Package 'sptrends'

September 22, 2026

Title Statistical Inference for Spatiotemporal Trends in Gridded Data

Version 1.6.3

Language en-GB

Description Provides a unified and reproducible framework for statistical inference of spatiotemporal trends in gridded environmental data. The framework addresses the interconnected challenges of serial correlation, spatial dependence and multiple testing that commonly arise when analysing gridded environmental time series. Its core methods support serial-correlation treatment through trend-preserving prewhitening, pixel-wise and spatially explicit trend inference, slope estimation and multiple-testing correction. These methods may be applied independently or integrated within configurable analytical workflows. Dedicated workflows are also provided to reproduce methodologies published in the scientific literature: Gutiérrez-Hernández and García (2025) <doi:10.1016/j.rsase.2024.101377> for the True Significant Trends workflow, Gutiérrez-Hernández and García (2024) <doi:10.3390/rs16203886> for the Robust Trend Analysis workflow, and Gutiérrez-Hernández and García (2025) <doi:10.3390/math13223630> for the adaptive false discovery rate procedure. Supporting utilities facilitate raster data import and inspection, anomaly calculation, spatial autocorrelation diagnostics, simulation studies, benchmarking, visualisation, mapping, and reporting.

License GPL (>= 3)

URL https://github.com/Olive-r/sptrends, https://olive-r.github.io/sptrends/

BugReports https://github.com/Olive-r/sptrends/issues

Depends R (>= 4.1)

Imports terra (>= 1.7-0), Matrix, parallel, stats, utils, graphics, grDevices, withr (>= 2.2.0)

Suggests fields, testthat (>= 3.2.0), knitr, rmarkdown, ncdf4, Kendall, modifiedmk, rkt, robslopes, trend, zyp

VignetteBuilder knitr

Encoding UTF-8

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Oliver Gutiérrez-Hernández [aut, cre] (ORCID: <https://orcid.org/0000-0003-2580-5465>, affiliation: Department of Geography, University of Málaga, Málaga, Spain), Luis V. García [aut] (ORCID: <https://orcid.org/0000-0002-5514-2941>, affiliation: Institute of Natural Resources and Agrobiology of Seville (IRNAS), Spanish National Research Council (CSIC), Seville, Spain)

Maintainer Oliver Gutiérrez-Hernández <olivergh@uma.es>

Repository CRAN

Date/Publication 2026-09-22 07:00:09 UTC

Contents

benchmark_methods *Benchmark statistical methods across known-truth simulation scenarios*

Description

Coordinates reproducible Monte Carlo experiments without tying the benchmark to sptrends implementations. Each method is an ordinary function, so methods from other packages can be evaluated under exactly the same simulated realisations and truth fields.

Usage

```
benchmark_methods(
  scenarios,
  methods,
  n_replicates = 100L,
 stage = c("trend_test", "prewhitening", "slope", "fdr", "fwer", "custom", "detection"),
  simulator = sim_trend_stack,
  prepare = NULL,
  evaluator = NULL,
  seed = 1L,
  metrics = c("type_i", "type_ii", "type_iii", "field_power", "global_power",
    "within_image_power", "directional_power", "fdr"),
  evaluation_mask = NULL,
  verbose = TRUE
)
```

Arguments

scenarios	Named list of argument lists passed to simulator.
methods	Named list of functions. Each receives (input, simulation) and returns a transformed raster series for stage = "prewhitening", a detection object for trend/FDR/FWER stages, a slope raster/vector for stage = "slope", or an object accepted by evaluator.
n_replicates	Positive integer number of realisations per scenario.
stage	One of "prewhitening", "trend_test", "slope", "fdr", "fwer", or "custom". "detection" is retained as an alias for "trend_test".
simulator	Function used to generate one known-truth realisation. It must accept the scenario arguments plus seed, and return at least series, true_signal, true_direction, and true_slope components when the corresponding built-in scorer is used.
prepare	Optional function called once per replicate as prepare(series, simulation). Its result becomes the identical input passed to every method. Use it to compute one common p-value or statistic map before comparing FDR or FWER methods.
evaluator	Optional scoring function called as evaluator(outputs, simulation). Required for stage = "custom".

seed	Integer seed that deterministically generates replicate seeds.
metrics	Metrics passed to compare_detections() for detection benchmarks.
evaluation_mask	Optional common mask or a function of simulation returning one. The same mask is applied to every method so all methods are compared on exactly the same cells.
verbose	Logical. If TRUE, report method-level progress, elapsed duration and estimated time remaining across the complete Monte Carlo experiment. Simulator-level verbosity is disabled automatically when the simulator exposes a verbose argument and the scenario does not override it.

Details

Function type: Benchmarking function – coordinates simulation, method execution, scoring and timing; it is not an inferential workflow.

Value

A data frame with one row per scenario, replicate and method, including explicit scenario factors, known-truth composition, elapsed time and stage-appropriate accuracy metrics. It has classes "sptrends_benchmark" and "sptrends", providing unified print(), summary(), and scenario-performance plot() methods.

Typical use

Define named scenarios, define package or external method wrappers, and run the same methods on every generated realisation. Built-in scoring is available for prewhitening, trend tests, slopes, FDR and FWER. A custom evaluator supports arbitrary result structures without changing the Monte Carlo engine.

Methodological details

Paired method comparison

A replicate seed is generated once and shared by every method within that replicate. Consequently, method differences are paired within identical data, known truth, prepared input and evaluation domain rather than confounded with different random fields or cell subsets. The returned table retains replicate-level results; uncertainty summaries must be calculated across those independent replicates, not across raster cells.

Statistical assumptions

Performance estimates are conditional on the simulated scenarios and the common evaluation domain. Monte Carlo replicates, not raster cells, are the independent units used to estimate repeated-sampling behaviour.

Computational considerations

Runtime grows with scenarios, replicates and methods. Scenario results are retained at replicate level so expensive experiments can be summarised or plotted without rerunning the methods.

Limitations

Built-in scorers require the documented truth components. Method-specific objects or cluster-level targets require a custom `evaluator`; incompatible method-specific evaluation domains are deliberately rejected.

Quality assurance

Tests cover every supported stage, paired inputs, external simulators, common masks, reproducible seeds, timing, failures, summaries and plots. In an independent full run, 500 replicates were evaluated in each of eight spatiotemporal scenarios. Cell-level MK decisions and directions agreed exactly between sptrends and `Kendall::MannKendall()` for every retained performance metric, while paired seeds, summaries and graphics passed all recorded controls. This validates the benchmark orchestration and scoring; it does not imply that every method compared by future users is equivalent.

See Also

`simulation_design()`, `sim_trend_stack()`, `compare_detections()`, `benchmark_summary()`

Other validation functions: `benchmark_summary()`, `compare_detections()`, `plot_detection_comparison()`, `simulation_design()`

Examples

```
methods <- list(
  MK = function(series, simulation) {
    fit <- trend_test(series, method = "MK", report = FALSE,
                      verbose = FALSE)
    list(significant = fit$stats$p <= 0.05,
         direction = fit$stats$S)
  }
)
scenarios <- list(null = list(nrow = 6, ncol = 6, n_time = 8,
                              trend_fraction = 0))
result <- benchmark_methods(scenarios, methods, n_replicates = 2,
                            seed = 1, verbose = FALSE)
result
```

---

benchmark_summary *Summarise a method benchmark across Monte Carlo replicates*

Description

Aggregates the replicate-level output of `benchmark_methods()` while preserving scenarios and methods. For detection-like stages, empirical FDR is the mean false-discovery proportion and empirical FWER is the proportion of replicates containing at least one false positive.

Usage

```
benchmark_summary(x, path = NULL, verbose = TRUE)
```

Arguments

x	Replicate-level result returned by benchmark_methods().
path	Character or NULL. If supplied, the summary is written as a CSV file at this path.
verbose	Logical. If TRUE, reports progress, elapsed time and the estimated time remaining while scenario-method groups are summarised.

Details

Function type: Benchmarking function – summarises known-truth experiments; it does not perform statistical inference on user data.

Value

A data frame with one row per scenario and method, retained scenario factors, the number of replicates, means and standard deviations of numerical metrics, and EmpiricalFDR/EmpiricalFWER when detection metrics are present.

Typical use

Run benchmark_methods() and pass its result directly to this function.

Methodological details

Monte Carlo aggregation

Replicates, rather than raster cells, are the independent Monte Carlo units. Therefore FDR and FWER are aggregated across replicate-level false discovery proportions and false-positive indicators, respectively.

Computational considerations

Aggregation operates on the retained benchmark table and does not rerun simulations or methods.

Limitations

Summary precision depends on the number of independent replicates and the range of scenarios evaluated. It does not generalise beyond those designs.

Quality assurance

Tests verify grouping, scenario retention, means, standard deviations, empirical FDR and FWER, CSV output and invalid input handling. The retained external validation also verifies the summaries against 4,000 paired MK replicates and 33 independent simulation-cycle controls.

See Also

benchmark_methods(), compare_detections()

Other validation functions: benchmark_methods(), compare_detections(), plot_detection_comparison(), simulation_design()

Examples

```
x <- structure(
  data.frame(Scenario = rep("null", 2), Replicate = 1:2,
             Method = rep("method", 2), FP = c(1, 0),
             FalseDiscoveryProportion = c(1, 0),
             AnyFalsePositive = c(1, 0)),
  class = c("sptrends_benchmark", "data.frame"))
benchmark_summary(x, verbose = FALSE)
```

`compare_detections` *Compare detection methods against a known ground truth*

Description

Evaluates how well one or more trend-detection methods recover a known truth – the scientific question this function exists to answer is simple: *which method actually recovers the true trends better?* A confusion-matrix comparison is how it answers that question, not the point of the function itself. Built for `sim_trend_stack()`'s `true_slope`, but deliberately agnostic to where either side comes from. `detections` and `ground_truth` are just logical vectors (or objects that reduce to one): this works equally well comparing `workflow_tst()` against a classic Mann-Kendall, a linear model, a method from another package, or an algorithm you implemented yourself – as long as each is reduced to a "significant / not significant" call per cell.

