Package: circaTime (via r-universe)

November 23, 2026

Type Package

Title Unified Inference of Circadian Phase from Bulk and Pseudobulk Transcriptomes

Version 0.99.3

Description Provides a single SummarizedExperiment-based interface to published circadian phase (time-of-sample-collection) inference methods for bulk and donor-level pseudobulk transcriptomes, along with a shared cross-validation benchmarking harness, cross-dataset transfer testing, and a calibrated uncertainty layer: bootstrap and Bayesian confidence intervals validated for real coverage, a fixed or dynamically-selected clock-gene quality flag, and consensus phase estimation across every bundled engine. Does not implement a new phase-inference algorithm; unifies existing published methods that are otherwise scattered across incompatible standalone implementations.

License Artistic-2.0

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

biocViews Software, GeneExpression, Transcriptomics, Sequencing, Microarray, TimeCourse, Preprocessing

Depends R (>= 4.4)

Imports graphics, S4Vectors, SummarizedExperiment, methods, stats, utils, withr

Suggests testthat (>= 3.0.0), knitr, rmarkdown, BiocStyle, GEOquery, tauFisher, nnet, TimeSignatR, zeitzeiger, AnnotationDbi, org.Mm.eg.db, org.Hs.eg.db, babelgene, callr, pkgload

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/dralperenuysal/circaTime

BugReports https://github.com/dralperenuysal/circaTime/issues

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.2

Config/pak/sysreqs zlib1g-dev

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

Date/Publication 2026-10-08 09:47:43 UTC

RemoteUrl https://github.com/bioc/circaTime

RemoteRef HEAD

RemoteSha 4790ef8d4a85371303706cd6f2a623750e716f59

Contents

availableReferenceModels . . . 3
bayesianPhasePosterior . . . 4
benchmarkPhase . . . 5
bootstrapPhaseCI . . . 7
calibrationCurve . . . 9
circularAlign . . . 10
circularConcentration . . . 11
circularCor . . . 11
circularCredibleInterval . . . 12
circularDiff . . . 13
circularError . . . 14
circularIntervalCovers . . . 14
circularMAE . . . 15
circularMean . . . 16
circularQuantileInterval . . . 16
circularWeightedMean . . . 17
clockAmplitude . . . 18
clockCoherence . . . 19
coreClockGenes . . . 21
deriveAmplitudeThreshold . . . 22
estimatePhase . . . 23
fitMolecularTimetable . . . 27
fitTauFisher . . . 28
fitTimeSignatR . . . 29
fitZeitZeiger . . . 30
mapOrthologs . . . 32
phaseConfounding . . . 33
PhaseReferenceModel-class . . . 35
plotCalibration . . . 36
predictMolecularTimetable . . . 37
predictTauFisher . . . 38
predictTimeSignatR . . . 39
predictZeitZeiger . . . 40

standardizeGeneIds 41
transferPhase 42
wrapPhase 44

Index **45**

`availableReferenceModels` *List the bundled reference models available to* `estimatePhase()`

Description

List the bundled reference models available to `estimatePhase()`

Usage

```
availableReferenceModels()
```

Details

This list is the exhaustive set of models `estimatePhase()` can use. It is currently `species = "Mmusculus"` only, covering all 12 GSE54650 tissues: Molecular Timetable and tauFisher, plus their `consensus_accuracy`/`consensus_equal` combinations, on `platform = "microarray"`, and Molecular Timetable only on `platform = "rnaseq"`. ZeitZeiger and TimeSignatR are wrapped engines but are deliberately absent here: they are reachable through `fitZeitZeiger()`/`predictZeitZeiger()`, `fitTimeSignatR()`/`predictTimeSignatR()`, `benchmarkPhase()`, and `transferPhase()` rather than as pre-trained `estimatePhase()` models.

Value

A `data.frame`: `species`, `tissue`, `method`, `platform`, `cv_mae` (that model's own held-out cross-validated circular MAE in hours; `NA` if not measured), `n_amp_genes`, `n_residual_draws`. Rows whose `method` starts with `consensus_` are the combined models reachable via `estimatePhase(method = "consensus", consensus_weights = ...)`.

Examples

```
availableReferenceModels()
```

`bayesianPhasePosterior` *Bayesian phase posterior built on the residual-pool bootstrap*

Description

Phase 3 Step 3.3. This does not replace `bootstrapPhaseCI()`/ `circularQuantileInterval()`'s empirical interval - it derives a genuinely different, complementary object from the *same* residual pool: a von Mises posterior predictive density for a sample's true phase, centred on the point estimate, with concentration (`circularConcentration()`) measured directly from that pool rather than assumed. `circularQuantileInterval()` answers "what interval did the observed residuals fall in"; this answers "what is the probability the true phase is any given value" - useful whenever a caller needs more than an interval endpoint, e.g. `P(|error| < 2h)` for a specific downstream decision threshold, or a smooth density to plot.

Usage

```
bayesianPhasePosterior(
  point,
  residualPool,
  level = 0.8,
  period = 24,
  nGrid = 121
)
```

Arguments

`point`	Single numeric point phase estimate (or a vector, one per sample), in `[0, period)` - typically `estimatePhase()`'s own point estimate.
`residualPool`	Numeric vector of real circular residuals (signed deviations from truth, as stored in `PhaseReferenceModel@residualPool`). Concentration is measured from this pool once and shared across every `point` (matching how the reference model's residual pool is tissue/model-wide, not per-sample).
`level`	Nominal credible level for the returned interval (default 0.8).
`period`	Length of the cycle (default 24).
`nGrid`	Number of points in the returned density grid per sample (default 121, i.e. every 12 minutes over a 24h period).

Value

A list with `kappa` (fitted von Mises concentration, shared across all `points`), `concentration` (the underlying mean resultant length, `[0, 1]`), `credible_lo`/`credible_hi` (numeric vectors, one per `point`), and `density` (a list of `data.frame(phase, density)`, one per `point`, each integrating to 1 over `[0, period)`).

Relationship to the residual pool's own coverage

Because the credible interval here is the equal-tailed interval of a *parametric* (unimodal, symmetric) fit to the pool's concentration, while `circularQuantileInterval()`'s interval is the *empirical* quantile interval of the pool itself, the two intervals agree exactly only if the pool really is von Mises-distributed. Real residual pools are approximately but not exactly von Mises (see `benchmarks/run_bayesian_ci_benchmark.R` for a direct comparison on real data) - the intended use is as a complementary posterior summary, not a replacement calibration target for the already-validated empirical interval.

Examples

```
pool <- circularDiff(rnorm(200, sd = 1.5), 0) # a tight, roughly von Mises pool
post <- bayesianPhasePosterior(point = c(6, 18), residualPool = pool)
post$kappa
post$credible_lo
```

`benchmarkPhase` *Cross-validated benchmark of phase-inference engines against ground truth*

Description

Cross-validated benchmark of phase-inference engines against ground truth

Usage

```
benchmarkPhase(
  se,
  truth_col,
  methods = "timetable",
  assay_name = NULL,
  donor_col = NULL,
  nFolds = 5,
  period = 24,
  seed = NULL,
  engineArgs = list()
)
```

Arguments

`se`	A `SummarizedExperiment` with an expression assay (genes x samples) and a ground-truth phase column in `colData`.
`truth_col`	Name of the `colData(se)` column holding ground-truth phase, in `[0, period)`.
`methods`	Character vector of engine names to benchmark: `"timetable"` (Phase 1), `"taufisher"`, `"zeitzeiger"`, and `"timesignatr"` (Phase 2).
`assay_name`	Name of the assay to use; defaults to the first assay.

donor_col	Name of the colData(se) column identifying donor/animal, used for held-out splitting. If NULL (default), each sample is treated as its own donor - only appropriate for datasets with no repeated-measures structure (see inst/scripts/ingest_GSE54650.R for a worked justification).
nFolds	Number of cross-validation folds, split by donor (default 5).
period	Length of the cycle (default 24). Passed through to every engine whose fit accepts a period argument (Molecular Timetable, ZeitZeiger, TimeSignatR), which stores it on the fit object its predict reads back. tauFisher has no period argument and is inherently 24-hour: requesting period != 24 with "taufisher" emits a warning, and its predictions stay on [0, 24).
seed	Optional integer seed for reproducible fold assignment; see .assignDonorFolds() for how the global RNG state is protected.
engineArgs	Named list, keyed by method, of extra arguments to pass to that engine's fit and predict functions separately (they generally take different arguments), e.g. list(timetable = list(fit = list(nGenes = 100), predict = list(minGenes = 5))).

Value

A list with two elements: summary, a data.frame with one row per method (method, circular_mae, circular_cor, n, n_failed); and residuals, a data.frame with one row per (method, held-out sample) (method, sample, donor, fold, truth, pred, circular_error, aligned, fold_failed).

If an engine's fit/predict errors on a given fold (e.g. because that fold's training data happens to violate an engine-specific requirement, such as tauFisher's rhythmicity-detection step needing every clock hour represented), that fold's test samples get pred = NA, fold_failed = TRUE, and a message() is emitted - the rest of that method's folds still run and count normally (n/n_failed in summary report how many held-out samples got a real prediction vs. how many were lost to a fold failure). This never fabricates a result for a failed sample (AGENTS.md Section 0.3).

circular_mae is the primary accuracy number. For evenly-sampled time-course ground truth (e.g. GSE54650), circular_cor can be numerically unstable and even strongly negative for accurate predictions - see the "Known instability" section of circularCor() before relying on it as a benchmark gate.

Examples

```
set.seed(1)
nGenes <- 15; nSamples <- 24
time <- rep(seq(0, 23, length.out = 6), 4)
theta <- time / 24 * 2 * pi
truePhase <- runif(nGenes, 0, 2 * pi)
mat <- t(sapply(seq_len(nGenes), function(i) {
  5 + 2 * cos(theta - truePhase[i]) + rnorm(nSamples, sd = 0.3)
}))
rownames(mat) <- paste0("gene", seq_len(nGenes))
colnames(mat) <- paste0("s", seq_len(nSamples))
se <- SummarizedExperiment::SummarizedExperiment(assays = list(logExpr = mat))
SummarizedExperiment::colData(se)$ZT <- time
bm <- benchmarkPhase(se, truth_col = "ZT", methods = "timetable", nFolds = 4, seed = 1)
bm$summary
```

`bootstrapPhaseCI` *Bootstrap confidence interval for a phase-inference engine's predictions*

Description

Refits the named engine on `nBoot` donor-resampled-with-replacement copies of the training data and predicts `testMat` each time; the spread of those bootstrap predictions gives each test sample a circular confidence interval (via `circularQuantileInterval()`), while a single fit on the original, non-resampled training data gives the reported point estimate. This wraps whichever registered engine's `fit`/`predict` functions (the same `.circaTimeEngines` registry `benchmarkPhase()` uses); it does not invent a new phase-inference method (AGENTS.md 0.3).

