE
)
```

Arguments

`scores1`	Numeric vector. Scores for forecaster 1.
`scores2`	Numeric vector. Scores for forecaster 2.
`alpha`	Numeric in (0,1). Significance level. Default: 0.05.
`c`	Numeric > 0. Sub-exponential scale. The process must satisfy |hat_delta_i| <= c/2. For score differences in `[a-b, b-a]`, c = b - a (e.g. c = 2 for Brier score differences in `[-1,1]`). Default: 2.
`v_opt`	Numeric > 0. Optimal intrinsic time. Default: 10.
`boundary`	Character. "mixture" (default, GE mixture) or "stitching" (polynomial stitched) or "hardcoded" (CR23 example formula, only valid for alpha=0.05, c=1).
`gammas`	Numeric vector or NULL. Predictable centering sequence. If NULL, constructed as lagged running mean (default).
`lcb_only`	Logical. If TRUE, return lower CS only: `[lower, +Inf)`. Requires finite lower bound on hat_delta_i; provide c.
`ucb_only`	Logical. If TRUE, return upper CS only: `(-Inf, upper]`.

Details

The CS is:

$$C_t^{EB} = \hat{\Delta}_t \pm u_{\alpha/2}^{GE}(\hat{V}_t;\, \rho, c) \;/\; t$$

Value

data.frame with columns t, estimate, lower, upper. lower = -Inf if ucb_only = TRUE; upper = Inf if lcb_only = TRUE.

Examples

```
scores1 <- c(-0.04, -0.09, -0.01, -0.16)
scores2 <- c(-0.09, -0.16, -0.04, -0.25)
cs_bernstein(scores1, scores2, alpha = 0.05)
```

`cs_hoeffding` *Hoeffding-style confidence sequence (Theorem 1, Choe & Ramdas 2023)*

Description

Constructs a time-uniform confidence sequence for the mean score difference $\Delta_t = \frac{1}{t}\sum_{i=1}^{t} E[\hat{\delta}_i \mid \mathcal{F}_{i-1}]$.

Usage

```
cs_hoeffding(
  scores1,
  scores2,
  alpha = 0.05,
  c = 1,
  v_opt = 10,
  boundary = "mixture"
)
```

Arguments

`scores1`	Numeric vector. Scores S(p_t, y_t) for forecaster 1.
`scores2`	Numeric vector. Scores S(q_t, y_t) for forecaster 2.
`alpha`	Numeric in (0,1). Significance level. The CS has coverage 1 - alpha uniformly over all t. Default: 0.05.
`c`	Numeric > 0. Sub-Gaussian scale. The process must satisfy |hat_delta_i| <= c for all i. For scores in `[a,b]`, the difference is in `[a-b, b-a]`, so c = b - a. Default: 1 (appropriate for Brier score differences in `[-1,1]`).
`v_opt`	Numeric > 0. Intrinsic time at which the CS is tightest. Default: 10 (recommended by CR23).
`boundary`	Character. "mixture" (default, recommended) or "stitching".

Value

data.frame with columns t, estimate, lower, upper.

Assumption

Requires `hat_delta_i` to be c-sub-Gaussian given $\mathcal{F}_{i-1}$, i.e. `|hat_delta_i| <= c` for all `i`.

Boundary

$$C_t^H = \hat{\Delta}_t \pm u_{\alpha/2}^{CM}(c^2 t; \rho)/t$$

where u^{CM} is the normal mixture boundary and $c^2 t$ is the intrinsic time for a c-sub-Gaussian process with deterministic variance proxy. The intrinsic time for Theorem 1 is `v_t = c^2 * t`, not `v_t = t`: the CM boundary implicitly assumes 1-sub-Gaussian inputs, so the `c^2` scaling must be applied explicitly. This matches the H21 convention, where the boundary absorbs the sub-Gaussian parameter via the variance process definition.

Relation to Python comparecast: Python uses `v_t = sigma * t` where `sigma = (hi - lo)/2 = c`. This is equivalent to our `c^2 * t` only when `c = 1`. For `c != 1` the parametrisations differ; we follow the paper.

Output

Returns a `data.frame` with one row per `t` and columns `t`, `estimate` (the running mean `hat_Delta_t`), `lower`, and `upper`, with coverage guarantee $P(\forall t \geq 1 : \Delta_t \in [\text{lower}_t, \text{upper}_t]) \geq 1 - \alpha$.

Examples

```
scores1 <- c(-0.04, -0.09, -0.01, -0.16)
scores2 <- c(-0.09, -0.16, -0.04, -0.25)
cs_hoeffding(scores1, scores2, alpha = 0.05)
```

---

eprocess *Sub-exponential mixture e-process (Theorem 3, Choe & Ramdas 2023)*

Description

Constructs two simultaneous one-sided e-processes for sequentially testing whether forecaster 1 (p) outperforms forecaster 2 (q) or vice versa.

Usage