Usage

```
compare_detections(
  detections,
  ground_truth,
 metrics = c("sensitivity", "specificity", "precision", "accuracy", "f1", "mcc", "fpr",
   "fdr", "fwer", "type_i", "type_ii", "type_iii", "field_power", "global_power",
    "within_image_power", "directional_power"),
  replicates = FALSE,
  directions = NULL,
  truth_direction = NULL,
  evaluation_mask = NULL,
  verbose = TRUE
)
```

Arguments

`detections`	When `replicates = FALSE` (default): a named list, one element per method being compared. Each element is the method's significance call: a logical vector (`TRUE` = called significant), a single-layer `terra::SpatRaster` of 0/1 or `TRUE`/`FALSE` (coerced with `!= 0`), or a numeric/integer vector coerced the same way. The list names become the `Method` column. When `replicates = TRUE`: a list of such named lists, one per replicate (e.g. one per random seed) – every inner list must use the same method names.

ground_truth When replicates = FALSE: the ground truth, in any of the same forms as one element of detections – most commonly sim_trend_stack()'s true_slope (a SpatRaster, non-zero where a true trend exists) or its raw values. Coerced to logical the same way as detections. When replicates = TRUE: either a single ground truth reused for every replicate (the common case, when the same underlying truth is being detected repeatedly under different noise draws), or a list the same length as detections with one ground truth per replicate (for the rarer case where the truth itself is re-simulated every time too, e.g. sim_trend_stack()'s own true_slope with a different seed on each call).

metrics Character vector of derived metrics to include, from c("sensitivity", "specificity", "precision", "accuracy", "f1", "mcc", "fpr", "fdr", "fwer", "type_i", "type_ii", "type_iii", "field_power", "global_power", "within_image_power", "directional_power") (case-insensitive). The omitted default remains the original eight confusion-matrix metrics. The confusion matrix counts (TP, FP, TN, FN) are always included regardless of metrics – every requested metric is computed from that same confusion matrix, not recomputed independently. "fwer" is structurally different from the other eight – see its own entry in "Value" below – and can only be requested when replicates = TRUE; it errors otherwise, since a single run's own false-positive count is either zero or not, with no per-replicate ratio to compute in the first place. "type_i" is the background false-positive proportion; "type_ii" is the missed-signal proportion; "type_iii" is the wrong-direction proportion among detected true signals. "field_power" records whether a replicate produces at least one rejection anywhere in the evaluation domain. It can equal one because of a false positive alone. "global_power" is stricter: it records whether a replicate detects at least one true signal, whereas "within_image_power" is the proportion of signal cells detected.

replicates Logical. FALSE (default): score a single run, as described above. TRUE: score every replicate the same way, then aggregate to the mean and standard deviation of every column, per method – a single seed can favour or disfavour a method by chance, so this is what actually supports a claim like "CMK detects more true trends" (see @examples below). Generating the data and computing each method's detections for every replicate still stays as an ordinary loop you write, since that step depends entirely on which methods you are comparing and cannot be generalised without either guessing at your workflow or adding a callback-style interface – replicates = TRUE only does the scoring and aggregation, for the per-replicate detections/ground-truth you already computed.

directions Optional named list of estimated direction vectors, one per method. Alternatively, each detection may be a list containing significant and direction.

truth_direction

Optional known direction vector, normally sim_trend_stack()'s true_direction. Required for Type III error and directional power.

evaluation_mask

Optional logical vector/raster selecting one common evaluation domain for every method. Missing and FALSE cells are not scored. Method-specific masks are rejected because methods must be compared against the same cells and the same ground truth.

verbose — Logical. If TRUE, reports progress, elapsed time and the estimated time remaining while runs are scored.

Details

Function type: Validation function – scores detection methods against known truth. It is not part of the inferential pipeline.

Value

When `replicates = FALSE`: a data frame with one row per method: `Method`, `TP`, `FP`, `TN`, `FN`, and the requested metric columns among `Sensitivity` (a.k.a. recall or power: `TP / (TP + FN)`), `Specificity` (`TN / (TN + FP)`), `Precision` (`TP / (TP + FP)`), `Accuracy` (`(TP + TN) / (TP + FP + TN + FN)` – included because it is widely expected, though it can be misleading whenever true trends are rare relative to the whole raster, which is common in these simulations), `F1` (`2 * TP / (2 * TP + FP + FN)`, zero when there are errors but no true positives, and `NA` when its denominator is zero), `MCC` (Matthews correlation coefficient – a single balanced summary of all four confusion-matrix cells at once, generally preferred over `Accuracy` or `F1` under the class imbalance a low `trend_fraction` produces), and `FPR` (false positive rate, `FP / (FP + TN)`). `FDR`, in this single-run table, is the realised false discovery *proportion* (Benjamini & Hochberg, 1995) in this one result (`FP / (FP + TP)`) – not the false discovery *rate* itself, which is formally defined as the expectation of that proportion over many repetitions. `q` in `fdr_correction()` bounds that expectation, not the value in any single realisation – see `replicates = TRUE` below for what actually estimates the expectation this proportion is a single draw from. By standard convention, this realised proportion is zero when no discoveries are made; retaining those zeroes is essential when estimating FDR across replicates. Other ratios with an undefined denominator (e.g. `Precision` when a method calls nothing significant, or `MCC` when any confusion-matrix margin is zero) are returned as `NA`, not `NaN` or `0`. `FieldPower`, `GlobalPower`, and `WithinImagePower` distinguish, respectively, whether any cell was rejected, whether at least one true signal cell was detected, and the fraction of true signal cells detected.

When `replicates = TRUE`: a data frame with one row per method: `Method`, `n_replicates`, and `<column>_mean`/`<column>_sd` for every column above (`TP_mean`, `TP_sd`, `Sensitivity_mean`, `Sensitivity_sd`, and so on). `FDR_mean` here is the actual estimate of the false discovery rate itself (the average of the per-replicate proportion described above, across replicates) – the quantity `q` in `fdr_correction()` is meant to bound, unlike the single-run `FDR` column above. If `"fwer"` was requested, also a plain `FWER` column (no `_mean`/`_sd` pair): the proportion of replicates with at least one false positive (`FP > 0`) – the family-wise error rate (Benjamini & Hochberg, 1995's own point of contrast for FDR), a single already-aggregated rate rather than a per-replicate value with its own mean and spread.

Typical use

Single-run validation:

```
sim_trend_stack()
    |
run one or more detection methods on sim$series
    |
compare_detections()
    |
one table of cell-wise detection metrics
```

Replicated validation is an alternative route, not a step after the single-run call:

```
repeat simulation and detection across seeds
    |
collect detections and ground truths in lists
    |
compare_detections(replicates = TRUE)
    |
mean and standard deviation of metrics + optional empirical FWER
```

See the examples below for both routes worked through in full.

Methodological details

Why use simulated data?

Real environmental datasets have no known "correct" answer, so detection performance cannot be measured directly – there is no ground truth to score against for a real NDVI or temperature raster. Simulation studies (`sim_trend_stack()`) provide a known ground truth instead, allowing sensitivity, specificity, precision, and the other metrics below to be computed objectively, something no amount of visual inspection of a real map can substitute for.

A platform for comparing methods, not just running them

Together, `sim_trend_stack()`, this function, and the two published workflows this package offers (`workflow_tst()` and `workflow_rta()`) mean sptrends is not only a tool for running a trend analysis, but also a platform for validating and comparing *how well* different methods do it – including methods this package did not itself implement, since both `detections` and `ground_truth` are accepted in plain, method-agnostic forms (see the `detections` argument below).

Methods and metric selection

All requested metrics derive from the same cell-wise confusion matrix. Sensitivity and specificity describe conditional detection behaviour; precision and FDR describe the rejection set; MCC provides a balanced summary under class imbalance. Empirical FDR and FWER require repeated simulations and therefore are only interpretable in replicated mode.

Statistical assumptions and limitations

Detection and truth must refer to the same cells in the same order and reduce to binary calls. Results describe performance under the supplied simulated scenarios; they do not establish performance for every real environmental process. A single replicate estimates realised proportions, not repeated-sampling error rates.

Computational considerations

Scoring is vectorised over the common evaluation domain and is generally inexpensive relative to generating data or fitting the compared methods. Replicated mode retains method-level summaries rather than raster outputs.

Quality assurance

Tests compare confusion counts and every derived metric with analytically tractable cases, including undefined denominators. Additional tests cover vector/raster inputs, replicated aggregation, empirical FDR and FWER, validation failures, method ordering and S3 printing, summaries and plots. In the retained external validation, `sptrends` and `Kendall` produced identical MK decisions, directions and derived performance metrics across 4,000 paired Monte Carlo replicates. See `inst/validation/` and `?sptrends` for scope and limitations.

References

Fawcett, T. (2006) An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874. doi:10.1016/j.patrec.2005.10.010

Source of the Matthews correlation coefficient (MCC):

- Matthews, B.W. (1975) Comparison of the predicted and observed secondary structure of T4 phage lysozyme. Biochimica et Biophysica Acta (BBA) - Protein Structure, 405(2), 442-451. doi:10.1016/00052795(75)901099

Source of the false discovery rate concept FDR/FDR_mean estimate, and of the family-wise error rate FWER is contrasted against (see "Value" above for both):

- Benjamini, Y., & Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x

See Also

Other validation functions: benchmark_methods(), benchmark_summary(), plot_detection_comparison(), simulation_design()

Examples

```
# Simulated data only, deliberately -- compare_detections() needs a
# KNOWN ground truth to score against (sim$true_slope below), which
# by definition only sim_trend_stack() can provide; real environmental
# data has no equivalent "true" answer to compare a detection against.

sim <- sim_trend_stack(nrow = 15, ncol = 15, n_time = 15, seed = 1)

# Two variants of the same test: classic Mann-Kendall (no
# neighbourhood averaging) vs. the Contextual version.
trend_mk  <- trend_test(sim$series, method = "MK",
                        report = FALSE, verbose = FALSE)
trend_cmk <- trend_test(sim$series, method = "CMK",
                        report = FALSE, verbose = FALSE)

# Since sim$true_slope is known exactly (this is simulated data), we
# can score each method's raw significance calls against the truth.
compare_detections(
  detections = list(MK = trend_mk$stats$p <= 0.05,
                    CMK = trend_cmk$stats$p <= 0.05),
  ground_truth = sim$true_slope, verbose = FALSE
)

# replicates = TRUE: repeat the comparison across several random
# seeds and aggregate -- a single run can favour a method just by
# chance. Both detections and the true_slope ground truth change
# every replicate here, so ground_truth is a list too, not a single
# shared vector.
detections_list <- list()
truths_list <- list()
```

```
for (s in 1:10) {
  sim_s <- sim_trend_stack(nrow = 12, ncol = 12, n_time = 12, seed = s)
  mk_s  <- trend_test(sim_s$series, method = "MK",
                      report = FALSE, verbose = FALSE)
  cmk_s <- trend_test(sim_s$series, method = "CMK",
                      report = FALSE, verbose = FALSE)
  detections_list[[s]] <- list(MK = mk_s$stats$p <= 0.05,
                               CMK = cmk_s$stats$p <= 0.05)
  truths_list[[s]] <- sim_s$true_slope
}
compare_detections(detections_list, truths_list, replicates = TRUE,
                   verbose = FALSE)

# FWER ("did at least one false positive happen in this run at all?")
# only means anything across many replicates -- request it alongside
# the usual metrics with replicates = TRUE; it errors with
# replicates = FALSE, since a single run cannot estimate a rate.
compare_detections(detections_list, truths_list, replicates = TRUE,
                   metrics = c("sensitivity", "fdr", "fwer"),
                   verbose = FALSE)
```

`compute_anomalies` *Remove the seasonal cycle from raster time series*

Description

Monotonic trend tests assume that the input series does not contain a regular periodic component – the ranking of observations should primarily reflect long-term change, not where in the year a value falls. When a seasonal cycle is present (e.g. monthly means over many years), that cycle itself dominates the ranking and can obscure or distort the monotonic signal a trend test is looking for.