Usage

```
bootstrapPhaseCI(
  method,
  trainMat,
  trainTime,
  testMat,
  donors = NULL,
  nBoot = 50,
  level = 0.8,
  seed = NULL,
  fitArgs = list(),
  predictArgs = list(),
  period = 24,
  residualPool = NULL
)
```

Arguments

`method`	Name of a registered engine.
`trainMat`, `trainTime`, `testMat`	As in the engine's own `fit`/`predict`.
`donors`	Optional character/factor vector, one donor ID per training column (`ncol(trainMat)` long); resampling draws whole donors with replacement so repeated-measures structure isn't broken. If `NULL` (default), each training column is its own donor.
`nBoot`	Number of bootstrap replicates (default 50).
`level`	Nominal coverage of the returned interval, in `(0, 1)` (default 0.8).
`seed`	Optional integer seed; RNG state is saved/restored (AGENTS.md Section 10).
`fitArgs`, `predictArgs`	Extra named arguments passed to the engine's `fit`/`predict` respectively.
`period`	Length of the cycle (default 24).

`residualPool` Optional numeric vector of independent circular residuals (each in `(-period/2, period/2]`, e.g. from `circularDiff()`) to fold into the bootstrap draws - see the section above. `NULL` (default) reproduces the training-resampling-only interval, which AGENTS.md Phase 3's own calibration check found to be overconfident on real data; supplying a residual pool from an independent benchmark run is strongly recommended for any interval that will actually be reported to a user.

Details

A bootstrap replicate's `fit`/`predict` call is allowed to fail (e.g. tauFisher's MetaCycle step erroring when a resample happens to drop every replicate of one clock hour - the same failure mode `benchmarkPhase()` isolates per-fold). Failed replicates are dropped from that sample's interval rather than aborting the whole bootstrap (AGENTS.md 0.3: report the failure, do not fabricate or crash).

Value

A `data.frame` with one row per test sample: `sample`, `phase` (point estimate from the original unresampled fit), `phase_lo`, `phase_hi`, `level`, `n_boot_ok`, `n_boot_failed`. The raw per-replicate draws are attached as `attr(., "draws")`, an `ncol(testMat) x nBoot` matrix (`NA` for failed replicates), so `calibrationCurve()` can recompute intervals at other levels without refitting anything.

Why `residualPool` matters

Refitting on resampled training data only captures how sensitive the fit is to *which training donors were used* - it says nothing about the error a genuinely new test sample's own biological noise or the model's irreducible bias contributes, which on real data is usually the dominant source of error. Verified directly on GSE54650 (mouse 12-tissue atlas): a training-resampling-only bootstrap for the `timetable` engine gave a stated 80% interval that actually covered truth only ~48% of the time - badly overconfident, exactly the failure mode AGENTS.md Phase 3 warns about ("uncalibrated intervals are worse than none"). Supplying `residualPool` - an independent sample of real circular residuals (e.g. `circularDiff(pred, truth)` from a prior cross-validated benchmark run on data the current `testMat` was not scored against) adds one resampled residual to every bootstrap draw, folding that test-time noise back into the interval. This is a standard prediction-interval technique (residual/ bagging bootstrap, as used e.g. for random forest prediction intervals), not a new phase-inference method (AGENTS.md 0.3).

Examples

```
set.seed(1)
nGenes <- 15; nSamples <- 20
trainTime <- runif(nSamples, 0, 24)
theta <- trainTime / 24 * 2 * pi
truePhase <- runif(nGenes, 0, 2 * pi)
trainMat <- t(sapply(seq_len(nGenes), function(i) {
  5 + 2 * cos(theta - truePhase[i]) + rnorm(nSamples, sd = 0.3)
}))
rownames(trainMat) <- paste0("gene", seq_len(nGenes))
ci <- bootstrapPhaseCI("timetable", trainMat, trainTime, trainMat[, 1:5, drop = FALSE],
  nBoot = 5, seed = 1
```

```
)
ci
```

`calibrationCurve` *Empirical coverage of bootstrap intervals across nominal levels*

Description

Recomputes `circularQuantileInterval()` at each of `levels` from the raw bootstrap draws already computed by `bootstrapPhaseCI()` (its `"draws"` attribute), and reports what fraction of the time the interval actually contains the known ground truth. This is the calibration check AGENTS.md Phase 3 task 3 requires before a stated interval can be trusted at all: "uncalibrated intervals are worse than none."

Usage

```
calibrationCurve(ci, truth, levels = c(0.5, 0.7, 0.8, 0.9, 0.95), period = 24)
```

Arguments

`ci`	A `data.frame` returned by `bootstrapPhaseCI()` (or `rbind()` of several such calls made with the same `nBoot`), carrying a `"draws"` attribute.
`truth`	Numeric vector of ground-truth phases, same length and order as `nrow(ci)`.
`levels`	Numeric vector of nominal coverage levels to evaluate (default `c(0.5, 0.7, 0.8, 0.9, 0.95)`).
`period`	Length of the cycle (default 24).

Value

A `data.frame` with one row per level: `nominal`, `empirical` (observed coverage fraction), `n` (non-`NA` intervals evaluated).

Examples

```
set.seed(1)
nGenes <- 15; nSamples <- 20
trainTime <- runif(nSamples, 0, 24)
theta <- trainTime / 24 * 2 * pi
truePhase <- runif(nGenes, 0, 2 * pi)
trainMat <- t(sapply(seq_len(nGenes), function(i) {
  5 + 2 * cos(theta - truePhase[i]) + rnorm(nSamples, sd = 0.3)
}))
rownames(trainMat) <- paste0("gene", seq_len(nGenes))
ci <- bootstrapPhaseCI("timetable", trainMat, trainTime, trainMat[, 1:5, drop = FALSE],
  nBoot = 5, seed = 1
)
calibrationCurve(ci, truth = trainTime[1:5], levels = c(0.5, 0.8))
```

`circularAlign` *Align unsupervised phase estimates to a reference by rotation and reflection*

Description

Unsupervised engines (e.g. CYCLOPS-family, CHIRAL) recover sample ordering around the circle only up to an arbitrary rotation and reflection (AGENTS.md 7.2). This finds the rotation offset and reflection (out of the two possible orientations) that best aligns `est` to `ref`, then returns the aligned estimate.

Usage

```
circularAlign(est, ref, period = 24, allow_reflection = TRUE)
```

Arguments

`est`	Numeric vector of unsupervised phase estimates, in `[0, period)`.
`ref`	Numeric vector of reference/ground-truth phases, same length as `est`. `NA` entries are excluded from fitting the alignment but the alignment is still applied to every element of `est`.
`period`	Length of the cycle (default 24).
`allow_reflection`	Logical; if `TRUE` (default), also try reversing the direction of `est` and keep whichever orientation gives lower circular MAE against `ref`.

Details

The rotation for each orientation is estimated as the circular mean of the pairwise differences to `ref`; this is the standard closed-form alignment used in the CYCLOPS/CHIRAL literature and is a close practical approximation to the rotation that minimises circular MAE.

Value

A list with elements `aligned` (numeric vector, same length as `est`), `rotation` (the applied additive offset), `reflected` (logical), and `mae` (the circular MAE of `aligned` vs `ref`, over the non-`NA` fitting subset).

Examples

```
truth <- c(0, 4, 8, 12, 16, 20)
est <- wrapPhase(-truth + 5) # reflected and rotated
circularAlign(est, truth)$rotation
```

circularConcentration *Mean resultant length of a set of phases (circular concentration)*

Description

The standard circular-statistics measure of how tightly a sample of phases clusters around its own circular mean: 1 for perfect agreement, 0 for phases spread uniformly around the whole circle. Equivalent to `1 - circular variance`. This is the primitive `circularWeightedMean()`'s `"concentration"` attribute already computes internally for a *weighted combination of engine estimates*; this function computes the same quantity directly for any raw sample of phases (or phase deviations), which is what Phase 3 Step 3.3's Bayesian posterior (see `bayesianPhasePosterior()`) needs applied to a reference model's residual pool.

Usage

```
circularConcentration(x, period = 24, na.rm = TRUE)
```

Arguments

`x`	Numeric vector of phase values (or circular deviations, e.g. from `circularDiff()`).
`period`	Length of the cycle (default 24).
`na.rm`	Logical; drop `NA` values before computing (default `TRUE`).

Value

A single numeric value in `[0, 1]`. `NA_real_` if no usable input remains.

Examples

```
circularConcentration(c(0, 0, 0)) # 1: no spread at all
circularConcentration(c(0, 6, 12, 18)) # ~0: spread evenly around the circle
```

circularCor *Circular-circular correlation coefficient (Jammalamadaka & SenGupta, 2001)*

Description

The circular analogue of Pearson correlation. Must be used instead of `cor()` whenever either input is a phase (AGENTS.md 7.1).

Usage

```
circularCor(x, y, period = 24)
```

Arguments

x, y	Numeric vectors of phase values (same units, same period).
period	Length of the cycle (default 24).

Value

A single numeric value in [-1, 1].

Known instability on uniformly-sampled time courses

This coefficient centres each variable on its own circular mean before comparing them. When one variable's values are (near-)evenly spaced around the *entire* circle - e.g. ground truth from a time course sampled every 2h across a full 24h day, such as GSE54650 - that variable's resultant vector length is ~0, its circular mean becomes numerically arbitrary (driven by floating-point noise, not signal), and the resulting correlation can come out strongly negative *even for near-perfect predictions*. This is a property of the Jammalamadaka-SenGupta coefficient itself, not a bug: verified directly on GSE54650-shaped ground truth (24 evenly-spaced timepoints), a prediction set with 0.2h mean circular error still yields a coefficient around -0.46. circularMAE() does not share this failure mode and should be treated as the primary accuracy metric for evenly-sampled benchmark data; treat circularCor as a secondary/diagnostic number on such data, not a reliability gate.

Examples

```
circularCor(c(0, 6, 12, 18), c(1, 7, 13, 19))
```

---

circularCredibleInterval

Bayesian (von Mises) credible interval around a point phase estimate

---

Description

The parametric counterpart to circularQuantileInterval(): instead of taking empirical quantiles of a finite set of bootstrap draws, this fits a von Mises distribution - the circular analogue of the normal distribution, and the standard choice for Bayesian circular models (Fisher 1993) - to the concentration already measured from real data, and returns the analytic credible interval of that distribution centred on mu. Because a von Mises distribution is unimodal and symmetric, the equal-tailed interval computed here is also the shortest (highest-density) interval at the same level - no separate HDI computation is needed.