```
eprocess(
  scores1,
  scores2,
  alpha = 0.05,
  c = 2,
  v_opt = 10,
  alpha_opt = NULL,
  gammas = NULL,
  clip_max = 1e+07
)
```

Arguments

`scores1`	Numeric vector. Scores S(p_t, y_t) for forecaster 1.
`scores2`	Numeric vector. Scores S(q_t, y_t) for forecaster 2.
`alpha`	Numeric in (0,1). Significance level. Rejection threshold is 2/alpha for the two-sided test. Default: 0.05.
`c`	Numeric > 0. Sub-exponential scale. Must satisfy |hat_delta_i| <= c/2 for all i. For score differences in `[-(b-a), b-a]`: c = 2*(b-a). Default: 2 (for Brier score differences in `[-1,1]`).
`v_opt`	Numeric > 0. Intrinsic time at which e-process grows fastest. Default: 10 (recommended by CR23).
`alpha_opt`	Numeric in (0,1). One-sided alpha used to compute rho. Default: alpha/2 (matches comparecast two-sided convention).
`gammas`	Numeric vector or NULL. Predictable centering sequence. If NULL, constructed as lagged running mean.
`clip_max`	Numeric. Maximum e-process value before clipping. Default: 1e7 (matches Python comparecast).

Details

The mixture e-process at time t is:

$$E_t^{\text{mix}} = m(S_t, \hat{V}_t)$$

where $S_t = \sum_{i=1}^{t} \hat{\delta}_i$, $\hat{V}_t = \sum_{i=1}^{t} (\hat{\delta}_i - \gamma_i)^2$, and $m(s, v)$ is the Gamma-Exponential mixture function (Proposition 3, CR23).

VARIANCE PROCESS: The intrinsic time V_hat_t uses NO floor (unlike the EB CS). The GE mixture m(s, v) is well-defined at v=0 (returns 1 when s=0), so no floor is needed. Adding a floor would distort e-values.

SCALE CONVENTION: c is the sub-exponential scale parameter such that |hat_delta_i| <= c/2. This is the Theorems 2 & 3 convention from CR23. For Brier score differences in `[-1,1]`: c = 2. For Winkler scores (bounded above by 1): c = 2.

LOG-SPACE: E-process values are computed in log-space and clipped before exponentiating to avoid numerical overflow.

Value

data.frame with columns t, e_pq, e_qp, log_e_pq, log_e_qp.

Rejection rule

At level `alpha`: reject $H_0^w(p, q)$ (conclude p outperforms q) when `e_pq >= 2/alpha`; reject $H_0^w(q, p)$ (conclude q outperforms p) when `e_qp >= 2/alpha`. Use `eprocess_rejections()` to extract the first crossing time for each.

Examples

```
scores1 <- c(-0.04, -0.09, -0.01, -0.16)
scores2 <- c(-0.09, -0.16, -0.04, -0.25)
ep <- eprocess(scores1, scores2, alpha = 0.05)
head(ep)
```

`eprocess_lag` *Lag-h e-process for sequential forecast comparison (Propositions 5 & 6)*

Description

For h-step-ahead forecasts, constructs an anytime-valid e-process by stream splitting and combining h individual e-processes.

Usage

```
eprocess_lag(
  scores1,
  scores2,
  h = 1,
  alpha = 0.05,
  c = 2,
  v_opt = 10,
  null = "pw",
  calibrate = TRUE,
  cal_strategy = "mixture"
)
```

Arguments

`scores1`	Numeric vector. Scores for forecaster 1.
`scores2`	Numeric vector. Scores for forecaster 2.
`h`	Integer >= 1. Forecast lag. For h=1, reduces to standard eprocess() — no splitting.
`alpha`	Numeric in (0,1). Significance level. Default: 0.05.
`c`	Numeric > 0. Sub-exponential scale. Default: 2.
`v_opt`	Numeric > 0. Default: 10.
`null`	Character. Null hypothesis type: • `"pw"` — period-wise weak null (average combination). • `"w"` — standard weak null (minimum combination).
`calibrate`	Logical. Apply p-to-e calibration. Default: TRUE.
`cal_strategy`	Character. "mixture" (default) or "simple".

Details

For h = 1: calls eprocess() directly and returns its output unchanged.

For h >= 2: 1. Split xs into h streams 2. Compute e-process on each stream independently 3. Combine using the appropriate null rule 4. Convert to p-process, combine, calibrate back to e-process 5. Unroll to original time scale

The period-wise ("pw") null is less conservative than the standard ("w") null but tests a different (weaker) hypothesis. See CR23 Section 4.4.

Value

data.frame with columns t, e_pq, e_qp, log_e_pq, log_e_qp.

Examples