Usage

```
compute_anomalies(
  x,
  cycle = 12,
  cycle_type = NULL,
  start_position = 1,
  standardise = FALSE,
  verbose = TRUE
)
```

Arguments

`x`	A `terra::SpatRaster` with `nlyr(x)` a multiple of `cycle` (or not – a partial final cycle is allowed, but its climatology mean will be based on fewer years for those positions).

cycle	Integer. Length of the seasonal cycle in layers – e.g. 12 for monthly data with an annual cycle (the default), 4 for quarterly/seasonal data. A daily annual cycle requires a consistent calendar with the same positions in every year. Ignored if cycle_type is supplied.
cycle_type	Optional named shortcut for cycle, using the same vocabulary as read_ordered_stack()'s own cycle_type (see ?read_ordered_stack, "Supported cycle types"): "monthly" (12), "16-day" (23), "semimonthly" (24), "10-day" (36), "8-day" (46), or "weekly" (52). "annual" and "daily" are deliberately not included – see "Methodological details" below. NULL (default) uses the numeric cycle instead.
start_position	Integer in [1, cycle]. Cycle position of the stack's first layer. Default 1 (the first layer is the first position of its cycle, e.g. January for monthly data). Set this when a series starts mid-cycle – e.g. start_position = 8 for monthly data beginning in August, so the climatology aligns layers to their true calendar position instead of assuming the stack starts at the beginning of a cycle.
standardise	Logical. If FALSE (default), returns raw anomalies (x - climatology_mean), in the original units – preserves the variable's own physical units, and is the more common choice when that is what downstream reporting needs. If TRUE, also divides by the per-cycle-position standard deviation (z-scores: (x - climatology_mean) / climatology_sd), making the anomaly magnitude comparable across cycle positions that have very different natural variability (e.g. winter vs. summer temperature variance) – useful specifically when comparing or combining anomalies across cycle positions whose natural spread differs. Cycle positions with zero standard deviation (constant across all years) get NA anomalies rather than a division by zero.
verbose	Logical. Print progress messages and elapsed time.

Details

This function removes that cycle by computing the mean value at each position within it (e.g. the mean of all Januaries, all Februaries, ...) – the **climatology** – and subtracting it from every layer at that position, leaving an **anomaly series**: successive observations that are directly comparable to one another, with the seasonal pattern removed. This is appropriate input for trend_test() or prewhiten(), both of which assume no periodic component – in a typical workflow, this function is applied first, before prewhiten(), since prewhitening's own AR(1) model assumes the series it receives has no remaining seasonal structure of its own.

Function type: Preprocessing function – prepares seasonal raster time series before prewhitening or trend analysis. It is not itself a trend test or slope estimator.

Value

Returns a list with:

anomalies	A SpatRaster, same number of layers as x – raw or standardised depending on standardise.
climatology	A SpatRaster with cycle layers (the mean field for each position in the cycle), ordered from position 1 to cycle, regardless of start_position.
climatology_sd	Only if standardise = TRUE: a SpatRaster with cycle layers (the standard deviation field for each position in the cycle), in the same order as climatology.

A plain list, not a classed "sptrends" object – unlike prewhiten() or trend_test(), this function's output is typically fed straight into the next preprocessing or inferential step rather than inspected on its own via print()/summary()/ plot().

Typical use

```
seasonal raster time series
    |
compute_anomalies()
    |
anomaly time series (`result$anomalies`)
    |
prewhiten(), trend_test(), or workflow_trends()
```

Apply this step only when x contains a recurring seasonal cycle. workflow_tst() and workflow_trends() start from prewhitening or trend analysis, so pass result$anomalies, rather than the original seasonal series, when deseasonalisation is required.

Methodological details

How it works

A simple mean per cycle position, over the whole series – **not** a moving climatology, a LOESS-smoothed seasonal curve, or a spline fit. This is deliberately the simplest well-defined choice, not a limitation to work around: for the purpose of removing a fixed seasonal pattern before a monotonic-trend test, a fixed per-position mean is sufficient, and a more elaborate seasonal model would add complexity without changing what this function is for.

Statistical assumptions

The seasonal cycle has a known, fixed length and corresponding cycle positions are comparable across repetitions. The estimated fixed climatology is assumed to represent the recurring component that should be removed before later analysis.

Limitations

This function does not detrend the series in any other sense: the climatology is computed directly over the raw values at each cycle position, so a strong underlying trend is already partly absorbed into the climatology mean itself (e.g. if summers have been getting warmer throughout the record, the "mean summer" climatology reflects an average across that warming, not any single year's summer). This is expected, not a defect – it is the trend test applied afterwards that isolates the trend itself; this function's only job is removing the periodic component. Likewise, a genuine regime shift (an abrupt, one-time change in the seasonal pattern partway through the series, as opposed to a gradual trend) would show up mixed into the climatology rather than being detected as such – this function has no mechanism to distinguish a regime shift from ordinary seasonal variability; that is a different kind of question from the one it answers.

Why cycle_type **excludes** "annual" **and** "daily"

"annual" would translate to cycle = 1, which this function always rejects immediately afterwards – an annual series has no sub-annual cycle to remove in the first place, so there is nothing for this function to do with it. "daily" would use a fixed cycle = 365 based on cycle position alone; this function does not use the dates in terra::time(x). Keeping leap days would misalign the climatology for the rest of the series (day 366 would be paired with the cycle position that day 1 of

the following year would otherwise occupy). Use a numeric `cycle` only when each cycle has the same number of comparable positions. Reading daily dates with `read_ordered_stack()` does not change this positional grouping rule.

Quality assurance

Tests verify monthly and arbitrary-cycle climatologies against direct calculations, centred and standardised anomalies, layer names, retained geometry, missing values, zero-variance cycles, non-default starting positions (including partial cycles), and invalid inputs. Workflow tests confirm that anomaly outputs remain compatible with later preprocessing and trend stages. See `?sptrends` for the common release-check protocol.

References

General references for the anomaly/standardisation concept (removing a periodic mean, optionally scaling by its standard deviation) – not a single named method with one original paper, but a standard technique in climatology and atmospheric science:

- Wilks, D.S. (2019) Statistical Methods in the Atmospheric Sciences (4th edn). Elsevier/Academic Press. No DOI available (book).
- Mather, P.M. (1999) Computer Processing of Remotely-Sensed Images. John Wiley and Sons.

See Also

`prewhiten()` for temporal dependence treatment after anomaly construction; `trend_test()` and `workflow_trends()` for subsequent trend inference; `sim_trend_stack()` for controlled example data.

Examples

```
# The bundled NDVI series is annual and therefore has no subannual
# climatological cycle to remove.
r <- read_ordered_stack(example_data("vhp_ndvi"))
terra::nlyr(r)
# Apply compute_anomalies() only to real observations with a genuine
# cycle, for example cycle = 12 for monthly data.
```

`example_data` *Path to sptrends' bundled example dataset*

Description

sptrends is built around one idea: spatiotemporal trend analysis of gridded (raster) data – a stack of layers over the same spatial grid, one layer per time step. `example_data()` gives you a real dataset in exactly that shape, bundled with the package, so every function's `@examples`, every vignette, and your own first attempts at the package have something real to run on without downloading anything.

Usage

```
example_data(path = NULL)
```

Arguments

path	Character or NULL. If NULL (default), lists every bundled example file (as paths relative to the package's extdata directory) – use this first, to see what is available, before deciding what to ask for. If a specific relative path from that listing (e.g. "vhp_ndvi", the whole dataset folder, or one particular .tif file inside it), the full absolute path to it on this machine.

Details

The bundled dataset is annual mean NDVI (Normalized Difference Vegetation Index) – derived from the NOAA STAR Blended Vegetation Health Product, 1982-2023, global land 100 km Eckert IV equal-area grid, ~3.5 MB – a folder of one GeoTIFF per year, which is exactly the layout read_ordered_stack() expects. So example_data() doubles as a realistic example of *that* function's intended use (a folder of yearly rasters), not just a shortcut to a pre-loaded R object. NDVI trend analysis (vegetation "greening" and "browning") is also the motivating application of the True Significant Trends workflow itself – see the primary reference in ?workflow_tst.

Function type: Support function – locates package example files; it performs no statistical analysis.

Value

If path = NULL, a character vector of relative paths. Otherwise, a single absolute file path (as from system.file()). Errors if the requested path does not exist.

Typical use

```
example_data("vhp_ndvi")
    |
read_ordered_stack()
    |
bundled annual raster time series
    |
workflow_tst(), workflow_rta(), or workflow_trends()
```

Call example_data() without an argument first to list all bundled paths.

Methodological details

Data source and licence

Derived from the Blended Vegetation Health Product (Blended-VHP), provided by NOAA's Center for Satellite Applications and Research (STAR) – see example_data("vhp_ndvi/Readme.txt") for full provenance, processing steps, and the required acknowledgement. This derived subset is for demonstration purposes only; it is not a substitute for the original product, which has global land coverage at the original 4 km resolution, weekly observations, and additional variables (brightness temperature, Vegetation Condition Index, Vegetation Health Index).

Limitations

The bundled raster is a small, coarsened demonstration dataset and must not be treated as a replacement for the original NOAA product.

Quality assurance

Tests verify file listing, path resolution, informative failure for absent paths, and end-to-end compatibility of the bundled dataset with `read_ordered_stack()`. See `?sptrends` for the package-wide release-check protocol.

References

NOAA Center for Satellite Applications and Research (STAR). Blended Vegetation Health Product (Blended-VHP). https://www.star.nesdis.noaa.gov/smcd/emb/vci/VH/vh_ftp.php

See Also

Other example data functions: `sim_trend_stack()`

Examples

```
# example_data() with no arguments lists every file bundled with
# the package, so you can see what is available before using any of
# it -- you do not need to know the file names in advance.
example_data()


# Passing "vhp_ndvi" (the folder name from the listing above) gives
# the full path to that folder on your machine. read_ordered_stack()
# reads every GeoTIFF inside it and stacks them into one multi-layer
# raster, one layer per year, already sorted chronologically -- this
# is the "gridded time series" shape sptrends is built around.
# report = FALSE just turns off the automatic year-order check plot,
# to keep this example quiet; leave it on (the default) when
# exploring interactively, as a sanity check on the file order.
# Wrapped in \donttest{} rather than run unconditionally: reading
# all 42 real GeoTIFFs takes several seconds, unlike every other
# example in this package, which uses small synthetic rasters --
# still runs when a user tries this example directly, just not
# timed as part of R CMD check's own example suite.
r <- read_ordered_stack(example_data("vhp_ndvi"), report = FALSE)

# nlyr() ("number of layers") confirms how many years came through --
# 42, one per year from 1982 to 2023.
terra::nlyr(r)

# r[[1]] is the first layer (year 1982) on its own. terra's default
# colour scheme is not designed for a vegetation index, so we ask for
# a better one: grDevices::hcl.colors(50, "Greens 3") gives 50 shades
# from pale to deep green, matching how NDVI maps are normally read
# (higher NDVI, more/denser vegetation).
terra::plot(r[[1]], col = rev(grDevices::hcl.colors(50, "Greens 3")))
```

`fdr_correction` *Apply false discovery rate (FDR) correction to multiple p-values*

Description

Applies one or more false discovery rate (FDR) correction procedures to a previously computed set of p-values. Input may be either a single-layer `SpatRaster` or a numeric vector. This function does not compute p-values itself – it is the final inferential step of the standard sptrends workflow, applied once one raw p-value has already been obtained for every spatial cell (e.g. with `trend_test()` or local `spatial_autocorrelation()`), and it controls the expected false discovery rate across the complete set of simultaneous tests.