Usage

```
circularCredibleInterval(mu, kappa, level = 0.8, period = 24)
```

Arguments

mu	Single numeric point phase estimate (or a vector, recycled per sample), in [0, period).
kappa	Von Mises concentration parameter (non-negative). See circularConcentration() + the internal R-to-kappa conversion, or supply a value fit elsewhere. kappa = 0 is the uniform distribution (maximally uninformative posterior - the interval spans the whole circle).
level	Nominal credible level (default 0.8, matching circularQuantileInterval()'s default so the two are directly comparable).
period	Length of the cycle (default 24).

Value

A list with lo, hi (interval bounds in [0, period), same length as mu).

Examples

```
circularCredibleInterval(mu = 12, kappa = 5, level = 0.8)
```

---

circularDiff *Signed circular difference between two phases*

Description

Signed circular difference between two phases

Usage

```
circularDiff(pred, truth, period = 24)
```

Arguments

pred, truth	Numeric vectors of phase values (same units, same period).
period	Length of the cycle (default 24).

Value

Numeric vector of signed differences in (-period/2, period/2], i.e. the shortest signed rotation from truth to pred.

Examples

```
circularDiff(0.5, 23.5) # 1, not -23
```

circularError *Per-sample circular absolute error*

Description

Vectorised version of the error term inside `circularMAE()`, useful for residual analysis (e.g. `benchmarkPhase()` per-sample output).

Usage

```
circularError(pred, truth, period = 24)
```

Arguments

`pred, truth`	Numeric vectors of phase values in `[0, period)`. `NA` entries are dropped pairwise before averaging.
`period`	Length of the cycle (default 24, for hours).

Value

Numeric vector of per-sample circular absolute errors.

Examples

```
circularError(c(0.5, 12), c(23.5, 0))
```

circularIntervalCovers *Does a circular interval cover a truth value?*

Description

Tests whether `truth` lies on the arc walked from `lo` to `hi` in the increasing direction (wrapping past `period` if needed) - i.e. the same direction `circularQuantileInterval()` builds its interval in. A zero-width interval (`lo == hi`) only covers an exact match, not the whole circle.

Usage

```
circularIntervalCovers(truth, lo, hi, period = 24)
```

Arguments

`truth`	Numeric vector of ground-truth phase values.
`lo, hi`	Numeric vectors of interval bounds, same length as `truth` (as returned by `circularQuantileInterval()` per sample).
`period`	Length of the cycle (default 24).

Value

Logical vector, NA wherever truth, lo, or hi is NA.

Examples

```
circularIntervalCovers(23.9, 23, 1) # TRUE: 23.9 is on the 23 -> 1 arc
circularIntervalCovers(12, 23, 1)   # FALSE
```

---

circularMAE *Circular mean absolute error*

Description

The primary error metric for this package (AGENTS.md 7.1). Computed as pmin(abs(d), period - abs(d)) where d = pred - truth, so that e.g. a prediction of 23.5 against a truth of 0.5 counts as 1 hour of error, not 23.

Usage

```
circularMAE(pred, truth, period = 24, na.rm = TRUE)
```

Arguments

pred, truth	Numeric vectors of phase values in [0, period). NA entries are dropped pairwise before averaging.
period	Length of the cycle (default 24, for hours).
na.rm	Logical; if FALSE (default TRUE), NA in either input makes the result NA instead of being dropped.

Value

A single numeric value: the mean circular absolute error.

Examples

```
circularMAE(c(0.5, 12), c(23.5, 0))
```

circularMean *Circular mean direction*

Description

Circular mean direction

Usage

```
circularMean(x, period = 24, na.rm = TRUE)
```

Arguments

x	Numeric vector of phase values.
period	Length of the cycle (default 24).
na.rm	Logical; drop NA values before averaging (default TRUE).

Value

A single numeric value in [0, period): the mean direction of x.

Examples

```
circularMean(c(23, 1)) # ~0, not 12
```

circularQuantileInterval *Circular quantile interval around a set of phase draws*

Description

Used by bootstrapPhaseCI() to turn a sample's bootstrap replicate predictions into a phase_lo/phase_hi interval. Centres the draws on their circular mean, takes the empirical (1-level)/2 and 1-(1-level)/2 quantiles of the *signed* circular deviation from that centre (via circularDiff(), so a deviation is always in (-period/2, period/2] and wraparound never inflates the spread), then maps those quantile deviations back onto the circle. This assumes the draws are unimodal around their circular mean, which holds for a bootstrap distribution of predictions clustered near the true phase, but would not hold for, say, a bimodal rotation/reflection-ambiguous set (AGENTS.md 7.2) - resolve that ambiguity before calling this.

Usage

```
circularQuantileInterval(x, level = 0.8, period = 24)
```

Arguments

x	Numeric vector of phase draws, in [0, period); NA entries are dropped.
level	Nominal interval coverage, in (0, 1) (default 0.8).
period	Length of the cycle (default 24).

Value

A list with center (circular mean of x), lo, hi (interval bounds in [0, period)). All NA_real_ if fewer than 2 non-NA draws remain.

Examples

```
circularQuantileInterval(c(23, 0, 1, 23.5, 0.5))
```

---

circularWeightedMean *Weighted circular mean direction*

Description

The weighted generalisation of circularMean(): each phase is converted to a unit vector, the vectors are summed with the supplied weights, and the direction of the resultant is returned. Used by estimatePhase()'s method = "consensus" to combine several engines' phase estimates (AGENTS.md Section 6 Step 3.1). circularWeightedMean(x, rep(1, length(x))) is identical to circularMean(x).

Usage

```
circularWeightedMean(x, w, period = 24, na.rm = TRUE)
```

Arguments

x	Numeric vector of phase values.
w	Numeric vector of non-negative weights, same length as x. Weights are normalised internally, so only their relative sizes matter.
period	Length of the cycle (default 24).
na.rm	Logical; drop pairs where x is NA before averaging (default TRUE). Weights of dropped entries are dropped with them, and the remainder renormalised.

Details

The resultant vector's *length* is also returned (as attribute "concentration", in [0, 1]): 1 means every input phase agreed exactly, 0 means they cancelled out completely. This is the standard circular measure of agreement, and for a consensus estimate it is a directly interpretable "did the engines agree?" signal rather than a hidden intermediate.

Value

A single numeric value in `[0, period)`, with attribute `"concentration"` (numeric, `[0, 1]`). `NA_real_` (concentration `NA`) if no usable input remains or all weights are zero.

Examples

```
# two engines disagreeing by 2h, the second trusted twice as much
circularWeightedMean(c(6, 8), c(1, 2))
# perfect agreement -> concentration 1
attr(circularWeightedMean(c(6, 6), c(1, 1)), "concentration")
```

---

clockAmplitude *Per-sample core-clock amplitude*

Description

Tissue-normalised amplitude of the AGENTS.md 7.7 core clock gene set, used to flag samples where the circadian clock is too damped for a phase estimate to be meaningful (AGENTS.md 7.4). Each matched gene is z-scored within its tissue group (so amplitude is comparable across tissues with different baseline expression/variance), then the z-scored block is reduced with PCA; amplitude is the Euclidean norm of a sample's first two PC scores. This mirrors the "eigengene" amplitude used in the CYCLOPS/circadian-transcriptomics literature (Anafi et al. 2017) for the same QC purpose: PC1/PC2 capture the near-circular manifold that a co-oscillating gene set traces out across a cohort, so their combined magnitude reflects how strongly a given sample's clock genes co-oscillate, independent of what phase they point to. This is a QC/amplitude summary, not a new phase-inference algorithm (AGENTS.md 0.3).

Usage

```
clockAmplitude(exprMat, tissue = NULL, minGenes = 4)
```

Arguments

`exprMat`	Genes x samples numeric matrix, log-scale, with gene symbol rownames (any case; matched case-insensitively against `coreClockGenes()`).
`tissue`	Optional character/factor vector, one tissue label per sample (`ncol(exprMat)` long). If supplied, genes are z-scored within each tissue group separately before PCA. If `NULL` (default), all samples are treated as one group.
`minGenes`	Minimum number of matched core clock genes required (default 4); errors below this since amplitude from too few genes is not a meaningful summary.

Value

A numeric vector, `ncol(exprMat)` long, of non-negative amplitude values, with attributes `genesUsed` (character vector of matched rownames), `varExplained` (named numeric, fraction of variance captured by PC1+PC2 per tissue group), and `center`/`scale`/`rotation` (named lists, one element per tissue group: the `scale()` center/scale vectors and `prcomp()` rotation matrix used for that group's PCA) - the latter three let a caller reproduce this exact projection for a *new* sample against an already-fitted reference group, which is what `estimatePhase()` does for small (often n=1) inference calls where there aren't enough samples in the call itself to fit a fresh PCA. `NA` for samples in a tissue group with fewer than 3 members (too few for PCA/z-scoring to be meaningful).

Known limitation — validated on real data, effect not yet confirmed

AGENTS.md 7.4 expects this amplitude to be low "in aged tissue, tumours, and critically ill donors," and Phase 3's gate requires showing flagged samples have demonstrably worse phase error. On the two labelled real datasets tested so far this effect was **not** statistically significant: GSE54650 (12-tissue healthy mouse atlas, no aged/damped-clock samples; p=0.083) and GSE84511 (young vs aged mouse epidermal stem cells, real ZT ground truth; p=0.20 for the derived threshold, and mean amplitude barely differs by age group, p=0.74 - see `benchmarks/results/phase3_amplitude_threshold_GSE84511_time` and `notes/decisions.md`). The likely reason, per GSE84511's source paper (Solanas et al. 2017 Cell) and its companion liver study (Sato et al. 2017 Cell): aging leaves core clock GENE expression population-average "virtually unaltered" and instead reprograms which downstream genes oscillate - so a core-clock-gene PCA amplitude may not be the right proxy for the kind of age-related damping the literature actually reports, even though aged samples in GSE84511 did have ~2.5x higher real cross-validated phase error than young ones. Kept as documented, honest scope for now rather than silently dropped or rounded up (AGENTS.md 0.3); revisit if a dataset or amplitude definition surfaces that separates the two groups.