```
scores1 <- c(-0.04, -0.09, -0.01, -0.16, -0.04, -0.09)
scores2 <- c(-0.09, -0.16, -0.04, -0.25, -0.09, -0.16)
ep <- eprocess_lag(scores1, scores2, h = 2, alpha = 0.05)
head(ep)
```

---

eprocess_predictable *Fixed-lambda e-process with predictable bounds (Proposition 7)*

Description

Constructs a valid e-process when score difference bounds vary over time but are predictable (known at time i-1 before observing hat_delta_i).

Usage

```
eprocess_predictable(
  scores1,
  scores2,
  c_seq,
  lambda = NULL,
  alpha = 0.05,
  gammas = NULL,
  clip_max = 1e+07,
  strict = FALSE
)
```

Arguments

`scores1`	Numeric vector. Scores for forecaster 1.
`scores2`	Numeric vector. Scores for forecaster 2.
`c_seq`	Numeric vector. Predictable bound sequence (c_i), same length as scores1. Must satisfy `|scores1[i]-scores2[i]| <= c_i/2` and c_i > 0 for all i.
`lambda`	Numeric in `[0, 1/c_0)`. Betting parameter. Must be strictly less than 1/c_0 where c_0 = max(c_seq). If NULL, uses the recommended default lambda = 0.5/c_0.
`alpha`	Numeric in (0,1). Significance level for rejection rule. Default: 0.05. Not used in computation, only for API consistency. Pass the same value to `predictable_rejections()` when evaluating rejection.
`gammas`	Numeric vector or NULL. Predictable centering sequence. If NULL, constructed as lagged running mean.
`clip_max`	Numeric. Maximum e-process value. Default: 1e7.
`strict`	Logical. If TRUE, any violation of the bound condition at any time point will stop execution with an error. If FALSE (default), a warning is issued but the e-process is still computed. Note that violations invalidate the e-process guarantee, so strict = TRUE is recommended for formal inference.

Details

The e-process is computed as:

$$\log E_t(\lambda) = \sum_{i=1}^{t} \Big[\lambda\, \hat{\delta}_i - \psi_{E,c_i}(\lambda)\, (\hat{\delta}_i - \gamma_i)^2 \Big]$$

where

$$\psi_{E,c}(\lambda) = \frac{-\log(1 - c\lambda) - c\lambda}{c^2}$$

is evaluated at each step with the current c_i.

LAMBDA CHOICE: lambda = 0.5/c_0 is a conservative default that stays well within the valid domain `[0, 1/c_0)`. For better power, lambda can be tuned to the expected signal size, but must never reach 1/c_0.

VALIDITY CHECK: The function verifies |hat_delta_i| <= c_i/2 at each step and warns if violated. Violations invalidate the e-process guarantee.

Value

data.frame with columns: t, e_pq, e_qp, log_e_pq, log_e_qp, c_seq, lambda_used

Predictability

The bound sequence `c_seq` (and the centering sequence `gammas`) must be predictable: `c_i` is fixed and known at time `i - 1`, before `scores1[i]`/`scores2[i]` (and hence `hat_delta_i`) are observed — formally, c_i is $\mathcal{F}_{i-1}$-measurable. A bound chosen after seeing `hat_delta_i` (e.g. derived from the realised data range) invalidates the e-process guarantee, even if it numerically satisfies `|hat_delta_i| <= c_i/2`.

Examples

```
scores1 <- c(0.10, 0.20, 0.15, 0.25)
scores2 <- c(0.05, 0.10, 0.10, 0.20)
c_seq <- rep(1, length(scores1))
ep <- eprocess_predictable(scores1, scores2, c_seq = c_seq)
head(ep)
```

`eprocess_rejections` *Determine rejection times for an e-process output*

Description

Determine rejection times for an e-process output

Usage

```
eprocess_rejections(ep, alpha = 0.05)
```

Arguments

ep	data.frame. Output of eprocess().
alpha	Numeric. Significance level. Threshold is 2/alpha.

Value

Named list with elements:

- threshold — rejection threshold (2 / alpha).
- tau_pq — first t where e_pq >= threshold (NA if never crossed).
- tau_qp — first t where e_qp >= threshold (NA if never crossed).
- reject_pq — logical: was $H_0^w(p, q)$ ever rejected?
- reject_qp — logical: was $H_0^w(q, p)$ ever rejected?

Examples

```
scores1 <- c(-0.04, -0.09, -0.01, -0.16)
scores2 <- c(-0.09, -0.16, -0.04, -0.25)
ep <- eprocess(scores1, scores2, alpha = 0.05)
eprocess_rejections(ep, alpha = 0.05)
```

---

ge_boundary *Gamma-exponential mixture boundary*

Description

Gamma-exponential mixture boundary

Usage