Usage

```
fdr_correction(
  p,
  method = c("BH", "BKY", "BY"),
  q = 0.05,
  bky_implementation = c("multtest", "original"),
  moran_check = FALSE,
  moran_args = list(),
  report = TRUE,
  verbose = TRUE
)
```

Arguments

`p`	A single-layer `terra::SpatRaster` of *p*-values, or a numeric vector of raw *p*-values in `[0, 1]`.
`method`	Character vector: which correction(s) to compute. The Usage lists all supported values, `c("BH", "BKY", "BY")`; when `method` is omitted, the established default remains `c("BH", "BKY")`. BH is the classic, fixed-threshold procedure – see `fdr_bh()` for the full detail. BKY is the adaptive method originally used in the TST methodology (Gutiérrez-Hernández & García, 2025): it adapts to the estimated proportion of true nulls, gaining statistical power over BH while still controlling the false discovery rate – see `fdr_bky()` for the full distinction. `"BY"` (Benjamini and Yekutieli, 2001) is a third, deliberately opt-in option – it is never computed unless explicitly requested (e.g. `method = "BY"` or `method = c("BH", "BY")`) – offered as a comparison point and theoretical safeguard against arbitrary dependence structures, not as a routine recommendation; see `fdr_by()` for why.
`q`	Numeric. Target FDR level. This is not a renamed or adjusted α: α is a per-test Type I error threshold, whereas q bounds the expected proportion of false discoveries among all discoveries. A cell is declared significant when its FDR-adjusted *p*-value is at or below q. See `fdr_bh()` for the full distinction.

bky_implementation
: "multtest" (default, unchanged from previous versions) or "original", passed straight to fdr_bky() – see its own documentation for the difference. Ignored if "BKY" is not in method.

moran_check
: Logical. If TRUE, and p is a SpatRaster, automatically run spatial_autocorrelation() on the p-value raster.

  What it does: computes Moran's I on the p-value raster as a diagnostic of spatial association relevant to the positive-dependence assumption behind FDR-BH (Benjamini & Yekutieli, 2001), and reports a plain-language assessment.

  Why: BH's guarantee formally relies on independence or positive dependence between tests. A significant positive Moran's I is compatible with that setting, but does not establish the full PRDS condition. A non-significant or negative result is inconclusive and does not make adaptive BKY a dependence safeguard; use BY when control under arbitrary dependence is scientifically required.

  Limitations: adds computation time (a permutation loop); ignored with a warning if p is a plain numeric vector (no spatial structure to test); a diagnostic, not a guarantee – see "Methodological details" above. Default FALSE.

  More control: for full control over the test (custom nperm, connectivity, n_cores, alternative), call spatial_autocorrelation() directly instead – this argument only covers the common case.

moran_args
: A named list of extra arguments, passed unchanged to spatial_autocorrelation() when moran_check = TRUE (e.g. list(nperm = 999, n_cores = 4)) – no intermediate processing.

report
: Logical. If TRUE (default), automatically print the summary table (fdr_summary()) and draw the diagnostic maps/plots (fdr_significance_maps(), fdr_comparison_barplot(), and, if BKY was requested, fdr_threshold_plot()) after computing. Set to FALSE for programmatic use (e.g. when called from workflow_tst()).

verbose
: Logical. Print progress messages and elapsed time.

Details

Function type: Core function – one of the core building blocks of TST and RTA. Typically follows trend_test() or another inferential function, once one raw p-value exists per spatial cell.

Value

Returns an object of class c("fdr", "sptrends"): a list with q (the target FDR level), method (the requested procedures), p (the raw input p-values, so plot()/summary() are self-contained and don't need the original input kept around separately), and, for each requested method (BH, BKY, and/or BY – q_BH/reject_BH, q_BKY/reject_BKY, q_BY/reject_BY respectively), numeric vectors of FDR-adjusted p-values and rejection decisions, plus summary_bky (Stage-1 figures, if BKY was requested) and threshold_data (for fdr_threshold_plot()), and reject_raw (the unadjusted comparison at the same numerical threshold). When the input is a raster, rasters additionally contains the raw p-value raster and corresponding adjusted-value and significance rasters for the requested methods. If moran_check = TRUE, the list also contains moran, the diagnostic result; moran_assessment, a deliberately qualified interpretation; and the compatibility

field `moran_recommendation`, which is `"BH"` only for significant positive association and `NA` otherwise. Neither field establishes the positive-regression-dependence condition required by BH. Use `print()`/`summary()`/`plot()` – see `print.sptrends()`, `summary.sptrends()`, and `plot.sptrends()`.

Typical use

```
raster time series or one spatial field
    |
trend_test() or spatial_autocorrelation(scope = "local")
    |
raw *p*-values (`trend$stats$p` or `local$p`)
    |
fdr_correction()
    |
adjusted *p*-values + rejection maps at target *q*
```

This function does not accept the time series itself. It operates on the family of raw p-values produced by a preceding inferential method, whether that method comes from sptrends or elsewhere.

Methodological details

Why multiple-testing correction matters for raster data

A raster trend analysis runs one significance test per cell – often thousands at once. Comparing every raw p-value with $\alpha = 0.05$ treats each cell as though it were the only test. That unadjusted interpretation is normally unsuitable as a final raster-wide result: chance alone can produce many apparently significant cells. FDR correction instead evaluates the complete family of p-values. Its target q limits the expected proportion of false discoveries among the cells declared significant; q is not another name for the per-test level α.

An FDR rejection is therefore a statement about membership in a family of discoveries whose expected false-discovery proportion is controlled. It is not a separate local error guarantee for that cell. Calling rejected cells "significant pixels" is convenient shorthand, but does not mean that each cell has error probability q or satisfies an individually adjusted α. The testing domain must also be defined before inspecting the results: changing it afterwards changes the family, ranks, thresholds, and inferential interpretation.

Methods and method selection

Why BH and BKY are recommended: Benjamini & Hochberg (1995) is the classic, widely adopted general-purpose procedure. Across the empirical validation and applications considered during sptrends development, BH has behaved stably, without anomalous inflation of discoveries being observed. Benjamini, Krieger & Yekutieli (2006) extends it adaptively – estimating how much of the map is likely non-null first, then adjusting the threshold accordingly – gaining power in exactly the case common to gridded environmental data, where a real trend affects a sizeable share of the map. Neither supersedes the other; see the `method` argument below for when each is the more defensible choice. BY is available for justified arbitrary-dependence settings, but is not recommended routinely because its safeguard can impose a substantial loss of power.

Statistical assumptions

Role and limit of the spatial autocorrelation diagnostic: BH's own guarantee formally holds under independence or recognised forms of positive dependence, commonly expressed through PRDS (Benjamini & Yekutieli, 2001). Moran's I, computed via `moran_check = TRUE`, is a *diagnostic* for

that assumption – not a mathematical precondition the function checks or enforces. A positive, significant Moran's I is evidence consistent with positive dependence; it is not proof, and its magnitude depends on the neighbourhood definition used. This function never runs that diagnostic automatically (`moran_check` defaults to `FALSE`) – the analyst decides whether to spend the extra computation checking it.

Computational considerations

FDR correction is extraordinarily efficient relative to the preceding cell-wise trend analysis. BH and BY are dominated by sorting the p-values; BKY adds a lightweight adaptive stage. With large raster families, raster input/output may cost more than the correction itself.

BKY is still an FDR procedure: its adaptive first stage estimates a proportion of true nulls to sharpen the threshold, but the underlying guarantee it targets is the same false discovery rate BH targets – it is a refinement of the same family, not a conceptually different kind of correction.

Common interface

Rather than exposing BH and BKY as completely separate workflows, this function provides a common interface returning comparable outputs regardless of the chosen correction – the same one-interface-many-methods philosophy this package uses throughout (see, e.g., `prewhiten()`'s `method` argument). This function wraps three distinct, individually citable methods – see `fdr_bh()`, `fdr_bky()` and `fdr_by()` for their methodological background, original publications and full reference lists. In short: Benjamini & Hochberg (1995) for BH, Benjamini, Krieger & Yekutieli (2006) for the adaptive BKY extension. Typical applications: correcting for multiple testing in gridded environmental data, whichever method's guarantee (fixed vs. adaptive) fits the analysis at hand.

Limitations and interpretation

FDR control concerns the expected proportion of false discoveries among rejected hypotheses; it does not control the probability of making even one false rejection. The guarantee also depends on the assumptions of the selected procedure. `moran_check` provides only a spatial diagnostic and cannot prove PRDS or choose a valid procedure automatically. The unadjusted comparison stored as `reject_raw` uses the numerical value of `q` only as a visual reference threshold; it is not an FDR-controlled result.

It is good practice to always report the number of valid cells actually tested (`m` in this function's own output and in `fdr_summary()`'s table, as `n_valid`) alongside any count or percentage of significant cells. The same percentage means a very different thing depending on whether it comes from 50 valid cells or 50,000 – omitting `m` leaves that scale invisible to the reader, silently expecting them to infer it from raster dimensions that may not even be shown alongside the result.

Quality assurance

BH and BY adjusted values are required to match `stats::p.adjust()` exactly. BKY stages, thresholds, rejection monotonicity, missing values, empty and degenerate inputs, and the equivalence of numeric-vector and `SpatRaster` interfaces are covered by automated tests. Integration tests verify that workflow printing, maps, and summaries report only methods actually requested. See `?sptrends` for the package-wide release-check protocol.

References

Combines the three methods below – see `fdr_bh()`, `fdr_bky()` and `fdr_by()` for the full reference list and the reasoning behind each citation, including why `BY` is offered but not run by default.

- Benjamini, Y., & Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x
- Benjamini, Y., Krieger, A. M., & Yekutieli, D. (2006) Adaptive Linear Step-Up Procedures that Control the False Discovery Rate. Biometrika, 93(3), 491-507. doi:10.1093/biomet/93.3.491
- Benjamini, Y., & Yekutieli, D. (2001) The control of the false discovery rate in multiple testing under dependency. Annals of Statistics, 29(4), 1165-1188. doi:10.1214/aos/1013699998

See Also

Other FDR correction functions: fdr_bh(), fdr_bky(), fdr_by(), fdr_comparison_barplot(), fdr_direction_plot(), fdr_direction_summary(), fdr_pvalue_histogram(), fdr_significance_maps(), fdr_summary(), fdr_threshold_plot()

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))
trend <- trend_test(r, report = FALSE, verbose = FALSE)

# Corrects trend$stats$p for the fact that every cell was tested at once
# (see the "Warning" section of ?trend_test).
fdr_result <- fdr_correction(
  trend$stats$p,
  method = c("BH", "BKY", "BY"),
  report = FALSE,
  verbose = FALSE
)
fdr_result$q_BH   # BH-adjusted p-values, one per cell
fdr_result$q_BKY  # BKY-adjusted p-values, one per cell
fdr_result$q_BY   # BY-adjusted p-values, one per cell
summary(fdr_result)
# Significance maps and a comparison barplot.
plot(fdr_result)

# Each method can also be requested separately -- useful when only
# one is actually needed downstream.
fdr_bh_only  <- fdr_correction(trend$stats$p, method = "BH",
                               report = FALSE, verbose = FALSE)
fdr_bky_only <- fdr_correction(trend$stats$p, method = "BKY",
                               report = FALSE, verbose = FALSE)
fdr_by_only  <- fdr_correction(trend$stats$p, method = "BY",
                               report = FALSE, verbose = FALSE)