Examples

```
set.seed(1)
genes <- c("Arntl", "Per1", "Per2", "Cry1", "Nr1d1", "Dbp")
mat <- matrix(rnorm(6 * 20), nrow = 6, dimnames = list(genes, paste0("s", 1:20)))
amp <- clockAmplitude(mat)
```

`clockCoherence` *Dynamic per-sample circadian coherence (Phase 3 QC)*

Description

The Phase 3 Step 3.2 replacement for the fixed 16-gene amplitude flag (`clockAmplitude()`). Rather than scoring every dataset against the same core-clock panel, this selects the informative gene panel *from the input dataset itself*: each gene is fitted with a cosinor against the cohort's (estimated) phases, and the panel is the set of genes whose fit beats a phase-permuted null at a chosen FDR. A sample's coherence is then the fraction of its own deviation-from-gene-mean that the cohort's fitted rhythm explains - 1 for a sample sitting exactly on the cohort's circadian manifold, near 0 (or negative) for a sample whose clock genes are damped, desynchronised, or simply inconsistent with the phase it was assigned.

Usage

```
clockCoherence(
  exprMat,
  phase,
  tissue = NULL,
  period = 24,
  nPerm = 200,
  fdr = 0.05,
  minPanel = 20,
  minSamples = 10,
  seed = 1
)
```

Arguments

exprMat	Genes x samples numeric matrix, log-scale, with gene rownames.
phase	Numeric vector of phases in hours, `ncol(exprMat)` long, usually `estimatePhase()`'s output. `NA` phases drop those samples from panel selection and receive `NA` coherence.
tissue	Optional character/factor vector, one label per sample. If supplied, genes are z-scored and the panel is selected within each tissue group separately (as in `clockAmplitude()`).
period	Rhythm period in hours (default 24).
nPerm	Number of phase permutations forming the null (default 200).
fdr	Benjamini-Hochberg FDR for panel selection (default 0.05).
minPanel	Minimum panel size (default 20). If fewer genes clear the null, the top `minPanel` genes by cosinor R-squared are used instead and the `relaxed` attribute is set for that group.
minSamples	Minimum samples per group (default 10); smaller groups get `NA`.
seed	Seed for the permutation null, so the panel is reproducible.

Value

A numeric vector, `ncol(exprMat)` long, of coherence scores (1 = perfect fit to the cohort rhythm, lower = worse; can be negative). Attributes: `panel` (named list, selected gene symbols per group), `panelSize` (named integer), `relaxed` (named logical, `TRUE` where the null-based selection fell through to the top-`minPanel` fallback), `r2Cutoff` (named numeric, the smallest R-squared admitted, `NA` when relaxed).

Why replace the fixed panel

The legacy flag scores amplitude on 16 core clock genes. Its documented failure (see `clockAmplitude()`) is that aging and similar insults leave core clock gene *expression* close to unaltered while reprogramming which downstream genes oscillate - so a fixed core-clock panel is measuring the wrong thing. Selecting the panel per dataset lets those reprogrammed, dataset-specific oscillators carry the QC signal.

Circularity - read before interpreting

`phase` is normally the *estimated* phase from `estimatePhase()`, not ground truth, so a sample whose phase was estimated badly will fit the cohort cosinor badly and score low. That is the intended behaviour (the flag is meant to catch exactly those samples) but it does mean coherence is **not** an independent measurement of data quality - it is a self-consistency score. Validation against held-out true phase (`benchmarks/run_dynamic_qc_benchmark.R`) is therefore the only claim made for it here; do not read it as a general sample-quality metric. Supplying known sampling times as `phase` removes the circularity and turns it into a pure rhythm-strength score.

Cohort-level by construction

Unlike the legacy flag, this needs a cohort: the panel and the fitted rhythm are both estimated across samples. Below `minSamples` (default 10) it returns all `NA` rather than a number computed from too little data, and `estimatePhase()` falls back to the legacy flag in that case. This is a deliberate design choice over storing a fixed dynamic basis in the bundled reference models - a stored panel would be "dynamic" only with respect to the training set, which is the very thing Step 3.2 sets out to remove.

See Also

`clockAmplitude()` for the legacy fixed-panel flag, `deriveAmplitudeThreshold()` for turning either score into a cutoff.

Examples

```
set.seed(1)
nGenes <- 200; nSamples <- 24
t <- seq(0, 23, length.out = nSamples)
truePhase <- runif(nGenes, 0, 24)
mat <- t(vapply(seq_len(nGenes), function(g) {
  2 * cos(2 * pi * (t - truePhase[g]) / 24) + rnorm(nSamples, sd = 1)
}, numeric(nSamples)))
dimnames(mat) <- list(paste0("g", seq_len(nGenes)), paste0("s", seq_len(nSamples)))
coh <- clockCoherence(mat, phase = t)
summary(coh)
```

---

coreClockGenes *The bundled core clock gene reference table*

Description

Returns the AGENTS.md 7.7 core clock gene set, one row per gene, with common aliases and a source citation. Matching against expression data rownames is always case-insensitive (see `clockAmplitude()`) - the mouse orthologs of every gene in this set share the same letters as the human symbol, differing only in case, so a single table serves both species without a separate mouse column.

Usage

```
coreClockGenes()
```

Value

A `data.frame` with columns `symbol` (canonical human HGNC symbol), `aliases` (semicolon-separated), `source` (citation).

Examples

```
head(coreClockGenes())
```

---

deriveAmplitudeThreshold

Empirically derive a low-amplitude flag threshold

Description

Finds the `clockAmplitude()` cutoff below which phase error is meaningfully worse, from held-out labelled data - AGENTS.md 7.4/Phase 3 task 4 requires this be derived empirically, not from intuition.

Usage

```
deriveAmplitudeThreshold(amp, error, minLowFrac = 0.05)
```

Arguments

`amp`	Numeric vector of `clockAmplitude()` values.
`error`	Numeric vector of per-sample circular error (e.g. from `circularError()` against known ground truth), same length as `amp`.
`minLowFrac`	Minimum fraction of samples the `low` group must contain (default 0.05).

Details

Candidate thresholds are deciles of `amp` (10th-50th percentile). For each candidate, samples split into `low` (`amp <= candidate`) and `high` (`amp > candidate`) groups; the candidate is scored by a one-sided Wilcoxon rank-sum test of `error(low) > error(high)`, converted to a common-language effect size (probability a random low-group error exceeds a random high-group error, in `[0, 1]`), subject to the `low` group holding at least `minLowFrac` of samples - too small a `low` group is not a reliable flag, however well it happens to separate this particular dataset. The candidate with the highest effect size among those satisfying the fraction constraint is returned.

Value

A list: `threshold` (chosen amplitude cutoff), `effect_size` (common-language effect size at that cutoff), `p_value` (Wilcoxon test p-value), `low_mae`/`high_mae` (mean error in each group), `low_frac` (fraction of samples flagged `low_amplitude`).

Examples

```
set.seed(1)
amp <- c(runif(15, 0, 1), runif(15, 2, 4))
error <- c(runif(15, 3, 5), runif(15, 0, 1))
deriveAmplitudeThreshold(amp, error)
```

`estimatePhase` *Estimate circadian phase for new samples against a bundled reference model*

Description

The package's main entry point (AGENTS.md Section 6). Looks up a pre-trained PhaseReferenceModel for the requested (`species`, `tissue`, `method`, `platform`) - see `availableReferenceModels()` for what is currently bundled - predicts phase with that engine's own `predict` function, attaches a circular confidence interval built from the reference model's real held-out residual pool (no live bootstrap refit at call time - see PhaseReferenceModel), a core-clock amplitude projected onto the reference cohort's PCA basis (works even for a single-sample call, unlike a fresh `clockAmplitude()` fit), and a QC flag.

Usage

```
estimatePhase(
  se,
  method,
  tissue,
  species = "Mmusculus",
  platform = "microarray",
  consensus_weights = c("accuracy", "equal"),
  assay_name = NULL,
  level = 0.8,
  qc_method = c("legacy", "dynamic"),
  qc_threshold = 0,
  qc_min_samples = 10,
  map_aliases = TRUE,
  data_species = species,
  map_orthologs = !identical(data_species, species)
)
```

Arguments

`se`	A `SummarizedExperiment` with an expression assay, genes x samples.
`method`	Name of a bundled engine for the requested tissue/species (see `availableReferenceModels()`); currently `"timetable"` or `"taufisher"`, or `"consensus"` to combine every engine bundled for that tissue/species/platform (see the Consensus section below).

`tissue` Required, no default (AGENTS.md 7.5) - a tissue code from `availableReferenceModels()` for the requested species (GSE54650's mouse tissue abbreviations, e.g. `"Liv"` for liver - see `inst/scripts/ingest_GSE54650.R`).

`species` Default `"Mmusculus"` - species of the bundled reference model to use. See the section above.

`platform` Default `"microarray"` - the data type the bundled reference model was trained on (AGENTS.md Section 5 Step 2.3), e.g. `"microarray"` (GSE54650) or `"rnaseq"` (GSE54651, the RNA-seq counterpart of the same study/tissues). Not inferred from `se` or from `data_species/map_orthologs` - pick the platform whose reference model matches (or is closest to) how your own data was generated; see `availableReferenceModels()` for what is bundled.

`consensus_weights` Only used when `method = "consensus"`; one of `"accuracy"` (default) or `"equal"`. See the Consensus section below. These are two separately-trained models, not one model with a runtime switch: each carries its own held-out residual pool measured under that weighting, since an accuracy-weighted and an equally-weighted consensus do not have the same error distribution and must not share a confidence interval.

`assay_name` Name of the assay to use. If `NULL` (default), uses the assay named `"logExpr"` if present (every bundled reference model is trained on log-scale expression), else the first assay.

`level` Nominal coverage of `phase_lo`/`phase_hi` (default 0.8).

`qc_method` `"legacy"` (default) uses the fixed 16-gene core-clock amplitude (`clockAmplitude()`) projected onto the bundled reference model's training-cohort PCA basis, with the threshold fit once during training (`deriveAmplitudeThreshold()`). `"dynamic"` uses `clockCoherence()` instead: the informative gene panel is selected from `se` itself rather than a fixed panel (Phase 3 Step 3.2), which needs a real cohort of samples in this call (see `qc_min_samples`) rather than a stored basis, since the whole point is that the panel is dataset-specific. See `clockCoherence()` for why this scores self-consistency with the (usually estimated, not true) `phase` column, and `benchmarks/run_dynamic_qc_benchmark.R` for validation against the legacy flag on real labelled data.

`qc_threshold` Only used when `qc_method = "dynamic"`. Coherence values at or below this are flagged `"low_amplitude"`. Default `0`: a coherence of 0 means the fitted cohort rhythm explains no more of this sample's panel than its own per-gene mean would - a model-free zero point, not fit to any one dataset. `benchmarks/run_dynamic_qc_benchmark.R` shows tightening this (e.g. to `0.1`) sharpens separation on GSE84511; left as a scale-appropriate default here rather than a per-tissue value snooped from one benchmark.