```
ge_boundary(v, alpha, rho, c, s_lo = -10, s_hi = 500)
```

Arguments

v	Numeric vector >= 0. Intrinsic time values.
alpha	Numeric in (0,1). ONE-SIDED significance level.
rho	Numeric > 0. From rho_from_vopt().
c	Numeric > 0. Sub-exponential scale.
s_lo	Numeric. Lower search bound for uniroot. Default: -10.
s_hi	Numeric. Upper search bound for uniroot. Default: 500.

Details

Computes $u_{GE}(v; \alpha, \rho, c) = \sup\{s : m(s, v) < 1/\alpha\}$ by solving `m(s, v) = 1/alpha` numerically for `s` via `uniroot()`, separately for each `v_i`.

Root-finding fallback: the search starts in `[s_lo, s_hi]`; if `m(s_hi, v_i)` has not yet crossed the target, `s_hi` is doubled once and retried. If it still fails, a warning is issued and `s_hi` is returned as a conservative fallback value. Increase `s_hi` directly if this warning appears often (e.g. at large `v` or small `alpha`); increase `abs(s_lo)` if no root is found at small `v`.

Computed elementwise; can be slow for long vectors — consider caching boundary values when the same `(alpha, rho, c)` are reused.

Value

Numeric vector of boundary values (cumulative-sum scale).

Examples

```
rho <- rho_from_vopt(v_opt = 10, alpha = 0.025)
ge_boundary(v = 1:3, alpha = 0.025, rho = rho, c = 2)
```

---

log_score — *Logarithmic score for binary and categorical forecasts*

Description

Computes the positively oriented logarithmic score. Vector probability input is treated as binary; matrix probability input is treated as categorical.

Usage

```
log_score(p, y, eps = 1e-15)
```

Arguments

`p`	Numeric vector in `[0, 1]` for binary forecasts, or a numeric matrix whose rows are probability vectors for categorical forecasts.
`y`	For binary vector input, numeric vector in `{0, 1}`. For categorical matrix input, integer vector in `{1, ..., K}`, where `K = ncol(p)`.
`eps`	Numeric. Probability floor used before taking logarithms. Default is `1e-15`. Set to `0` to disable clipping.

Details

For binary forecasts, this computes

$$S(p, y) = y \log(p) + (1 - y) \log(1 - p).$$

For categorical forecasts, this computes

$$S(\mathbf{p}, y) = \log(p_y),$$

where `p_y` is the forecast probability assigned to the realised category.

Value

Numeric vector of scores in `(-Inf, 0]`. Higher is better.

Use with seqcomp

The logarithmic score is unbounded below. It should not be used directly with the finite-sample bounded-difference confidence sequences or e-processes. For binary outcomes, use `winkler_score()` and `winkler_cs()` when the Winkler construction is appropriate. For unbounded score differences, use `cs_asymptotic()` or supply genuine predictable bounds to `eprocess_predictable()`.

Examples

```
p <- c(0.2, 0.7, 0.9)
y <- c(0, 1, 1)
log_score(p, y)
```

---

predictable_rejections

Summarise predictable bounds e-process

Description

Summarise predictable bounds e-process

Usage

```
predictable_rejections(ep, alpha = 0.05)
```

Arguments

`ep`	data.frame. Output of eprocess_predictable().
`alpha`	Numeric. Significance level.

Value

Named list matching the `eprocess_rejections()` format (`threshold`, `tau_pq`, `tau_qp`, `reject_pq`, `reject_qp`), plus:

- `c_range` — range of `c_seq` used.
- `lambda` — `lambda` value used.

Examples

```
scores1 <- c(0.10, 0.20, 0.15, 0.25)
scores2 <- c(0.05, 0.10, 0.10, 0.20)
c_seq <- rep(1, length(scores1))
ep <- eprocess_predictable(scores1, scores2, c_seq = c_seq)
predictable_rejections(ep, alpha = 0.05)
```

`ps_boundary` *Polynomial stitched (PS) boundary*

Description

Alternative boundary for both Theorem 1 and Theorem 2 constructions, included for completeness and for cross-checking CR23 Table results.

Usage

```
ps_boundary(v, alpha, v_opt = 10, c = 1, s = 1.4, eta = 2)
```

Arguments

`v`	Numeric vector >= 0. Intrinsic time values.
`alpha`	Numeric in (0,1). ONE-SIDED significance level.
`v_opt`	Numeric > 0. Optimal intrinsic time (= m in H21). Default: 10.
`c`	Numeric > 0. Sub-exponential scale.
`s`	Numeric > 1. Stitching parameter. Default: 1.4.
`eta`	Numeric > 1. Geometric spacing. Default: 2.

Details

This is **not** the recommended primary boundary: the CM/GE mixture boundaries (`cm_boundary()`, `ge_boundary()`) are tighter in CR23 and are used by default throughout `seqcomp`. Use `ps_boundary()` only when you specifically need the polynomial-stitched construction.

Value

Numeric vector of boundary values (cumulative-sum scale).

Examples

```
ps_boundary(v = 1:5, alpha = 0.025, v_opt = 10, c = 1)
```

`qlike_score` *Negated QLIKE score for variance forecasts*

Description

Computes the positively oriented (negated) QLIKE quasi-likelihood loss for variance forecasts.

Usage