# The same FDR implementation accepts local spatial p-values. Global
# spatial_autocorrelation() performs one test, so it needs no FDR.
field <- r[[1]]
local <- spatial_autocorrelation(
  field, scope = "local", nperm = 99, seed = 1,
```

```
  report = FALSE, verbose = FALSE
)
local_fdr <- fdr_correction(
  local$p, method = c("BH", "BKY", "BY"), q = 0.05,
  report = FALSE, verbose = FALSE
)
plot(local_fdr)
```

`inspect_ts_cell` *Inspect a single cell's (or area's) raw time series interactively*

Description

Click a cell (or draw a polygon) on **whatever map is currently displayed** – a trend map, a significance map, a raw data map, it does not matter what it shows, only that it shares the same spatial extent as `x`.

Usage

```
inspect_ts_cell(
  x,
  prewhitened = NULL,
  selection_type = c("point", "polygon"),
  neighbourhood = TRUE,
  connectivity = c("queen", "rook"),
  conf_level = 0.95,
  t = NULL,
  show_neighbours = FALSE,
  slope_method = c("TS", "OLS", "RM"),
  compare_slopes = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	The full time series stack (a `terra::SpatRaster`, one layer per time step) – this is always where the plotted raw series comes from, regardless of what is currently displayed on screen.
`prewhitened`	Optional. The full list returned by `prewhiten()` (not just its `$series`) – if supplied, a second panel shows the same location's prewhitened series, its own Theil-Sen fit and confidence interval, and whether that location was actually modified by prewhitening (many cells are not, if their own Durbin-Watson statistic never crossed the gating threshold; see `?prewhiten`). Default `NULL`: only the raw-data panel is drawn.

`selection_type` `"point"` (default): click a single cell. `"polygon"`: draw a polygon (left-click to add vertices, press `Esc` or right-click to finish).

`neighbourhood` Logical, only used when `selection_type = "point"`. If `TRUE` (default), the clicked cell and its queen (or rook) neighbours are combined – see the "How each mode aggregates its series" section below – before estimating anything, borrowing spatial context the same way `trend_test()` does for significance. If `FALSE`, only the clicked cell's own series is used. Ignored when `selection_type = "polygon"` (a polygon is already an explicit choice of area). Intended for exploratory visualisation, at this one location, rather than a substitute for formal inference – the slope shown here does not carry the same significance guarantee CMK's own variance-adjusted statistic does.

`connectivity` `"queen"` (default, 8 neighbours) or `"rook"` (4 neighbours), passed to `terra::adjacent()`. Only relevant when `neighbourhood = TRUE`.

`conf_level` Confidence level for the fitted slope's confidence interval, reported in the legend as text (not drawn as a shaded band – see "Confidence interval" below). Default `0.95`. Ignored when `slope_method = "RM"` or `compare_slopes = TRUE` (see both below).

`t` Finite numeric vector of unique, strictly increasing time points, one per layer. Defaults to `1:nlyr(x)`.

`show_neighbours` Logical, only used when `neighbourhood = TRUE` and `selection_type = "point"`. Answers, at a glance, a question any user looking at an unusual pixel asks: **is this cell's trend representative of its neighbourhood, or an outlier the aggregation is smoothing over?** If `TRUE`, draws a second figure after the main panel(s): a small-multiples grid, one mini-panel per cell actually aggregated (the clicked cell, highlighted, plus each of its individual queen/rook neighbours), each with its own raw series and fitted line (same `slope_method` as the main panel) – not the median-aggregated series the main panel shows, but each contributing cell on its own. Ignored (with a message) if the clicked cell has no neighbours with complete data (e.g. a corner cell with all-NA neighbours), since there would be nothing to compare against. Default `FALSE`.

`slope_method` Which estimator `slope_estimator()` uses for the fitted line(s) shown here. `"TS"` (default): robust to outliers, matches this package's own default trend workflow. `"OLS"`: ordinary least squares – faster, but sensitive to outliers the way a single anomalous time step in the plot above can be. `"RM"`: Siegel's repeated median – more robust than `"TS"`, but no confidence interval is shown (none implemented for it in this package). Ignored when `compare_slopes = TRUE`. This only affects this function's own quick single-cell fit; it has no bearing on which method a `workflow_tst()`/`workflow_rta()` run itself used.

`compare_slopes` Logical. If `TRUE`, ignores `slope_method` and draws all three estimators (`"TS"`, `"OLS"`, `"RM"`) as separate lines on the same panel(s), point estimates only – no confidence intervals in this mode, since the three use genuinely different inferential frameworks (or none, for `"RM"`) and no single interval formula would honestly apply to all three. Default `FALSE`.

`verbose` Logical. If `TRUE` (default), print the click/draw instructions and the `show_neighbours` no-effect note; matches the `verbose` convention used throughout this package,

though the interactive click/draw step itself still happens either way – this only silences the accompanying messages and elapsed time, not the interaction.

... Ignored.

Details

Why inspect a single cell?: raster trend maps summarise thousands of time series into one image. Inspecting an individual location helps determine whether an apparently unusual pixel reflects a genuine temporal pattern, an isolated outlier, or a behaviour representative of its surrounding neighbourhood – a question no summary map, on its own, can answer.

What it shows: the raw time series behind the clicked location, with a single fitted line overlaid (Theil-Sen or OLS – see `slope_method`), together with its confidence interval.

Prewhitening comparison: optionally, a second panel shows the same location's *prewhitened* series side by side (see `prewhitened` below), so you can see the effect of that step at exactly the location you are looking at.

When to use it: to get oriented early on (click around right after loading data) and to investigate a specific cell that looks anomalous on a map you have already produced.

Built from the package's own pieces: this function is not a separate implementation of its own – the fitted line calls `slope_estimator()`'s own estimator, the neighbourhood aggregation follows `trend_test()`'s own logic, and the prewhitened panel reads `prewhiten()`'s own output directly. It is, in that sense, less a standalone utility than a visual demonstration of how the rest of sptrends' pieces fit together, applied to one location at a time.

Function type: Interactive exploration function – combines existing sptrends estimators and neighbourhood logic at one selected location; it introduces no new inferential method.

Value

Returns invisibly, a list with `cell` (the clicked/representative cell number) and `raw` (a list with `series`, `slope`, `ci_lower`, `ci_upper`, `conf_level` for the raw-data panel). If `prewhitened` was supplied, also `prewhitened` (the same fields for that panel) and `n_modified`/`n_total` (how many of the aggregated cells were actually modified by prewhitening). If `show_neighbours = TRUE` and at least one neighbour had a complete series, also `neighbours`: a list, one element per plotted cell (including the clicked one), each with `cell`, `slope`, and `is_centre`.

Typical use

```
raster time series + a displayed map with matching geometry
    |
inspect_ts_cell()
    |
selected cell or area -> temporal plot + fitted slope
```

Optionally supply the complete `prewhiten()` result to compare the raw and transformed series at the same location.

Methodological details

Methods and method selection

`slope_method` selects Theil-Sen, OLS, or repeated median for the exploratory fit. `compare_slopes = TRUE` displays all three point estimates together; confidence intervals are then omitted because no single inferential framework applies to all three estimators.

How each mode aggregates its series

All three modes – point-only, point-with-neighbourhood, and polygon – follow the *same* two-step procedure, in the *same* order, so a single confidence interval formula applies rigorously to all of them: first take the per-time-step **median of raw values** across whichever cells are included (just the one clicked cell, the clicked cell plus its neighbours, or every cell inside the drawn polygon), producing one aggregated series; *then* fit a single Theil-Sen slope to that series. The same aggregation (same cells) is applied to the prewhitened series too, when `prewhitened` is supplied, so the two panels are directly comparable. Aggregating values first and estimating second is what makes the Sen/Gilbert confidence interval below valid – it is defined for the pairwise slopes of a single series, not for a median of several already-computed slope estimates (the reverse order), which has no standard interval formula. This also means `neighbourhood = TRUE`'s result is **not** the same number as `slope_estimator(x, smooth_neighbourhood = TRUE)` at that same cell in a full-map run – that other mechanism deliberately aggregates in the opposite order (median of independently-computed per-cell slopes); see its own documentation for why. The two exist for different purposes and are not meant to match numerically.

Statistical assumptions and confidence intervals

What it represents: uncertainty in the *rate of change* itself, not a prediction interval around the raw data points. Reported as text in the legend (`ci_lower`/`ci_upper` in the returned value too), not drawn as a shaded band on the plot – only the single fitted line and its slope/intercept are drawn. Not shown when `slope_method = "RM"` or `compare_slopes = TRUE`.

How it is computed (the standard rank-based method for the Theil-Sen slope; Sen, 1968; Gilbert, 1987, pp. 217-219): given the N pairwise slopes of the (already-aggregated) series, sorted, the interval is $[\hat{\beta}_{(M1)}, \hat{\beta}_{(M2+1)}]$, where $M1 = (N - C_\alpha)/2$, $M2 = (N + C_\alpha)/2$, and $C_\alpha = z_{1-\alpha/2}\sqrt{Var(S)}$ (the same Mann-Kendall variance of `S` used throughout this package, tie-corrected). `M1`/`M2` are rounded to the nearest integer rather than interpolated between adjacent ranked slopes, unlike Gilbert's own recommendation – a deliberate simplification, documented here rather than left silent. This formula applies when `slope_method = "TS"` (the default); for `slope_method = "OLS"`, the interval instead uses the standard parametric simple-linear-regression interval (which assumes normally-distributed, homoscedastic residuals) – a genuinely different formula, not the same computation reused, since it rests on different assumptions matching its own point estimate.

Limitations

This is an exploratory, interactive view of selected locations, not a raster-wide significance procedure. Neighbourhood or polygon medians change the series being fitted, and repeated-median or multi-estimator displays do not include confidence intervals.

Quality assurance

Tests cover point and polygon selection, neighbourhood aggregation, raw/prewhitened comparisons, all slope choices, confidence intervals, time metadata, incomplete series, interactive selection failures and silent operation. Core numerical results are compared with direct calculations and `slope_estimator()`. See `?sptrends` for the common release-check protocol.

References

Confidence interval method:

- Sen, P.K. (1968) Estimates of the regression coefficient based on Kendall's tau. Journal of the American Statistical Association, 63, 1379-1389. doi:10.1080/01621459.1968.10480934
- Gilbert, R.O. (1987) Statistical Methods for Environmental Pollution Monitoring. Van Nostrand Reinhold, New York, pp. 217-219.

Origin of the Var(S) formula reused for the confidence interval (the same one `trend_test()` builds on for CMK's adjusted variance; see its own references for that extension):

- Mann, H.B. (1945) Nonparametric tests against trend. Econometrica, 13(3), 245-259. doi:10.2307/1907187
- Kendall, M.G. (1975) Rank Correlation Methods (4th edn). Charles Griffin, London. No DOI available (pre-DOI-era publication).

See Also

`prewhiten()` for transformed time series, `trend_test()` for trend significance, and `slope_estimator()` for raster-wide magnitude estimation.