`qc_min_samples` Only used when `qc_method = "dynamic"`; minimum samples in `se` for `clockCoherence()`'s per-dataset panel selection to be meaningful (default 10, matching `clockCoherence()`'s own default). Below this, `qc_method` silently falls back to `"legacy"` for this call and `metadata(se)$circaTime_qc$method_used` records the fallback - a cohort this small cannot support fitting its own rhythm, so "dynamic" would just be noise dressed up as a QC score.

`map_aliases` Logical; if `TRUE` (default), normalize common clock-gene aliases (for example `BMAL1` -> `ARNTL/Arntl`) and strip Ensembl version suffixes before matching genes to the reference model. A mapping summary is stored in `metadata(se)$circaTime_gene_mappin`

`data_species` Species of `rownames(se)` itself, default equal to `species` (the common case: data and model are the same species). Set this when `se`'s genes are in a *different* species' symbol convention than the requested model - e.g. human input data (`data_species = "Hsapiens"`) against the bundled mouse model (`species = "Mmusculus"`, the only one currently bundled).

`map_orthologs` Logical, default `TRUE` whenever `data_species` differs from `species`. Ortholog-maps `se`'s genes from `data_species` to `species` via `mapOrthologs()` (one-to-one only; genes with no unambiguous ortholog are dropped, not guessed at) before the usual gene-matching step. A mapping summary is stored in `metadata(se)$circaTime_ortholog_mapping`. Setting this `FALSE` while `data_species != species` is a hard error rather than silently comparing mismatched-species gene symbols.

Details

Per the Section 6 contract, this never hard-errors on a per-sample basis - a sample that can't be scored gets `phase = NA` and an informative `phase_flag`. Hard errors are reserved for malformed input: an unrecognised species/tissue/method combination, or an `se` missing gene rownames.

Value

`se` with `colData` columns added: `phase`, `phase_lo`, `phase_hi` (empirical residual-pool interval), `phase_credible_lo`, `phase_credible_hi` (the Phase 3 Step 3.3 von Mises credible interval built on the same residual pool - see `bayesianPhasePosterior()`; `metadata(se)$circaTime_bayesian` records the fitted concentration), `clock_amp`, `phase_flag` (factor: `"ok"` / `"low_amplitude"` / `"gene_coverage"` / `"failed"`). `clock_amp` holds whichever score `qc_method` produced (amplitude or coherence - `metadata(se)$circaTime_qc` records which). For `method = "consensus"`, `metadata(se)$circaTime_consensus` additionally records the per-engine estimates and the weights used (see the Consensus section).

What this does NOT do (yet)

Only `species = "Mmusculus"` is currently bundled, trained from GSE54650 (`platform = "microarray"`) and GSE54651 (`platform = "rnaseq"`, its RNA-seq counterpart - see `inst/scripts/train_reference_models.R`) - calling with any other species/tissue/method/platform combination is a hard error listing what IS available, not a silent guess (AGENTS.md 7.5/0.3). `platform` is never inferred from `se` itself; picking the wrong one does not error, it just scores against a model trained on a different assay technology than your data. Gene matching is case-insensitive symbol matching, the same convention `clockAmplitude()`/`coreClockGenes()` already use. `data_species`/`map_orthologs` (AGENTS.md Section 4 Step 1.2) let `se` carry genes in a *different* species' symbol convention than the bundled model - see those arguments below - but this remains one-to-one symbol/ortholog matching, not full Ensembl-ID mapping within a single species (see `standardizeGeneIds()` for that). A larger multi-species/tissue reference cohort (the `circaTimeData` ExperimentHub package, AGENTS.md Section 9 Phase 4 task 4) is future work, not silently substituted here.

Bayesian credible interval (Phase 3 Step 3.3)

Alongside the empirical `phase_lo`/`phase_hi` interval (residual-pool bootstrap quantiles), every call also returns `phase_credible_lo`/`phase_credible_hi`: a von Mises credible interval at the same `level`, fit from the same reference model's residual pool via `bayesianPhasePosterior()` rather than a second, independent computation. The two intervals answer related but distinct questions - the empirical one is "what interval did the observed residuals actually span", the Bayesian one is "what is a smooth probability distribution over the true phase" - and are expected to be close but not identical (see `?bayesianPhasePosterior`'s Relationship section for when and why they can diverge). `metadata(se)$circaTime_bayesian` records the fitted `kappa` and `concentration`; call `bayesianPhasePosterior()` directly for the full posterior density rather than just the interval endpoints.

Consensus (`method = "consensus"`)

Rather than making the user pick one engine, `method = "consensus"` runs every engine bundled for that (species, tissue, platform) and combines their phase estimates on the circle via `circularWeightedMean()` (AGENTS.md Section 6 Step 3.1). Two weighting schemes are available via `consensus_weights`:

- `"accuracy"` (default) - **tissue-aware**: each engine is weighted by `1/cvMAE`, where `cvMAE` is that engine's own held-out cross-validated circular MAE *on this specific tissue*, measured during training and stored on the model (see PhaseReferenceModel). An engine that is more accurate on liver than on brainstem is therefore weighted differently in each.
- `"equal"` - a plain circular mean of the engines' estimates, with no accuracy prior at all. Useful as the honest baseline for whether the accuracy weighting is buying anything.

The weights are not hidden: `metadata(se)$circaTime_consensus` records the engines used, the `stored_weights` (the trained prior) and the `effective_weights` actually applied to this call, each engine's own per-sample `per_engine_phase`, the per-sample `concentration` (how strongly the engines agreed, in `[0, 1]`) and `n_engines_used`. An engine that fails its gene-coverage check on your data has its weight set to zero and the remainder renormalised, rather than contributing an unusable phase; the consensus is flagged `"gene_coverage"` only if *no* engine had coverage.

What consensus is available over is limited by what is bundled: mouse microarray has both Molecular Timetable and tauFisher, so consensus there combines two engines. ZeitZeiger and TimeSignature are deliberately not bundled as pre-trained models (their fitted objects are two orders of magnitude larger for no measured accuracy benefit), and the mouse RNA-seq platform bundles Molecular Timetable only, so no consensus model exists for it - requesting one is an error listing what does exist, not a silent fallback to a single engine.

Examples

```
mat <- matrix(rnorm(200), nrow = 100, dimnames = list(paste0("g", 1:100), c("s1", "s2")))
se <- SummarizedExperiment::SummarizedExperiment(assays = list(logExpr = mat))
# random genes won't overlap the bundled model's trained gene set - this shows
# the "never hard-error on a sample" contract: phase_flag is set, not an error.
se <- estimatePhase(se, method = "timetable", tissue = "Liv")
SummarizedExperiment::colData(se)$phase_flag
```

`fitMolecularTimetable` *Fit a Molecular Timetable model from labelled training data*

Description

Fit a Molecular Timetable model from labelled training data

Usage

```
fitMolecularTimetable(
  trainMat,
  trainTime,
  period = 24,
  nGenes = 100,
  minCor = 0.5
)
```

Arguments

`trainMat`	Numeric matrix of expression, genes x samples (log-scale expression is assumed, as for the harmonic-regression fit to be meaningful).
`trainTime`	Numeric vector of known circadian phase (same units as `period`, one value per column of `trainMat`).
`period`	Length of the cycle (default 24, for hours).
`nGenes`	Maximum number of time-indicating genes to keep, ranked by fit quality (Pearson correlation between fitted and observed expression). Default 100, following the general magnitude used in the original method.
`minCor`	Minimum per-gene fit correlation required to be eligible as a time-indicating gene, evaluated before the `nGenes` cap. Default 0.5.

Value

An object of class `"molecularTimetableFit"`: a list with the selected gene names (`genes`), their harmonic-fit coefficients (`mesor`, `b`, `c`), fit correlation (`r`), and `period`.

Examples

```
set.seed(1)
nGenes <- 20; nSamples <- 24
trainTime <- runif(nSamples, 0, 24)
theta <- trainTime / 24 * 2 * pi
trainMat <- t(sapply(seq_len(nGenes), function(i) {
  5 + 2 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(nSamples, sd = 0.3)
}))
rownames(trainMat) <- paste0("gene", seq_len(nGenes))
fit <- fitMolecularTimetable(trainMat, trainTime, nGenes = 10)
length(fit$genes)
```

`fitTauFisher` *Fit a tauFisher model from labelled training data*

Description

Calls the authors' own pipeline (`tauFisher::run_metacycle()` + `find_periodic_genes()` + `train_tauFisher()`) rather than reimplementing it. `run_metacycle()` writes intermediate files; these are written to and cleaned up from a temporary directory, never the working directory.

Usage

```
fitTauFisher(
  trainMat,
  trainTime,
  method = c("JTK", "LS"),
  thres = 1.01,
  nrep = 1,
  numbasis = 5,
  seed = NULL
)
```

Arguments

`trainMat`	Numeric matrix of expression, genes x samples (linear or log-scale; `train_tauFisher()` log2-transforms internally).
`trainTime`	Numeric vector of known circadian phase in `[0, 24)`, one value per column of `trainMat`.
`method`	Character vector of MetaCycle method(s) to detect rhythmic genes, passed to `run_metacycle()`/`find_periodic_genes()`. Default `c("JTK", "LS")`, the package's own default.
`thres`	p-value threshold passed to `find_periodic_genes()`. Default 1.01 (i.e. no filtering), matching the authors' own `SingleDataset` vignette, which ranks candidate genes by p-value and takes the top 10 rather than thresholding.
`nrep`	Number of training replicates, passed to `train_tauFisher()`. Default 1, appropriate whenever there is exactly one array per (tissue, timepoint) cell (as for the GSE54650 benchmark backbone); set higher only if `trainMat` truly contains stacked biological replicates as separate columns per timepoint.
`numbasis`	Number of Fourier basis functions for the FDA smoothing step, passed to `train_tauFisher()`. Default 5, the package's own default.
`seed`	Optional integer seed. `train_tauFisher()` fits a `nnet::multinom` model with random initial weights; if `seed` is supplied, the global RNG state is saved and restored (AGENTS.md Section 10), so this call does not affect the caller's subsequent random draws.

Value

An object of class "tauFisherFit": a list with the chosen periodic genes (Genes), the PCA embedding (PCA_embedding) and trained multinomial model (Model) returned by train_tauFisher(), which MetaCycle method supplied the genes (method), and period (always 24).