```
qlike_score(sigma2_hat, sigma2)
```

Arguments

`sigma2_hat`	Numeric vector. Forecast variance (strictly positive).
`sigma2`	Numeric vector. Realised variance (strictly positive).

Details

Standard QLIKE loss is

$$L_{QL}(\hat{\sigma}^2, \sigma^2) = \frac{\sigma^2}{\hat{\sigma}^2} - \log \frac{\sigma^2}{\hat{\sigma}^2} - 1.$$

This is loss-oriented (lower = better, minimum 0 at a perfect forecast), so the function negates it: $S_{QL} = -L_{QL}$.

Literature note: some sources define QLIKE as `log(sigma2_hat) + sigma2 / sigma2_hat`, which differs by constants from the form above. Here the loss is normalised to have minimum 0 and is then negated for positive orientation.

Value

Numeric vector of negated QLIKE scores. Higher is better. Maximum value is 0, achieved at a perfect forecast `sigma2_hat = sigma2`. Unbounded below.

Unbounded below

QLIKE is unbounded below. It should not be used directly with the finite-sample bounded-difference confidence sequences or e-processes. Use `cs_asymptotic()` for QLIKE-based confidence sequences, or use `eprocess_predictable()` only when genuine ex ante predictable bounds are available. QLIKE is not compatible with the Winkler construction because Winkler scores are restricted to binary outcomes and probability forecasts.

Examples

```
sigma2_hat <- c(1.0, 1.5, 2.0)
sigma2 <- c(1.1, 1.4, 2.2)
qlike_score(sigma2_hat, sigma2)
```

---

rho_from_vopt *Convert optimal intrinsic time to rho*

Description

Maps the user-specified intrinsic time `v_opt` (the point at which a boundary is tightest) to the `rho` tuning parameter, via the Lambert W formula of Howard et al. (2021), Proposition 3 (paper-exact).

Usage

```
rho_from_vopt(v_opt = 10, alpha = 0.025)
```

Arguments

`v_opt`	Numeric > 0. Intrinsic time at which the boundary is tightest. Recommended default from CR23: 10.
`alpha`	Numeric in (0,1). Significance level (one-sided). For a two-sided boundary at level alpha, pass alpha/2 here.

Details

$$\rho = \frac{v_{opt}}{-W_{-1}(-\alpha^2/e) - 1}$$

The lower branch W_{-1} is defined for `x` in `[-1/e, 0)` and returns values `<= -1`. For `alpha` in `(0, 1)`, `-alpha^2/e` is always in `(-1/e, 0)`, so the branch is well-defined.

Value

Numeric > 0. The rho tuning parameter.

Examples

```
rho_from_vopt(v_opt = 10, alpha = 0.025)
```

`score_bounds` *Score difference bounds for a named scoring rule*

Description

Returns lo, hi and the derived scale parameters c_thm1, c_thm23 for the score difference process hat_delta_t = S(p, y) - S(q, y), in those cases where a genuine, theorem-valid bound is available.

Usage

```
score_bounds(scoring_rule)
```

Arguments

`scoring_rule` Character. One of:

- `"brier"`, `"spherical"` — bounded, exact finite-sample `c`.
- `"winkler"` — descriptive helper for the one-sided CS on the log score.
- `"tick"` — unbounded; returns `NULL` with guidance.
- `"crps"`, `"crps_normal"`, `"crps_empirical"`, `"crps_std"` — unbounded; returns `NULL` with guidance.
- `"log"`, `"qlike"` — unbounded; returns `NULL` with guidance.

Details

Convention (utils.R::score_diff_scales): c_thm1 = (hi - lo) / 2 # Theorem 1: |delta_i| <= c c_thm23 = hi - lo # Theorems 2 & 3: |delta_i| <= c/2

Value

Named list with elements lo, hi, c_thm1, c_thm23 for bounded rules, or NULL for unbounded rules (with an informative message).

Per-rule notes

- **Brier / Spherical** — individual scores lie in `[-1, 0]` (Brier) or `[0, 1]` (Spherical), so score differences lie in `[-1, 1]` either way. This bound is exact and yields finite-sample anytime-valid CS via Hoeffding/Bernstein.
- **Winkler** — bounded above by 1; the lower bound is problem-dependent, so `lo = -Inf` and only `hi = 1` is used, as a descriptive helper for the one-sided CS wrapper `winkler_cs()`. Not intended for generic Hoeffding/Bernstein use (Theorem 1 requires a finite symmetric interval).
- **Tick loss** — unbounded on general financial returns. Any bound derived from an empirical data range is ex-post and not filtration-respecting, so it cannot justify finite-sample anytime validity. Use `cs_asymptotic()` for tick comparisons.
- **CRPS** (normal, t, empirical) — unbounded, since both the predictive distributions and the realised outcomes are unbounded. A historical data range is again an ex-post surrogate and does not provide a theorem-valid `c` for Hoeffding/Bernstein. Use `cs_asymptotic()`, or supply genuine ex ante bounds in problem-specific code if available.

- **Log / QLIKE** — both unbounded. For binary log-score comparisons, use `winkler_score() + winkler_cs()` when the Winkler construction is appropriate. For categorical log-score, QLIKE, and other unbounded score differences, use `cs_asymptotic()`, or `eprocess_predictable()` only with genuine ex ante predictable bounds.

Examples