Examples

```
if (interactive()) {
# Interactive -- run this yourself, requires clicking on a plot.

# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))
terra::plot(r[[1]])         # any single-layer map with the same extent as r

# That cell's own raw series, nothing borrowed from its surroundings.
inspect_ts_cell(r, neighbourhood = FALSE)

# Compare against the function's own default -- the same cell combined
# with its queen neighbourhood -- to see how much borrowing spatial
# context changes the fit.
inspect_ts_cell(r)

inspect_ts_cell(r, selection_type = "polygon")       # draw an area instead

# Raw vs. prewhitened, side by side, at the clicked location
pw <- prewhiten(r, report = FALSE, verbose = FALSE)
inspect_ts_cell(r, prewhitened = pw)

# Is the clicked cell's trend representative of its neighbourhood,
# or an outlier the median aggregation is smoothing over? Draws a
# second figure: one mini-panel per neighbour, plus the clicked cell
# itself (highlighted), each with its own Theil-Sen fit.
inspect_ts_cell(r, show_neighbours = TRUE)

# Faster, but not robust to the effect a single anomalous time step
```

```
# can have on the fitted line -- see the "slope_method" argument.
inspect_ts_cell(r, slope_method = "OLS")

# Siegel's repeated median -- more robust than Theil-Sen, but no
# confidence interval is shown for it (none is implemented here).
inspect_ts_cell(r, slope_method = "RM")

# All three estimators at once, as three lines with no intervals.
inspect_ts_cell(r, compare_slopes = TRUE)
}
```

`plot.sptrends` *Plot a sptrends result*

Description

The graphical exploration counterpart to `print()`'s one-line overview and `summary()`'s textual report – see `print.sptrends()` for the full class list and the rationale for one shared entry point per generic. What `which` (and any other named argument) accepts depends entirely on `x`'s own class – see the sections below, grouped by what kind of result each class represents.

Usage

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

Arguments

`x`	An object of class `"sptrends"` (also one of `"tst"`, `"rta"`, `"workflow_trends"`, `"trend_test"`, `"slope"`, `"prewhiten"`, `"fdr"`, `"spatial_autocorrelation"`, `"compare_detections"`, `"sptrends_simulation"`, `"sptrends_simulation_design"`, or `"sptrends_benchmark"`), from `workflow_tst()`, `workflow_rta()`, `workflow_trends()`, `trend_test()`, `slope_estimator()`, `prewhiten()`, `fdr_correction()`, `spatial_autocorrelation(`, `compare_detections()`, `sim_trend_stack()`, `simulation_design()`, or `benchmark_methods()`.
`...`	Passed on to the underlying plotting logic – see each section below for the arguments (typically `which`, and sometimes `method`, `smooth`, `alpha`, or `path`) `x`'s own class accepts.

Details

Function type: Reporting/derived function – visualises an existing result and does not alter or recompute its statistical analysis.

Value

`x`, invisibly.

Typical use

`result <- workflow_tst(x); plot(result)` draws the class-specific default; use `which` for an alternative diagnostic view where supported.

Methodological details

Published workflow: `"tst"`. By default, draws a binarised trend map: increases and decreases from the selected trend statistic are retained only where the selected multiple-testing procedure rejects the null hypothesis (see `direction_map()`).

`which` – the default and the three main views:

`which`	Draws
`"direction"` (default)	Binarised trend map after FDR correction (titled "TST direction map"; "RTA direction map" for the
`"significance"`	FDR-BH/FDR-BKY significance maps side by side
`"trend"`	Uncorrected trend statistic/p-value/significance/direction
`"slope"`	Theil-Sen slope, masked to significant cells

`which = "trend"` draws **uncorrected** diagnostics only – see the "Warning" section of `?trend_test` before reporting significance from this view. `which = "slope"` is the more reliable source for per-cell direction/magnitude than the CMK statistic `Sm`, since `Sm`'s neighbourhood averaging can occasionally disagree in sign with a cell's own Theil-Sen estimate in a spatially heterogeneous neighbourhood (see "Display smoothing" below for how this view handles that visually).

Eight further views, all **uncorrected** (no FDR, no significance masking) – diagnostics on the raw slope or the raw p-value, not a final result to report:

`which`	Draws
`"slope_map"`	Continuous Theil-Sen slope, unmasked
`"slope_direction"`	Binary sign of the slope, unmasked
`"slope_hist"`	Histogram of slope values with a density curve
`"slope_bar"`	Bar chart of positive/negative/zero slope counts
`"pvalue_map"`	Continuous, uncorrected p-value
`"pvalue_significance"`	Binary significant/not at `alpha`
`"pvalue_hist"`	Classic binned histogram of p-values
`"pvalue_bar"`	Bar chart of significant vs. not

`slope_map`/`slope_direction`/`slope_hist`/`slope_bar` need `x$theil_sen` (i.e. `workflow_tst()` must have been run with `theil_sen = TRUE`); the four `pvalue_*` views always work, since `x$trend` is never `NULL`. `slope_bar`/`pvalue_bar` accept `probability = TRUE` for percentages instead of counts; `pvalue_significance` uses `alpha` (default `0.05`), uncorrected for multiple testing.

`method`: `"BKY"` (default, matching `workflow_tst()`'s default `fdr_method`), `"BH"` or `"BY"` – which correction to use for `which = "direction"` or `which = "slope"`. Ignored otherwise. If `x` only has the other one (e.g. `workflow_tst()` was called with `fdr_method = "BH"`), set `method` to match.
Published workflow: `"rta"`. By default, draws the map you actually want to report: direction of change masked by FDR-BH significance (see `direction_map()`). Unlike the `"tst"` case above,

there is no method argument – workflow_rta() always uses FDR-BH (see ?workflow_rta's "How RTA differs from TST, and why"), so there is nothing to choose between.

which: "direction" (default), "significance", "trend", "slope", and the eight further slope_*/pvalue_* diagnostic views – same meaning as the "tst" case above, but always FDR-BH. **Configurable workflow: "workflow_trends".** Provides the same trend, slope, p-value, significance and direction views, using whichever optional slope and FDR stages were selected. **Display smoothing ("tst"/"rta"/"workflow_trends", which = "slope" only).** smooth: logical. If TRUE (default) and x$theil_sen (x$slope for a "workflow_trends" object – same mechanism, different field name) was **not** already smoothed at source (via workflow_tst()'s own theil_sen_args = list(smooth_neighbourhood = TRUE), which is *not* workflow_tst()'s own default – see ?workflow_tst's "Computational considerations" section for why workflow_tst() and workflow_rta() deliberately do not differ here), the significant-cells-only slope map is smoothed with a queen-3x3 median filter **for this plot only**; if it was already smoothed at source, this is not applied a second time. x$theil_sen itself is never modified by plotting, and neither is any part of the significance decision; this only changes what gets drawn. Because smoothing here runs *after* masking to significant cells (a necessarily sparser set of cells than the full raster), the result can look visually blockier than smoothing a dense, unmasked raster would – this is an expected consequence of masking before smoothing, not a bug (see .mask_and_smooth_slope()'s own na.policy = "omit" for the related, and separate, fix ensuring a non-significant cell is never itself painted with a colour). Setting smooth = FALSE when workflow_tst() already smoothed at source cannot recover the unsmoothed values (they were never kept) – a message explains this if it happens. The plot title always states which stage (if any) applied smoothing, so the display is never ambiguous about what it shows. See ?slope_estimator's "Optional queen-neighbourhood smoothing" section for why this is a display convenience, not a validated estimator, and is not applied to the value returned by workflow_tst(). **Trend estimation: "trend_test".** which: "maps" (default), all four uncorrected diagnostic maps (trend statistic, p-value, significance, and direction), via trend_maps(); "histograms", histograms of the trend statistic and p-value, via trend_histograms(). These are the **uncorrected** result; see the "Warning" section of ?trend_test before reporting significance from them without a multiple-testing correction (see fdr_correction() and the "fdr" case below).

alpha: significance threshold, only used when which = "maps". **Trend estimation: "slope".** Draws the zero-centred diverging slope map; no which argument (only one view exists for this class). **Diagnostic: "prewhiten".** which: "maps" (default), the four diagnostic maps; "histograms", the two diagnostic histograms. **Diagnostic: "fdr".** which: "significance" (default), significance maps (raster input only); "pvalue_histogram", histogram of the raw input p-values; "comparison", bar chart comparing significant counts across whichever of raw/BH/BKY/BY were actually requested; "threshold", the BH vs. BKY step-up threshold plot (only if BKY was requested). **Diagnostic: "spatial_autocorrelation".** Global results draw the null distribution with the observed statistic marked. Local results draw the statistic, empirical z, raw permutation p-value and exploratory raw-significance map. Apply and plot fdr_correction() separately for BH, BKY or BY inference. **Validation: "compare_detections".** Intended mainly for simulation studies – not typically the plot an analyst runs on a real dataset, since it needs a known ground truth to have been scored against in the first place (see compare_detections()).

A grouped bar chart of the comparison table – one group of bars per method, one bar per metric; no which argument (only one view exists for this class). metrics chooses which columns to plot (defaults to all of them). **Simulation and benchmarking.** Simulation plots expose the known signal, slope, direction, breaks, or complete raster series. Design plots show the number of levels per factor. Benchmark plots show performance against a varying scenario factor using lines with uncertainty, grouped bars, replicate boxplots, two-factor heatmaps, or multi-metric profiles. Use metric,

`scenario`, `group`, `facet`, `type`, `interval`, and `level` to configure these views. Confidence intervals use the between-replicate standard error and a Student-t critical value; intervals for rates and powers are clipped to their admissible range from zero to one.

See Also

`print.sptrends()` for a concise overview and `summary.sptrends()` for detailed textual output.

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))
result <- workflow_tst(r, report = FALSE, verbose = FALSE)

# Default map: direction of change (greening/browning/no change),
# masked by FDR-BKY significance -- cells with grey are not
# significant, so they are left out of the coloured pattern.
plot(result)

# The rate of change (not just direction) among significant cells,
# queen-3x3 median smoothed for display (the default for this view).
plot(result, which = "slope")
```

---

prepare_cmk_neighbourhood

Precompute a CMK spatial neighbourhood

Description

Builds the sparse adjacency matrix (`W[i, j]` = 1 if `j` is a queen neighbour of `i` and both have complete data) and the valid-neighbour count per cell. Kept separate from `trend_test()` so it can be reused without recomputation when the test is called repeatedly on the same raster geometry (e.g. inside a permutation loop).

Usage

```
prepare_cmk_neighbourhood(x, ok, connectivity = "queen", window_size = 3L)
```

Arguments

`x`	A `terra::SpatRaster` defining the geometry.
`ok`	Logical vector, length `ncell(x)`: `TRUE` where the cell has a complete time series.
`connectivity`	`"queen"` (default) or `"rook"`.
`window_size`	Odd integer greater than or equal to `3`. Width and height, in raster cells, of the moving neighbourhood. The default `3L` reproduces the CMK region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module.

Details

Function type: Support function – computes something real, but is not one of the core building blocks of TST or RTA (used internally by, or as a standalone diagnostic alongside, the core functions). Called internally by `trend_test()` whenever `precomputed_neighbourhood` isn't supplied; exported directly for the repeated-call optimisation described above, since there is no single-object S3 method this folds into (it takes a raw geometry and a logical vector, not a classed result).

Value

A list with `W` (sparse adjacency `Matrix`), `nb_count` (numeric vector, valid-neighbour count per cell), and an internal geometry/valid-cell signature used to reject unsafe reuse.

References

- Neeti, N. and Eastman, J.R. (2011) A Contextual Mann-Kendall Approach for the Assessment of Trend Significance in Image Time Series. Transactions in GIS, 15(5), 599-611. doi:10.1111/j.14679671.2011.01280.x

See Also

Other Contextual Mann-Kendall functions: `trend_histograms()`, `trend_maps()`, `trend_summary()`

Examples