Examples

```
if (requireNamespace("tauFisher", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    100 + 40 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  fit <- fitTauFisher(trainMat, time, seed = 1)
  length(fit$Genes)
}
```

---

fitTimeSignatR *Fit a TimeSignatR model from labelled training data*

Description

Calls the authors' own TimeSignatR::trainTimeStamp() rather than reimplementing it. Unlike fitMolecularTimetable()/fitTauFisher(), TimeSignatR does no gene selection of its own - the fitted cv.glmnet object's coefficient matrix spans every gene in trainMat, with the elastic-net penalty (not a pre-filtering step) doing the shrinkage - so predictTimeSignatR() requires every one of those genes to be present in testMat, same all-or-nothing rule as predictTauFisher().

Usage

```
fitTimeSignatR(
  trainMat,
  trainTime,
  subjIDs = NULL,
  period = 24,
  alpha = 0.5,
  recalib = FALSE,
  seed = NULL,
  ...
)
```

Arguments

trainMat	Numeric matrix of expression, genes x samples.
trainTime	Numeric vector of known circadian phase in [0, period), one value per column of trainMat.

subjIDs	Character/factor vector of subject/donor IDs, one per column of trainMat, passed to trainTimeStamp() (used for its recalib option and its internal subject sampling, which is a no-op here since trainFrac = 1 selects every subject regardless). Default: colnames(trainMat), i.e. each sample treated as its own subject, if trainMat has column names; otherwise seq_len(ncol(trainMat)).
period	Length of the cycle (default 24, for hours). trainTimeStamp() itself is hard-coded to a 24-hour period; this wrapper rescales trainTime to hours before calling it (see file header).
alpha	Elastic-net mixing parameter, passed to trainTimeStamp()'s a argument (1 = lasso, 0 = ridge). Default 0.5, the package's own default.
recalib	Logical, passed to trainTimeStamp()'s recalib argument: whether to per-subject mean-centre expression (via TimeSignatR::recalibrateExprs()) before fitting. Default FALSE, the package's own default; only meaningful with repeated-measures subjIDs.
seed	Optional integer seed. trainTimeStamp() draws its (here, no-op) subject sample and cv.glmnet()'s cross-validation fold assignment from the global RNG; if seed is supplied, the global RNG state is saved and restored (AGENTS.md Section 10), so this call does not affect the caller's subsequent random draws.
...	Additional arguments passed to trainTimeStamp() (and on to glmnet::cv.glmnet()).

Value

An object of class "timeSignatRFit": a list with the training gene names (genes), the raw trainTimeStamp() result (timestamp, containing the fitted cv.glmnet object predTimeStamp() needs), and period.

Examples

```
if (requireNamespace("TimeSignatR", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    5 + 2 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 0.3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  colnames(trainMat) <- paste0("s", seq_len(length(time)))
  fit <- fitTimeSignatR(trainMat, time, seed = 1)
  length(fit$genes)
}
```

---

fitZeitZeiger *Fit a ZeitZeiger model from labelled training data*

Description

Calls the authors' own pipeline (`zeitzeiger::zeitzeigerFit()` + `zeitzeiger::zeitzeigerSpc()`) rather than reimplementing it. Unlike `fitTauFisher()`, zeitzeiger has no separate "trained model" object - the fitted periodic splines are re-derived from `xTrain`/`timeTrain` at prediction time (`zeitzeiger::zeitzeigerPred` own design) - so the training matrix and times are carried in the returned object as well as the SPC result.

Usage

```
fitZeitZeiger(
  trainMat,
  trainTime,
  period = 24,
  nKnots = 3,
  nTime = 10,
  sumabsv = 1,
  orth = TRUE,
  useSpc = TRUE
)
```

Arguments

`trainMat`	Numeric matrix of expression, genes x samples.
`trainTime`	Numeric vector of known circadian phase in `[0, period)`, one value per column of `trainMat`.
`period`	Length of the cycle (default 24, for hours).
`nKnots`	Number of internal knots for the periodic smoothing spline, passed to `zeitzeigerFit()`. Default 3, the package's own default.
`nTime`	Number of time-points used to discretise each gene's fitted curve before computing sparse principal components, passed to `zeitzeigerSpc()`. Default 10, the package's own default.
`sumabsv`	L1-constraint on the sparse principal components, passed to `zeitzeigerSpc()`. Default 1, the package's own default.
`orth`	Logical; require the components be orthogonal, passed to `zeitzeigerSpc()`. Default `TRUE`, the package's own default.
`useSpc`	Logical; use `PMA::SPC()` for the sparse components (default) or fall back to `base::svd()`, passed to `zeitzeigerSpc()`.

Value

An object of class `"zeitZeigerFit"`: a list with the training gene names (`genes`), the transposed training matrix (`xTrain`) and fractional training time (`timeTrain`) that `zeitzeigerPredict()` needs at prediction time, the SPC result (`spcResult`), `nKnots`, and `period`.

Examples

```
if (requireNamespace("zeitzeiger", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    5 + 2 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 0.3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  fit <- fitZeitZeiger(trainMat, time)
  length(fit$genes)
}
```

`mapOrthologs` *Map gene symbols to their one-to-one ortholog in another species*

Description

Wraps `babelgene::orthologs()` (AGENTS.md Section 4 Step 1.2), restricted to unambiguous one-to-one orthologs: a gene with zero or more than one candidate ortholog is dropped (`ortholog = NA`, `status` records why), never guessed at via a many-to-many mapping.

Usage

```
mapOrthologs(genes, fromSpecies, toSpecies)
```

Arguments

`genes`	Character vector of gene symbols in `fromSpecies`'s convention (`Mmusculus`: `Arntl`; `Hsapiens`: `ARNTL` - the same convention `standardizeGeneIds()` and `estimatePhase()` use).
`fromSpecies, toSpecies`	`"Mmusculus"` or `"Hsapiens"`, must differ.

Details

`babelgene`'s bundled symbol table is a HomoloGene-vintage snapshot and can lag current HGNC symbols - verified directly for a gene this package cares about: the human ortholog of mouse `Arntl` is stored there under the legacy symbol `"BMAL1"`, not the current official `"ARNTL"`, so querying `"ARNTL"` on its own returns zero rows. To avoid that silently dropping known circadian genes, every query also tries the gene's known aliases from `circaTime`'s own alias table (the same one `standardizeGeneIds()` uses) in a single batched lookup, and results are relabelled back to the originally requested symbol.

Value

A `data.frame`, one row per unique input gene (input order preserved): `input_gene`, `ortholog` (`NA` if none), `support_n` (number of supporting ortholog databases, `NA` if none), `status` (`"ok"`, `"not_found"`, or `"ambiguous"`). A `warning()` reports how many genes were dropped and why.

Examples

```
mapOrthologs(c("Arntl", "Clock", "Per1"), fromSpecies = "Mmusculus", toSpecies = "Hsapiens")
```

`phaseConfounding` *Is circadian phase confounding a differential expression design?*

Description

The diagnostic for users who are not circadian researchers (AGENTS.md Section 6): given a design such as `~ condition`, is the samples' circadian phase distributed unevenly across that design's terms? If it is, rhythmic genes will show apparent condition effects that are really collection-time effects.

Usage

```
phaseConfounding(
  se,
  design,
  phase_col = "phase",
  flag_col = "phase_flag",
  method = NULL,
  tissue = NULL,
  species = "Mmusculus",
  nPerm = 1000,
  period = 24,
  seed = NULL
)
```

Arguments

`se`	A `SummarizedExperiment`.
`design`	A one-sided formula naming `colData(se)` columns, e.g. `~ condition` or `~ condition + age`. Each term is tested separately. Interaction terms (`a:b`) are skipped with a warning - a term-by-term imbalance report is what is actionable here, and a full multivariable circular regression is out of scope.
`phase_col`	Name of the `colData(se)` column holding phase. If absent, `estimatePhase()` is called first, which requires `method` and `tissue`.
`flag_col`	Name of the `colData(se)` column holding `estimatePhase()`'s quality flag. When present, only samples flagged `"ok"` are tested - `"low_amplitude"`/`"gene_coverage"`/`"failed"` phases are unreliable and would only add noise. Set to `NULL` to use every non-`NA` phase regardless.
`method, tissue, species`	Passed to `estimatePhase()`, and used only when `se` does not already carry `phase_col`.

nPerm	Number of permutations (default 1000).
period	Length of the cycle (default 24).
seed	Optional integer seed; the global RNG state is left unchanged either way (AGENTS.md Section 10).

Details

Phase is only a confounder when it is *unbalanced* across a term's levels. Phase that varies randomly within every group inflates residual variance but does not bias the term's coefficient, so a non-significant result here is genuinely reassuring about bias (though not about power).

Value

A `data.frame` with one row per tested term: `term`, `type` (`"categorical"`/`"numeric"`), `n` (samples used), `n_groups` (`NA` for numeric terms), `statistic` (Watson-Williams F, or circular-linear correlation), `effect_size` (for categorical terms the largest pairwise circular distance between group mean phases, in the same units as `period`, i.e. directly readable as hours; for numeric terms the circular-linear correlation itself), and `p_value`.

Two attributes carry the supporting detail: `"groupMeans"`, a `data.frame` (`term`, `group`, `n`, `circular_mean`) of per-group mean phases for the categorical terms, and `"nDropped"`, the number of samples excluded for having an `NA` or non-`"ok"` phase.

What is tested, and what is not

This tests the association between phase and each design term. It does **not** test which genes are rhythmic - rhythmicity testing on time-labelled data is a solved problem and an explicit non-goal (AGENTS.md Section 3); hand the phased object to `limorhyde/dryR` for that. It also does not adjust or correct the design; it reports the imbalance so the analyst can decide whether to add phase as a covariate, which is a modelling decision, not something to apply silently.

Statistics

Both tests are circular (AGENTS.md 7.1) and both take their null distribution from permuting phase against the term, which avoids the von Mises and equal-concentration assumptions the parametric versions rely on:

- **Categorical terms** (factor, character, logical): the Watson-Williams between-group statistic in its F form.
- **Numeric terms**: the Mardia/Johnson-Wehrly circular-linear correlation.

P-values use the add-one estimator `(1 + #{perm >= observed}) / (nPerm + 1)`, so the smallest reportable value is `1 / (nPerm + 1)` and a p-value is never reported as exactly zero.