```
score_bounds("brier")
score_bounds("winkler")
```

`score_diff_scales` *Score difference bounds -> sub-Gaussian / sub-exponential scale*

Description

Given score-difference bounds `[lo, hi]`, computes the two scale constants used elsewhere in `seqcomp`:

- `c_thm1 = (hi - lo) / 2` — sub-Gaussian scale for Theorem 1, where `|delta_i| <= c_thm1`.
- `c_thm23 = hi - lo` — sub-exponential scale for Theorems 2 & 3, where `|delta_i| <= c_thm23 / 2`.

Usage

```
score_diff_scales(lo, hi)
```

Arguments

`lo`	Numeric. Lower bound of score difference (usually a - b).
`hi`	Numeric. Upper bound of score difference (usually b - a).

Details

Both conventions bound the same quantity: after centering, `delta_i` lies in `[-(hi-lo)/2, (hi-lo)/2]`, so `max|delta_i| = (hi-lo)/2`, which equals both `c_thm1` and `c_thm23 / 2`.

Value

Named list with elements c_thm1 and c_thm23.

Examples

```
score_diff_scales(lo = -1, hi = 1)
```

`spherical_score` *Spherical score for binary and categorical forecasts*

Description

Computes the positively oriented spherical score. Vector probability input is treated as binary; matrix probability input is treated as categorical.

Usage

```
spherical_score(p, y)
```

Arguments

`p`	Numeric vector in `[0, 1]` for binary forecasts, or a numeric matrix whose rows are probability vectors for categorical forecasts.
`y`	For binary vector input, numeric vector in `{0, 1}`. For categorical matrix input, integer vector in `{1, ..., K}`, where `K = ncol(p)`.

Details

For binary forecasts, this computes

$$S(p, y) = \frac{py + (1-p)(1-y)}{\sqrt{p^2 + (1-p)^2}}.$$

For categorical forecasts, this computes

$$S(\mathbf{p}, y) = \frac{p_y}{\|\mathbf{p}\|_2},$$

where `p_y` is the forecast probability assigned to the realised category.

Score differences lie in `[-1, 1]`, so use `c = 1` for Theorem 1 and `c = 2` for Theorems 2 and 3.

Value

Numeric vector of scores in `[0, 1]`. Higher is better.

Examples

```
p <- c(0.2, 0.7, 0.9)
y <- c(0, 1, 1)
spherical_score(p, y)
```

split_streams — *Split a sequence into h interleaved lag streams*

Description

For lag h, the k-th stream (k = 1, ..., h) contains indices $\{k,\ k+h,\ k+2h,\ \ldots\}$, following the CR23 convention.

Usage

```
split_streams(xs, h)
```

Arguments

xs	Numeric vector. Score differences hat_delta_t.
h	Integer >= 1. Lag (number of steps ahead).

Value

List of length h. Each element is a numeric vector containing the score differences for that stream.

Examples

```
split_streams(1:10, h = 3)
# stream 1: indices 1, 4, 7, 10
# stream 2: indices 2, 5, 8
# stream 3: indices 3, 6, 9
```

tick_loss — *Negated tick loss for quantile forecasts*

Description

Computes the positively oriented (negated) tick/quantile loss (Koenker & Bassett, 1978).

Usage

```
tick_loss(q, y, alpha)
```

Arguments

q	Numeric vector. Quantile forecasts at level alpha.
y	Numeric vector. Realised outcomes.
alpha	Numeric in (0,1). Quantile level.

Details

The standard tick loss is

$$\rho_\alpha(u) = u\,(\alpha - \mathbb{1}(u < 0))\,,$$

where $u = y - q_\alpha$ is the forecast error. This is loss-oriented (lower = better), so the function negates it:

$$S_T(q, y; \alpha) = -(y - q)\,(\alpha - \mathbb{1}(y < q))\,.$$

Tick loss is unbounded on general real-valued outcomes. Bounds derived from an empirical data range are ex-post and do not provide theorem-valid constants for finite-sample Hoeffding/Bernstein confidence sequences or e-processes.

Sign convention: the negation means `hat_delta_t > 0` when forecaster `p` has smaller tick loss, hence a better quantile forecast, than forecaster `q`.

Value

Numeric vector of negated tick loss scores. Higher = better.

Examples