```
r <- read_ordered_stack(example_data("vhp_ndvi"))

# A matrix with one row per cell and one column per time step, and a
# logical vector marking which cells have a complete (no-NA) series --
# both are what trend_test() needs internally.
X <- terra::values(r, mat = TRUE)
ok <- stats::complete.cases(X)

# Precomputes the spatial adjacency (queen/rook neighbourhood) once,
# so it can be reused across repeated calls instead of recomputed
# every time -- see the precomputed_neighbourhood argument of
# trend_test().
nb <- prepare_cmk_neighbourhood(r, ok, window_size = 3)
names(nb)
```

`prewhiten` *AR(1) prewhitening of raster time series*

Description

Removes temporal (serial) autocorrelation from each cell's time series while preserving its linear trend, using one of four published methods (see the `method` argument below for the choice). The default, `"TFPW_WS"`: for each cell, fit an OLS trend, compute the Durbin-Watson statistic of the residuals, and – **only** for cells where DW indicates relevant serial autocorrelation – iteratively estimate the AR(1) coefficient `rho` and apply a trend-preserving transformation (with a Prais-Winsten correction for the first observation). Cells that pass the DW check are left untouched.

Usage

```
prewhiten(
  x,
  method = c("TFPW_WS", "TFPW_Y", "TFPW_Z", "VCTFPW"),
  t,
  dw_low = 1.4,
  dw_high = 2.6,
  dw_method = c("threshold", "test"),
  dw_inconclusive = c("conservative", "power"),
  eps = 1e-04,
  itmax = 20,
  refit_method = c("OLS", "TS"),
  report = TRUE,
  verbose = TRUE
)
```

Arguments

`x` A `terra::SpatRaster`; each layer is one time step, in increasing chronological order.

`method` Which prewhitening method to use, sharing one interface between four related procedures (see "Methodological details" below for the full comparison and citations).

`"TFPW_WS"` (default; TFPW-WS, also seen as "Wang-Swail prewhitening" in the literature):

- **Description**: the selective AR(1) prewhitening described above, gated by a Durbin-Watson diagnostic.
- **Characteristics**: only cells that need it get modified; `rho` is estimated on the raw trend residuals.
- **Advantage**: avoids the power loss of prewhitening cells that already behave like white noise (see "Statistical rationale" above).

`"TFPW_Y"` (TFPW-Y, also seen simply as "TFPW" – this is the best-known formulation of trend-free prewhitening in the literature):

- **Description**: trend-free pre-whitening (Yue, Pilon, Cavadias, 2002).
- **Characteristics**: estimates a Theil-Sen slope per cell first, removes it, estimates lag-1 autocorrelation on the *detrended* residuals (not the raw series, which would over-estimate rho when a real trend is present), prewhitens

those residuals, and adds the slope back; every valid cell is processed unconditionally, there is no DW gate.

- **Advantage**: a single, simple rule applied uniformly, with no threshold or test choice to make.

`"TFPW_Z"` (TFPW-Z here – not a universally established acronym in the literature the way the others are, so defined explicitly wherever used):

- **Description**: Zhang, Vincent, Hogg and Niitsoo (2000), as later refined into `TFPW_WS` above – this method is the earlier, unrefined form the same iterative mechanism (see "Implementation notes" above) is traced against and mirrors.
- **Characteristics**: identical iteration to `TFPW_WS`, but with no Durbin-Watson gate – every valid cell is processed unconditionally, matching `zyp::zyp.TFPW_Z()`'s own published behaviour.
- **Advantage**: no gating decision to make, at the same cost `TFPW_Y` has for the same reason.

`"VCTFPW"` (VCTFPW):

- **Description**: variance-corrected trend-free prewhitening (Wang, Chen, Becker and Liu, 2015).
- **Characteristics**: removes a Sen slope, estimates lag-1 autocorrelation on the detrended residuals and applies the published transformation only where that autocorrelation is significant at the two-sided 95% level. It then applies the published variance-ratio correction and corrects the restored slope for positive autocorrelation. Cells that do not cross the gate retain their original series.
- **Advantage**: addresses a specific, published criticism of simpler trend-free methods – their own reduced variance biases the slope and its significance – at the cost of a substantially more involved procedure.

Comparison: `TFPW_WS` uses a Durbin-Watson gate, `TFPW_Z` and `TFPW_Y` process every valid cell unconditionally, and `VCTFPW` uses its published 95% lag-1-autocorrelation gate before applying its variance and slope corrections. See "Methodological details" above for the full citations and practical implications of each.

`t` Numeric vector of time points, one per layer. Defaults to `1:nlyr(x)`. Irregular spacing is supported, but values must be finite, unique and strictly increasing.

`dw_low, dw_high` Only used when `method = "TFPW_WS"`. Fixed Durbin-Watson gate thresholds, used only when `dw_method = "threshold"`.

`dw_method` Only used when `method = "TFPW_WS"`. `"threshold"` (default): use the fixed `dw_low`/`dw_high` cutoffs. `"test"`: use the formal Durbin-Watson test with critical values (dL, dU) tabulated for the actual sample size, at the 5% level.

`dw_inconclusive`

Only used when `method = "TFPW_WS"` and `dw_method = "test"`. The classic DW test has an inconclusive zone (`dL <= DW <= dU`) where it cannot decide – this is *not* the same as "autocorrelation present". `"conservative"` (default) prewhitens that zone too (favours not leaving autocorrelation uncorrected); `"power"` prewhitens only when the test actually rejects H0 (`DW < dL`), favouring statistical power downstream. Neither is "the correct" choice in absolute terms.

eps — Only used when method = "TFPW_WS" or "TFPW_Z". Convergence threshold for the iterative estimation of rho.

itmax — Only used when method = "TFPW_WS" or "TFPW_Z". Maximum number of iterations. TerrSet's own Earth Trends Modeler, which implements this same method, caps this at 5 "to avoid the rare cases that fail to converge" (its own documentation's wording) – sptrends uses a higher default (20) instead, relying on Clamped (see "Value" below) to flag a cell whose own iteration did not settle within that budget, rather than capping the budget itself as tightly as TerrSet does.

refit_method — Only used when method = "TFPW_WS" or "TFPW_Z". Which slope estimator refits the trend at each iteration, before removing it to isolate the residual autocorrelation for the next rho estimate: "OLS" (default, matching this function's own behaviour in earlier package versions) or "TS" (Theil-Sen, matching the original iterative procedure as multiple independent sources describe Wang and Swail (2001) actually publishing it – see "Implementation notes" below for the specific evidence and why the default was not simply changed to match it outright). "TS"'s own robustness to outliers can matter specifically for a cell whose iteration is already unstable (see Clamped below): an extreme value in one iteration's own transformed series distorts an "OLS" refit more than a "TS" one, which can itself feed into a worse rho estimate the following iteration. Comparing both directly on a specific Clamped = 1 cell, the same way comparing TFPW_WS against TFPW_Y is already recommended for that case (see "Value" below), is worth doing before trusting either estimate in isolation.

report — Logical. If TRUE (default), automatically print a summary and draw diagnostic histograms and maps after computing (which functions, exactly, depends on method – see "Value" below). Set to FALSE for programmatic use (e.g. inside a loop, or when called from workflow_tst()) where you don't want console output or plots as a side effect.

verbose — Logical. Print progress messages and elapsed time.

Details

Function type: Preprocessing function – prepares the raw raster time series before trend estimation or significance testing (see compute_anomalies() for the other preprocessing step this package offers). Not one of the core trend-analysis pillars itself. This function typically precedes trend estimation (trend_test()) and slope estimation (slope_estimator()) within the standard sptrends workflow.

Value

Returns a list of class "prewhiten", with:

series — A SpatRaster. For method = "TFPW_WS": same structure as x, with prewhitened cells replaced and the rest unchanged; layer names get a "_prewhitened" suffix. For method = "TFPW_Y": **one fewer layer than** x (the classic Yue-Pilon transform loses the first time step to lag-1 differencing); every valid cell is prewhitened, with its Theil-Sen trend preserved; layer names come from x's own layers 2 through n, with a "_prewhitened" suffix. For method = "TFPW_Z": the

same number of layers as x; every valid cell enters the same iterative transformation used by TFPW_WS, but without the Durbin-Watson gate. For method = "VCTFPW": the same number of layers as x; significantly autocorrelated cells contain the published VCTFPW transformation and other valid cells remain unchanged.

diagnostics — A SpatRaster. For method = "TFPW_WS": 4 layers, DW_initial, Rho, Modified (0/1), and Clamped (0/1) – Clamped is 1 for a gated cell whose iterative rho estimate hit the +-1 stability bound (see "Implementation notes" below) at any point during the iteration, even if a later iteration's own rho - rho_old difference happened to fall under eps immediately afterwards, since a Rho reaching that bound reflects the bound itself, not necessarily a precise, well-converged estimate. A clamped cell is deliberately left uncorrected (series keeps its original observed values there, Modified is 0 for it) rather than transformed using that unreliable rho: the correction divides by (1 - rho), so at rho = 0.99 the residuals would be multiplied by 100 – an extreme, disproportionate transform for what was, underneath, an unstable rather than a trustworthy estimate. Rho still records the clamped value itself, so the reason a cell went uncorrected remains visible in the diagnostics rather than indistinguishable from a cell that never needed correcting in the first place. Reachable via summary()/plot() as prewhiten_summary()/prewhiten_histograms()/ prewhiten_maps(). For method = "TFPW_Y": 2 layers, Beta_TheilSen (the slope removed and restored) and Rho (lag-1 autocorrelation of the detrended residuals); reachable via internal reporting functions specific to this method, not the TFPW_WS ones above (their structure differs – there is no DW gate, Modified, or Clamped field for TFPW_Y, which does not iterate to estimate rho and so cannot hit this same failure mode – precisely why comparing both methods on the same data is worth doing for a cell flagged Clamped = 1, rather than trusting either one in isolation: agreement between them is reassuring, and disagreement is itself informative about how sensitive that cell's own result is to the prewhitening method chosen.). For method = "TFPW_Z": the same 4 layers as TFPW_WS, but DW_initial is NA because this method has no Durbin-Watson gate; every valid cell enters the iteration, while Modified and Clamped retain the meanings described above. For method = "VCTFPW": 3 layers, Rho (lag-1 autocorrelation of the detrended series), Beta_corrected, and Modified (0/1, indicating whether the 95% autocorrelation gate was crossed).

method — The selected method, recorded as one of "TFPW_WS", "TFPW_Y", "TFPW_Z", or "VCTFPW".

Typical use

This function is the preprocessing step between reading data and testing it for a trend: read_ordered_stack()/read_netcdf
-> compute_anomalies() (optional) -> prewhiten() -> trend_test() -> slope_estimator()
-> fdr_correction(). workflow_tst() runs this whole chain in one call.