Examples

```
set.seed(1)
# 20 samples: cases collected around ZT6, controls around ZT14 - a design where
# collection time tracks the condition almost perfectly.
phase <- c(rnorm(10, mean = 6, sd = 1), rnorm(10, mean = 14, sd = 1))
```

```
mat <- matrix(rnorm(100 * 20), nrow = 100)
rownames(mat) <- paste0("g", seq_len(100))
colnames(mat) <- paste0("s", seq_len(20))
se <- SummarizedExperiment::SummarizedExperiment(assays = list(logExpr = mat))
SummarizedExperiment::colData(se)$phase <- wrapPhase(phase)
SummarizedExperiment::colData(se)$condition <- rep(c("case", "control"), each = 10)
phaseConfounding(se, ~ condition, nPerm = 199, seed = 1)
```

`PhaseReferenceModel-class` *A trained circadian phase-inference reference model*

Description

Bundles everything `estimatePhase()` needs to predict phase for new samples against one (species, tissue, method, platform) reference cohort: the engine's own fit object, a pool of real held-out circular residuals for `phase_lo`/`phase_hi` (AGENTS.md Section 9 Phase 3 - the same residual-pool mechanism `bootstrapPhaseCI()` uses, but precomputed once at training time rather than live-bootstrapped per call, since refitting e.g. tauFisher dozens of times per user call would be prohibitively slow), and the core-clock PCA basis + derived threshold for `clock_amp`/`phase_flag` (AGENTS.md 7.4). Built by `inst/scripts/train_reference_models.R` from GSE54650/GSE54651 and bundled internally as `.referenceModels` (see `availableReferenceModels()` to list what is covered).

Usage

```
## S4 method for signature 'PhaseReferenceModel'
show(object)
```

Arguments

`object`	A `PhaseReferenceModel`.

Value

An S4 object of class `PhaseReferenceModel`.

Slots

`species,tissue,method` Character scalars identifying this model. Only `species = "Mmusculus"` is currently bundled - see `estimatePhase()`'s documentation for why (AGENTS.md 7.6 gap).

`platform` Character scalar: the data type this model was trained on, e.g. `"microarray"` (GSE54650) or `"rnaseq"` (GSE54651, AGENTS.md Section 5 Step 2.3). Defaults to `"microarray"` for models built before this slot existed, so old `new("PhaseReferenceModel", ...)` calls that omit it keep working.

`fit` The engine's own fit object (e.g. from `fitMolecularTimetable()` or `fitTauFisher()`), passed as-is to that engine's `predict` function. Typed `ANY` rather than `list`: both fit objects are S3-classed lists (`"molecularTimetableFit"`/`"tauFisherFit"`), and S4's `validObject()` rejects an S3-classed object against a `list`-typed slot even though `is.list()` on it is `TRUE` - the engines' own `predict` functions require that S3 class via `inherits()`, so it must survive storage in this slot unchanged.

`residualPool` Numeric vector of real held-out circular residuals (`circularDiff(pred, truth)`) pooled from tissues OTHER than this model's own (leave-tissue-out, same principle as `benchmarks/run_phase3_cali`

`ampCenter, ampScale` Named numeric vectors: the per-gene z-score center/scale from `clockAmplitude()`'s PCA fit on this cohort.

`ampRotation` Numeric matrix (genes x nPC): the PCA rotation from `clockAmplitude()` on this cohort.

`ampGenes` Character vector, the core clock genes (row order matching `ampCenter`/`ampScale`/`ampRotation`) used for this cohort's amplitude PCA.

`ampThreshold` Numeric scalar: the `clockAmplitude()` cutoff below which `phase_flag` becomes `"low_amplitude"` (from `deriveAmplitudeThreshold()`).

`cvMAE` Numeric scalar: this exact (species, tissue, method, platform) combination's OWN held-out cross-validated circular MAE, in hours, measured during training. Distinct from `mean(abs(residualPool))`, which summarises *other* tissues' residuals (the pool is leave-tissue-out) and therefore cannot say how well this engine does on this particular tissue. That distinction is the whole point of the slot: `estimatePhase()`'s `method = "consensus"` weights each engine by `1/cvMAE`, which is only tissue-aware because this number is. `NA_real_` if not measured.

`trainedOn` Character scalar, a citation/provenance string.

Examples

```
rm <- methods::new("PhaseReferenceModel",
  species = "Mmusculus", tissue = "Liv", method = "timetable",
  fit = list(genes = c("Arntl", "Per1")), residualPool = rnorm(50, sd = 1.5),
  ampCenter = c(Arntl = 0, Per1 = 0), ampScale = c(Arntl = 1, Per1 = 1),
  ampRotation = matrix(c(0.7, 0.7, 0.7, -0.7), nrow = 2),
  ampGenes = c("Arntl", "Per1"), ampThreshold = 2,
  trainedOn = "toy example, not a real fit"
)
rm
```

`plotCalibration` *Plot a calibration curve*

Description

Base-graphics plot of empirical vs. nominal coverage (AGENTS.md Phase 3 task 3: "Plot it"), with the identity line marking perfect calibration. Points below the line mean the stated interval is too narrow (overconfident); points above mean it is too wide (underconfident, but honest).

Usage

```
plotCalibration(calCurve)
```

Arguments

calCurve	Output of calibrationCurve().

Value

calCurve, invisibly. Called for its plotting side effect.

Examples

```
calCurve <- data.frame(nominal = c(0.5, 0.8, 0.95), empirical = c(0.53, 0.81, 0.96))
plotCalibration(calCurve)
```

---

predictMolecularTimetable

Predict circadian phase with a fitted Molecular Timetable model

---

Description

For each test sample, finds the phase (on a fine grid over one cycle) whose per-gene harmonic-fit template best correlates with the sample's observed expression of the fitted model's time-indicating genes.

Usage

```
predictMolecularTimetable(fit, testMat, gridN = 288, minGenes = 5)
```

Arguments

fit	A "molecularTimetableFit" object from fitMolecularTimetable().
testMat	Numeric matrix of expression, genes x samples, on the same scale as the training matrix. Only genes overlapping fit$genes are used per sample; genes absent from testMat are silently dropped from that computation.
gridN	Number of grid points spanning one cycle to search over (default 288, i.e. 5-minute resolution over 24h).
minGenes	Minimum number of overlapping time-indicating genes required to attempt a prediction for a sample; below this, the sample gets NA (default 5).

Value

A data.frame with one row per column of testMat: phase (numeric, [0, period), or NA if prediction failed), and confidence (the maximum per-sample correlation achieved across the grid, or NA).

Examples

```
set.seed(1)
nGenes <- 20; nSamples <- 24
trainTime <- runif(nSamples, 0, 24)
theta <- trainTime / 24 * 2 * pi
truePhase <- runif(nGenes, 0, 2 * pi)
trainMat <- t(sapply(seq_len(nGenes), function(i) {
  5 + 2 * cos(theta - truePhase[i]) + rnorm(nSamples, sd = 0.1)
}))
rownames(trainMat) <- paste0("gene", seq_len(nGenes))
fit <- fitMolecularTimetable(trainMat, trainTime, nGenes = 10, minCor = 0.3)
pred <- predictMolecularTimetable(fit, trainMat)
head(pred)
```

---

predictTauFisher *Predict circadian phase with a fitted tauFisher model*

Description

Calls the authors' own `tauFisher::test_tauFisher()`. Unlike `train_tauFisher()`, this step is deterministic given the fitted model (no RNG use).

Usage

```
predictTauFisher(fit, testMat, minGenes = 2)
```

Arguments

`fit`	A `"tauFisherFit"` object from `fitTauFisher()`.
`testMat`	Numeric matrix of expression, genes x samples, same scale as the training matrix. Prediction requires every one of `fit$Genes` to be present in `testMat`'s rownames (tauFisher's gene-pair-difference features need the full set); samples are only ever predicted as a whole model, so if any fitted gene is missing, every sample gets `NA` rather than a prediction computed on a silently shrunken gene set.
`minGenes`	Minimum number of `fit$Genes` that must be present in `testMat` to attempt prediction at all (default 2, the minimum tauFisher needs to form even one gene pair).

Value

A `data.frame` with one row per column of `testMat`: `sample`, `phase` (integer hour in `[0, 24)`, from `nnet::multinom`'s discrete class prediction, or `NA` if prediction was not attempted), and `confidence` (`NA_real_` for every row - tauFisher has no native per-sample confidence signal; calibrated uncertainty is Phase 3 scope, AGENTS.md Section 9).

Examples

```
if (requireNamespace("tauFisher", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    100 + 40 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  fit <- fitTauFisher(trainMat, time, seed = 1)
  pred <- predictTauFisher(fit, trainMat)
  head(pred)
}
```

---

predictTimeSignatR *Predict circadian phase with a fitted TimeSignatR model*

Description

Calls the authors' own `TimeSignatR::predTimeStamp()`. `cv.glmnet`'s prediction matches features by column position, not by name, so - like `predictZeitZeiger()`

- this requires every one of `fit$genes` to be present in `testMat`'s rownames; if any is missing, every sample gets `NA` rather than a prediction computed on a reordered or shrunken gene set.

Usage

```
predictTimeSignatR(fit, testMat)
```

Arguments

`fit`	A `"timeSignatRFit"` object from `fitTimeSignatR()`.
`testMat`	Numeric matrix of expression, genes x samples, same scale as the training matrix.

Value

A `data.frame` with one row per column of `testMat`: `sample`, `phase` (numeric, `[0, period)`, or `NA` if `testMat` was missing any fitted gene), and `confidence` (`NA_real_` for every row - `predTimeStamp()` returns a point estimate with no native per-sample confidence signal; calibrated uncertainty is Phase 3 scope, AGENTS.md Section 9).

Examples

```
if (requireNamespace("TimeSignatR", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
```

```
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    5 + 2 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 0.3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  colnames(trainMat) <- paste0("s", seq_len(length(time)))
  fit <- fitTimeSignatR(trainMat, time, seed = 1)
  pred <- predictTimeSignatR(fit, trainMat)
  head(pred)
}
```

`predictZeitZeiger` *Predict circadian phase with a fitted ZeitZeiger model*

Description

Calls the authors' own `zeitzeiger::zeitzeigerPredict()`. zeitzeiger matches training and test features by column position, not by name, so - unlike `predictMolecularTimetable()`, which silently uses whatever fitted genes overlap `testMat` - this requires every one of `fit$genes` to be present in `testMat`'s rownames; if any is missing, every sample gets `NA` rather than a prediction computed on a reordered or shrunken gene set (same all-or-nothing rule as `predictTauFisher()`).

Usage

```
predictZeitZeiger(fit, testMat, nSpc = 2, timeRange = seq(0, 1 - 0.01, 0.01))
```

Arguments

`fit`	A `"zeitZeigerFit"` object from `fitZeitZeiger()`.
`testMat`	Numeric matrix of expression, genes x samples, same scale as the training matrix.
`nSpc`	Number of sparse principal components to use for prediction, passed to `zeitzeigerPredict()`'s `nSpc` argument as a single value (a single prediction per sample, not zeitzeiger's default sweep over `1:K`). Default 2. Clamped down to the number of components `zeitzeigerSpc()` actually returned (`length(fit$spcResult$d)`) if `nSpc` exceeds it, e.g. for small gene sets.
`timeRange`	Vector of fractional-time grid points searched for the maximum-likelihood initial value, passed through to `zeitzeigerPredict()`. Default matches the package's own default resolution (100 points over one cycle).