```
q <- c(1.0, 1.5, 2.0)
y <- c(1.2, 1.4, 2.3)
tick_loss(q, y, alpha = 0.5)
```

`unroll_stream` *Unroll a stream-wise quantity back to the original time scale*

Description

After computing a per-stream cumulative quantity (e.g. e-process values), restores them to the original length `T` by zero-padding the first `k - 1` positions, repeating each stream value `h` times, then truncating to length `T`.

Usage

```
unroll_stream(stream_vals, k, h, T_)
```

Arguments

`stream_vals`	Numeric vector. Values for stream k (length ~ T/h).
`k`	Integer. Stream index (1-based).
`h`	Integer. Lag.
`T_`	Integer. Total original sequence length.

Value

Numeric vector of length T_.

Alignment only, not theoretical updating

For lagged forecasts (h >= 2), the returned series is aligned to the evaluated score-difference index after stream splitting. It should **not** be interpreted as a process that updates at the original forecast-issuance time. The unrolled process is for visualization and alignment only; the theoretical validity argument relies strictly on the streamwise sub-filtrations, not on this unrolled representation.

Examples

```
unroll_stream(c(1, 2, 3), k = 2, h = 2, T_ = 6)
```

---

winkler_compare *Full Winkler comparison pipeline (Proposition 4)*

Description

Convenience wrapper that computes Winkler scores, one-sided CS, and e-process in a single call.

Usage

```
winkler_compare(
  p,
  q,
  y,
  alpha = 0.05,
  base_score = log_score,
  v_opt = 10,
  lower_bound = NULL
)
```

Arguments

p	Numeric vector in (0,1).
q	Numeric vector in (0,1).
y	Numeric vector containing only 0 and 1. Binary outcomes.
alpha	Numeric in (0,1). Default: 0.05.
base_score	Function. Default: log_score.
v_opt	Numeric > 0. Default: 10.
lower_bound	Numeric or NULL. See winkler_cs().

Value

Named list with elements:

- `winkler_scores` — raw Winkler score vector.
- `cs` — `data.frame` from `winkler_cs()`.
- `etest_p_worse` — one-sided e-process testing whether `p` is worse than `q`.
- `etest_q_worse` — one-sided e-process testing whether `q` is worse than `p`.
- `rejections` — list of one-sided rejection summaries.

Examples

```
p <- c(0.7, 0.6, 0.8, 0.65)
q <- c(0.5, 0.7, 0.6, 0.55)
y <- c(1, 1, 0, 1)
winkler_compare(p, q, y, alpha = 0.05)
```

---

winkler_cs — *One-sided empirical Bernstein CS for Winkler scores (Proposition 4)*

Description

Applies the Winkler normalisation and constructs a one-sided upper confidence sequence for the mean Winkler score W_t = (1/t)*sum w_i. The CS takes the form `(-Inf, U_t]`, valid uniformly over all t >= 1.

Usage

```
winkler_cs(
  p,
  q,
  y,
  alpha = 0.05,
  base_score = log_score,
  v_opt = 10,
  lower_bound = NULL
)
```

Arguments

`p`	Numeric vector in (0,1). Forecasts from model 1.
`q`	Numeric vector in (0,1). Forecasts from model 2.
`y`	Numeric vector containing only 0 and 1. Binary outcomes.
`alpha`	Numeric in (0,1). Significance level. Default: 0.05.
`base_score`	Function. Underlying scoring rule. Default: log_score.

`v_opt`	Numeric > 0. Optimal intrinsic time. Default: 10.
`lower_bound`	Numeric or NULL. Analytical lower bound on w_i for two-sided CS via Corollary 2. If NULL (default), returns one-sided CS only. If supplied, must satisfy w_i >= lower_bound for all i almost surely.

Details

Scale convention: Winkler score bounded above by 1, so c/2 = 1, c = 2. This is hardcoded — do not change c without re-deriving the bound.

Value

data.frame with columns t, estimate, lower, upper. lower = -Inf always (one-sided) unless lower_bound is supplied.

Interpretation

If `U_t < 0` for some `t`, this is time-uniform evidence that forecaster 1 (`p`) is worse than forecaster 2 (`q`) on average — i.e. a rejection is evidence against `p`, not for it. More generally, `W_t > 0` suggests `p` outperforms `q`; `W_t < 0` suggests `q` outperforms `p`.

Examples

```
p <- c(0.7, 0.6, 0.8, 0.65)
q <- c(0.5, 0.7, 0.6, 0.55)
y <- c(1, 1, 0, 1)
winkler_cs(p, q, y, alpha = 0.05)
```

`winkler_etest` *E-process for Winkler scores (Proposition 4 + Theorem 3)*

Description

Tests whether the mean Winkler score W_t >= 0 for all t. A rejection provides time-uniform evidence that forecaster 1 (p) is worse than forecaster 2 (q) under the base scoring rule.

Usage

```
winkler_etest(
  p,
  q,
  y,
  alpha = 0.05,
  base_score = log_score,
  v_opt = 10,
  clip_max = 1e+07
)
```

Arguments

`p`	Numeric vector in (0,1). Forecasts from model 1.
`q`	Numeric vector in (0,1). Forecasts from model 2.
`y`	Numeric vector containing only 0 and 1. Binary outcomes.
`alpha`	Numeric in (0,1). Significance level. Default: 0.05.
`base_score`	Function. Underlying scoring rule. Default: log_score.
`v_opt`	Numeric > 0. Default: 10.
`clip_max`	Numeric. Maximum e-process value before clipping. Default: 1e7.

Value

data.frame with columns t, e, log_e.

Rejection rule

Reject at level `alpha` when `e >= 1 / alpha`; this provides time-uniform evidence that `p` is worse than `q`.

Examples

```
p <- c(0.7, 0.6, 0.8, 0.65)
q <- c(0.5, 0.7, 0.6, 0.55)
y <- c(1, 1, 0, 1)
winkler_etest(p, q, y, alpha = 0.05)
```

`winkler_score` *Winkler-normalized binary score*

Description

Normalises the score difference S(p,y) - S(q,y) by the maximum possible score difference given the forecaster ordering, mapping the result to `(-Inf, 1]` (Proposition 4, Choe & Ramdas 2023). Used to apply Theorems 2 & 3 to unbounded scoring rules on binary outcomes.

Usage

```
winkler_score(p, q, y, base_score = log_score, eps = 1e-08)
```

Arguments

`p`	Numeric vector in (0,1). Forecasts from model 1.
`q`	Numeric vector in (0,1). Forecasts from model 2.
`y`	Numeric vector containing only 0 and 1. Binary outcomes.
`base_score`	Function. The underlying scoring rule S(p, y). Must accept two arguments: forecast probability and outcome. Default: log_score (with eps clipping).
`eps`	Numeric. Zero-protection for the normaliser denominator. Default: 1e-8 (matches Python comparecast convention).

Details

$$w(p, q, y) = \frac{S(p, y) - S(q, y)}{S(p, \mathbb{1}(p > q)) - S(q, \mathbb{1}(p > q))}$$

with the convention 0/0 := 0.

The lower bound is problem-dependent (depends on how extreme p and q can be). For a two-sided CS via Corollary 2, the user must establish a finite lower bound analytically. If no finite lower bound can be guaranteed, use the one-sided (upper) CS only, as in the CR23 MLB experiments.

Value

Numeric vector. Winkler scores in `(-Inf, 1]`. Upper bound of 1 is tight: w = 1 when y = 1(p > q).

When to use

Strictly limited to binary outcomes `y` in `{0, 1}` and probability forecasts `p`, `q` in `(0, 1)`. Not applicable to QLIKE or other continuous-outcome scoring rules. See CR23 Section G for discussion.

For use in Theorems 2 & 3: upper bound = 1 implies c/2 = 1, so use `c = 2` in all GE boundary and e-process calls.

Examples

```
p <- c(0.7, 0.6, 0.8, 0.65)
q <- c(0.5, 0.7, 0.6, 0.55)
y <- c(1, 1, 0, 1)
winkler_score(p, q, y)
```

Index

brier_score, 4
brier_score(), *3*, *8*

calibrate_p_to_e, 5
cm_boundary, 6
compare_forecasts, 6
compare_forecasts(), *3*
crps_empirical, 9
crps_empirical(), *3*
crps_normal, 10
crps_normal(), *3*
crps_std, 10
crps_std(), *3*
cs_asymptotic, 11
cs_asymptotic(), *3*, *8*
cs_bernstein, 12
cs_bernstein(), *3*, *7*, *8*
cs_hoeffding, 13
cs_hoeffding(), *3*, *7*, *8*

eprocess, 15
eprocess(), *3*, *8*
eprocess_lag, 16
eprocess_lag(), *3*
eprocess_predictable, 18
eprocess_predictable(), *3*
eprocess_rejections, 19
eprocess_rejections(), *3*

ge_boundary, 20

log_score, 21
log_score(), *3*, *8*

predictable_rejections, 22
predictable_rejections(), *18*
ps_boundary, 23

qlike_score, 24
qlike_score(), *3*

rho_from_vopt, 25
rho_from_vopt(), *6*

score_bounds, 26
score_diff_scales, 27
seqcomp *(*seqcomp-package*)*, 2
seqcomp-package, 2
spherical_score, 28
spherical_score(), *3*, *8*
split_streams, 29

tick_loss, 29
tick_loss(), *3*

uniroot(), *21*
unroll_stream, 30

winkler_compare, 31
winkler_compare(), *3*, *8*
winkler_cs, 32
winkler_cs(), *3*
winkler_etest, 33
winkler_etest(), *3*
winkler_score, 34
winkler_score(), *3*