Value

A `data.frame` with one row per column of `testMat`: `sample`, `phase` (numeric, `[0, period)`, or `NA` if `testMat` was missing any fitted gene), and `confidence` (`NA_real_` for every row - zeitzeiger's per-sample likelihood surface is not converted to a calibrated confidence score here; calibrated uncertainty is Phase 3 scope, AGENTS.md Section 9).

Examples

```
if (requireNamespace("zeitzeiger", quietly = TRUE)) {
  set.seed(1)
  nGenes <- 40; time <- seq(0, 23, by = 1)
  theta <- time / 24 * 2 * pi
  trainMat <- t(sapply(seq_len(nGenes), function(i) {
    5 + 2 * cos(theta - runif(1, 0, 2 * pi)) + rnorm(length(time), sd = 0.3)
  }))
  rownames(trainMat) <- paste0("gene", seq_len(nGenes))
  fit <- fitZeitZeiger(trainMat, time)
  pred <- predictZeitZeiger(fit, trainMat)
  head(pred)
}
```

---

standardizeGeneIds *Standardize gene identifiers in a SummarizedExperiment to gene symbol*

Description

Auto-detects the identifier type of `rownames(se)` per gene (Ensembl, Entrez, RefSeq, or already gene symbol - AGENTS.md Section 4 Step 1.1) and maps everything to gene symbol, the internal identifier convention every other circaTime function uses (`estimatePhase()`, `phaseConfounding()`, `clockAmplitude()`). Runs in two layers: a dependency-free pass first (Ensembl version-suffix stripping and the hand-curated circadian clock-gene alias table also used internally by `estimatePhase()`), then, only for whatever remains unresolved, `AnnotationDbi::mapIds()` against `org.Mm.eg.db`/`org.Hs.eg.db` if installed (`Suggests`, not required for the common already-symbol case - see `notes/decisions.md` Section 8).

Usage

```
standardizeGeneIds(se, species = "Mmusculus", map_aliases = TRUE)
```

Arguments

`se`	A `SummarizedExperiment` with gene identifiers as `rownames(se)`.
`species`	`"Mmusculus"` or `"Hsapiens"`.
`map_aliases`	Logical, default `TRUE`: resolve circadian clock-gene aliases (e.g. `BMAL1` -> `Arntl`/`ARNTL`) in the free layer before falling through to `AnnotationDbi`.

Details

This does not collapse rows that map to the same symbol (e.g. two RefSeq transcripts of one gene) - `rownames(se)` may contain duplicates afterward, same as an unstandardized `SummarizedExperiment` can already have. Functions that need a unique gene set (e.g. `estimatePhase()`'s internal `.prepareExpressionForModel()` already collapse duplicates by row mean at that point.

Value

se with rownames(se) replaced by standardized gene symbols. metadata(se)$circaTime_id_mapping is set to a one-row-per-input-gene data.frame: input_id, id_type ("symbol"/"ensembl"/"entrez"/"refseq"), mapped_symbol, resolved_by ("unchanged", "free_layer" – Ensembl version-stripping, the hardcoded core-clock-gene Ensembl table, or the alias table –, "annotationdbi", or "unresolved"). A summary is also emitted via message(): how many genes were resolved, and by which layer.

Examples

```
mat <- matrix(rnorm(9), nrow = 3, dimnames = list(c("Arntl", "Bmal1", "Clock"), paste0("s", 1:3)))
se <- SummarizedExperiment::SummarizedExperiment(assays = list(logExpr = mat))
se <- standardizeGeneIds(se, species = "Mmusculus")
rownames(se)
```

---

transferPhase *Cross-dataset transfer benchmark of phase-inference engines*

Description

Fits each engine on one SummarizedExperiment and scores it on a *different* one (AGENTS.md Section 5 Step 2.2), e.g. training on microarray data and predicting on RNA-seq data from the same tissue/study. This is a distinct question from benchmarkPhase()'s cross-validation (how well an engine predicts held-out samples from the *same* dataset it was trained on) and benchmarkPhase() has no way to ask it - there is exactly one train/test split here, fixed by which dataset a sample belongs to, not nFolds random ones. This function is not a parallel benchmarking harness, though: it reuses the same engine registry, gene-name-based fit/predict matching (each engine's own predict function already intersects fit$genes against rownames(testMat), so trainSe/testSe do not need to share an identical gene panel), and circular-error metrics that benchmarkPhase() uses internally.

Usage

```
transferPhase(
  trainSe,
  testSe,
  truth_col,
  methods = "timetable",
  train_assay_name = NULL,
  test_assay_name = NULL,
  period = 24,
  engineArgs = list()
)
```

Arguments

trainSe, testSe	SummarizedExperiment objects to fit on and score against, respectively (e.g. a microarray cohort and an RNA-seq cohort of the same tissue). Their gene panels need not match exactly.

truth_col — Name of the colData column holding ground-truth phase, in [0, period). Must exist in both trainSe and testSe.

methods — Character vector of engine names; see benchmarkPhase().

train_assay_name, test_assay_name — Assay to use in trainSe/testSe respectively; each defaults to that object's first assay.

period — Length of the cycle (default 24). Passed through to every engine whose fit accepts a period argument (Molecular Timetable, ZeitZeiger, TimeSignatR), which stores it on the fit object its predict reads back. tauFisher has no period argument and is inherently 24-hour: requesting period != 24 with "taufisher" emits a warning, and its predictions stay on [0, 24).

engineArgs — Named list, keyed by method, of extra arguments to pass to that engine's fit and predict functions; see benchmarkPhase().

Value

A list with two elements: summary, a data.frame with one row per method (method, circular_mae, circular_cor, n, n_failed); and residuals, a data.frame with one row per (method, testSe sample) (method, sample, truth, pred, circular_error, aligned, failed).

If an engine's fit/predict call actually *errors*, that method's predictions are all NA and n_failed records how many test samples were lost, exactly as benchmarkPhase() does for a failed fold. Separately, three of the four wrapped engines (predictZeitZeiger(), predictTimeSignatR(), and predictTauFisher()) require the *entire* fitted gene set to be present in testMat and silently return all-NA phase (no error at all) when it is not - only predictMolecularTimetable() tolerates partial gene-panel overlap (see their own docs). That case shows up here as n = 0 with n_failed = 0 (not counted as a "failure" - nothing errored), not n_failed = n; check n as well as n_failed when auditing a transfer result, and see notes/decisions.md for a worked example against real cross-platform data. This never fabricates a result for a failed sample (AGENTS.md Section 0.3).

Examples

```
set.seed(1)
makeSe <- function(n) {
  time <- seq(0, 23, length.out = n)
  theta <- time / 24 * 2 * pi
  mat <- rbind(
    g1 = 5 + 2 * cos(theta) + rnorm(n, sd = 0.2),
    g2 = 5 + 2 * sin(theta) + rnorm(n, sd = 0.2)
  )
  colnames(mat) <- paste0("s", seq_len(n))
  se <- SummarizedExperiment::SummarizedExperiment(assays = list(logExpr = mat))
  SummarizedExperiment::colData(se)$ZT <- time
  se
}
tb <- transferPhase(makeSe(12), makeSe(8), truth_col = "ZT", methods = "timetable")
tb$summary
```

`wrapPhase` *Wrap a phase value onto* `[0, period)`

Description

Wrap a phase value onto `[0, period)`

Usage

```
wrapPhase(x, period = 24)
```

Arguments

`x`	Numeric vector of phase values, in the same units as `period`.
`period`	Length of the cycle (default 24, for hours).

Value

Numeric vector, each element wrapped into `[0, period)`.

Examples

```
wrapPhase(c(-1, 0, 23.5, 24, 25.5))
```

Index

.assignDonorFolds(), *6*

availableReferenceModels, 3
availableReferenceModels(), *23, 24*, *35*

bayesianPhasePosterior, 4
bayesianPhasePosterior(), *11*, *25, 26*
benchmarkPhase, 5
benchmarkPhase(), *3*, *8*, *42, 43*
bootstrapPhaseCI, 7
bootstrapPhaseCI(), *4*, *9*, *16*, *35*

calibrationCurve, 9
calibrationCurve(), *8*, *37*
circularAlign, 10
circularConcentration, 11
circularConcentration(), *4*, *13*
circularCor, 11
circularCor(), *6*
circularCredibleInterval, 12
circularDiff, 13
circularDiff(), *8*, *11*, *16*
circularError, 14
circularError(), *22*
circularIntervalCovers, 14
circularMAE, 15
circularMAE(), *12*, *14*
circularMean, 16
circularMean(), *17*
circularQuantileInterval, 16
circularQuantileInterval(), *4, 5*, *7*, *9*, *12–14*
circularWeightedMean, 17
circularWeightedMean(), *11*, *26*
clockAmplitude, 18
clockAmplitude(), *19–25*, *36*
clockCoherence, 19
clockCoherence(), *24*
coreClockGenes, 21
coreClockGenes(), *18*, *25*

deriveAmplitudeThreshold, 22
deriveAmplitudeThreshold(), *21*, *24*, *36*

estimatePhase, 23
estimatePhase(), *3*, *17*, *19–21*, *32, 33*, *35, 36*, *41*

fitMolecularTimetable, 27
fitMolecularTimetable(), *29*, *36, 37*
fitTauFisher, 28
fitTauFisher(), *29*, *31*, *36*, *38*
fitTimeSignatR, 29
fitTimeSignatR(), *3*, *39*
fitZeitZeiger, 30
fitZeitZeiger(), *3*, *40*

mapOrthologs, 32
mapOrthologs(), *25*

phaseConfounding, 33
PhaseReferenceModel, *23*, *26*
PhaseReferenceModel-class, 35
plotCalibration, 36
prcomp(), *19*
predictMolecularTimetable, 37
predictMolecularTimetable(), *40*, *43*
predictTauFisher, 38
predictTauFisher(), *29*, *40*, *43*
predictTimeSignatR, 39
predictTimeSignatR(), *3*, *29*, *43*
predictZeitZeiger, 40
predictZeitZeiger(), *3*, *39*, *43*

scale(), *19*
show,PhaseReferenceModel-method *(PhaseReferenceModel-class)*, 35
standardizeGeneIds, 41
standardizeGeneIds(), *25*, *32*

transferPhase, 42
transferPhase(), *3*

wrapPhase, 44