```
raster time series
    |
prewhiten()
    |
```

```
transformed time series (`result$series`) + diagnostics
    |
trend_test() and slope_estimator()
```

If the input has a seasonal cycle, remove it first with `compute_anomalies()`. Pass `result$series` to later stages; the original input object is not modified and remains available under the name supplied by the caller.

Methodological details

Methods and method selection

- **Wang & Swail (2001),** `method = "TFPW_WS"`: trend-free prewhitening (a Prais-Winsten-style correction that preserves the linear trend, unlike simple differencing).
- **Yue, Pilon & Cavadias (2002),** `method = "TFPW_Y"`: also a trend-preserving prewhitening method, but a different mechanism: `rho` is estimated on the detrended residuals directly, every valid cell is processed unconditionally (no DW-based gate), and the classic transform loses the first time step. See the `method` argument below for the practical difference this makes to the output.
- **Zhang et al. (2000),** `method = "TFPW_Z"`: uses the same iterative mechanism as `TFPW_WS`, but applies it to every valid cell without a Durbin-Watson gate.
- **Wang et al. (2015),** `method = "VCTFPW"`: applies the published variance and slope corrections only where lag-1 autocorrelation crosses its two-sided 95% gate.
- **Main references**: Wang & Swail (2001) for the `TFPW_WS` transformation itself; Durbin & Watson (1950, 1951) for the statistic used as its selective gate; Yue & Wang (2002) for why gating selectively, rather than prewhitening every cell, matters for `TFPW_WS` specifically; Yue, Pilon & Cavadias (2002) for `TFPW_Y`; Zhang et al. (2000) for `TFPW_Z`; and Wang et al. (2015) for `VCTFPW`. Full citations appear under "References" below.
- **Typical applications**: hydrology and climatology time series with suspected serial autocorrelation, applied before a Mann-Kendall-family trend test to avoid inflating its false positive rate.

Classical prewhitening is deliberately not provided as a fifth method because filtering the observed series directly can remove or attenuate part of the trend and reduce the power of the subsequent test. The four implemented procedures explicitly preserve or restore the trend, or otherwise correct the transformation.

How it works

Prewhitening every cell indiscriminately (including cells that already behave like white noise) reduces the statistical power of any trend test applied afterwards (Yue, Pilon & Cavadias, 2002); hence the selective gate.

The method preserves the linear trend, unlike simple differencing, by solving `Y_t = a + b*t + X_t` with `X_t = rho * X_(t-1) + e_t` for `W_t = (Y_t - rho * Y_(t-1)) / (1 - rho)` (Wang & Swail, 2001, "trend-free prewhitening").

This selective, DW-gated approach to prewhitening follows the same overall methodology as the Earth Trends Modeler (ETM) module of TerrSet (Clark Labs) – an independent implementation, not a port of that software's code.

Implementation notes

On the core iterative mechanism: this function's own iteration was substantially rewritten in this version to mirror `zyp::zyp.TFPW_Z()`'s own mechanics, after empirically comparing this function's own `rho` estimate against two independent implementations of essentially the same published method (`zyp`'s `"TFPW_Z"`, and `MannKendallTrends`'s `nanprewhite.AR()`/`prewhite()`) on a real near-unit-root cell: this function's own earlier iteration hit the `+-1` stability clamp (`0.99`), while `zyp` converged to `0.776` and `MannKendallTrends` to `0.860` – both substantially more moderate, and reasonably close to each other despite differing implementations. Tracing both algorithms step by step on the identical series (see `NEWS.md` for the specific numbers at each iteration) found two mechanical differences, both now adopted: first, each iteration measures lag-1 autocorrelation on the *raw* series detrended by the latest slope estimate, never on a running transformed series – so what changes between iterations is only which slope estimate is used to detrend the same raw data, not the data being detrended itself; second, the very first estimate, before any detrending, is the raw series' own lag-1 autocorrelation directly, matching `zyp`'s own initial step, rather than the residuals of an initial trend fit (which, on a near-unit-root series, can themselves carry *more* apparent autocorrelation than the raw series did, pushing the very first estimate toward the clamp before any subsequent iteration gets a chance to recover from it).

On `refit_method` and the evidence behind its default: TerrSet's own Earth Trends Modeler documentation for this method states only that its iteration is "determined exactly as described by Wang and Swail (2001)", without specifying which slope estimator that refitting step itself uses – this function's own primary source does not settle the question directly. Independently, several papers describing the original Wang and Swail (2001) procedure in more methodological detail (including Zhang and Zwiers (2004), a direct comment on this literature, and Collaud Coen et al. (2020)) describe its own iterative refit as using the Sen slope specifically, and `MannKendallTrends`'s own `prewhite()` source, inspected directly, confirms this: every refit inside its own `TFPW.WS` branch calls its package's own `sen.slope()`, never an OLS fit. Zhang and Zwiers (2004) describe substituting an OLS refit as their own added variant for comparison, not as what Wang and Swail (2001) themselves used. This function's own default (`refit_method = "OLS"`) was kept despite this evidence, both because the primary source available to this package does not confirm it directly, and because `"OLS"` was this function's own behaviour in earlier package versions – with the outer iteration mechanism now unified between the two values (see above), the choice of refit estimator itself is a materially smaller effect than it was before that rewrite; `"TS"` remains available for a user who prefers it or wants to compare both directly, as recommended above for a `Clamped = 1` cell specifically.

Computational considerations

On TerrSet's own iteration cap: TerrSet's documentation states its own maximum of 5 iterations "to avoid the rare cases that fail to converge" – tested directly against a real near-unit-root cell under this function's own earlier iteration, raising `itmax` from 20 to 100 left the estimate unchanged (still at the clamp), showing the earlier instability was a genuine fixed point of that iteration, not merely an iteration budget cut short – so `itmax` was not lowered to match TerrSet's own cap; this function's default (20) is kept.

`TFPW_WS` and `TFPW_Z` are iterative. Within either procedure, `refit_method = "OLS"` is computationally lighter than `"TS"`; Theil-Sen refitting gains robustness at the cost of evaluating many pairwise slopes. The Theil-Sen steps in `TFPW_Y` and `VCTFPW` also become more expensive as the number of time points increases.

Statistical assumptions

All four methods assume an AR(1) noise process (no higher orders) and an approximately linear trend.

Limitations

This is a purely temporal (per-cell) preprocessing step with no spatial component. Cells with any missing value in the time series are excluded entirely from the output.

`method = "TFPW_WS"`: the `"threshold"` method uses a widely-used but non-universal empirical DW cutoff (`[1.4, 2.6]`); `rho` is assumed constant across the whole series.

`method = "TFPW_Y"`: loses one observation to lag-1 differencing; unlike `TFPW_WS`, has no selective gate, so a cell with genuinely no autocorrelation still gets a (small) `rho` estimated and applied from that specific sample – see `TFPW_WS` above for the alternative that only touches cells that need it.

`method = "TFPW_Z"`: processes every valid cell without a selective gate. As with `TFPW_WS`, an iteration that reaches the stability clamp is reported and that cell is left uncorrected.

`method = "VCTFPW"`: uses the method's published two-sided 95% lag-1-autocorrelation gate and variance correction. Its slope correction follows the published positive-autocorrelation rule.

Quality assurance

Automated tests compare the transformed series and diagnostics with hand-calculated references, verify method-specific gates and first-year retention, exercise constant/invalid series and boundary cases, and require identical sequential and parallel results. Yue-Pilon trend-free prewhitening is additionally compared with `modifiedmk::tfpwmk()` for the quantities both implementations define identically. See `?sptrends` for the package-wide release-check protocol; current check results belong only in `cran-comments.md`.

References

Primary method reference (`method = "TFPW_WS"`):

- Wang, X.L. and Swail, V.R. (2001) Changes of Extreme Wave Heights in Northern Hemisphere Oceans and Related Atmospheric Circulation Regimes. Journal of Climate, 14(10), 2204-2221.

On the evidence behind `refit_method`, and the core iterative mechanism this version's rewrite mirrors (see "Implementation notes" above): the original iterative method this function's own `method = "TFPW_WS"` implements, as refined by Wang and Swail (2001) from an earlier procedure, and the R package whose own mechanics this version's rewrite was traced against and now mirrors:

- Zhang, X., Vincent, L.A., Hogg, W.D. and Niitsoo, A. (2000) Temperature and precipitation trends in Canada during the 20th century. Atmosphere-Ocean, 38(3), 395-429. doi:10.1080/07055900.2000.9649654
- Bronaugh, D. and Werner, A. (2013) zyp: Zhang + Yue-Pilon Trends Package. R package. https://CRAN.R-project.org/package=zyp
- Zhang, X. and Zwiers, F.W. (2004) Comment on "Applicability of prewhitening to eliminate the influence of serial correlation on the Mann-Kendall test" by Sheng Yue and Chun Yuan Wang. Water Resources Research, 40(3), W03805. doi:10.1029/2003WR002073
- Collaud Coen, M., Andrews, E., Bigi, A., Martucci, G., Romanens, G., Vogt, F.P.A. and Vuilleumier, L. (2020) Effects of the prewhitening method, the time granularity, and the time segmentation on the Mann-Kendall trend detection and the associated Sen's slope. Atmospheric Measurement Techniques, 13(12), 6945-6964. doi:10.5194/amt1369452020

Primary method reference (`method = "TFPW_Y"`):

- Yue, S., Pilon, P., Phinney, B. and Cavadias, G. (2002) The influence of autocorrelation on the ability to detect trend in hydrological series. Hydrological Processes, 16(9), 1807-1829. doi:10.1002/hyp.1095

(Corrected from an earlier, different Yue, Pilon and Cavadias (2002) paper this citation previously pointed to – Power of the Mann-Kendall and Spearman's rho tests for detecting monotonic trends in hydrological series, Journal of Hydrology 259, a related but distinct paper by essentially the same authors, published the same year, that does not itself specify the TFPW procedure. Found by tracing which paper `zyp`'s own documentation and the `mannkendall` project's own "Spirit of mannkendall" reference for this same method – both cite the paper now cited here.)

Primary method reference (`method = "VCTFPW"`):

- Wang, W., Chen, Y., Becker, S. and Liu, B. (2015) Variance Correction Prewhitening Method for Trend Detection in Autocorrelated Data. Journal of Hydrologic Engineering, 20(12), 04015033. doi:10.1061/(ASCE)HE.19435584.0001234

`VCTFPW`'s own implementation here was adapted from the logic of (not copied verbatim from – see "Implementation notes" above for this package's own vectorised helpers used instead) the R package whose scientific article is already cited above for `refit_method`:

- Collaud Coen, M., Andrews, E., Bigi, A., Martucci, G., Romanens, G., Vogt, F.P.A. and Vuilleumier, L. (2020) Effects of the prewhitening method, the time granularity, and the time segmentation on the Mann-Kendall trend detection and the associated Sen's slope. Atmospheric Measurement Techniques, 13(12), 6945-6964. doi:10.5194/amt1369452020

Source of the Durbin-Watson statistic used as the selective gate (`method = "TFPW_WS"` only):

- Durbin, J. and Watson, G.S. (1950) Testing for Serial Correlation in Least Squares Regression, I. Biometrika, 37(3-4), 409-428. doi:10.1093/biomet/37.34.409
- Durbin, J. and Watson, G.S. (1951) Testing for Serial Correlation in Least Squares Regression, II. Biometrika, 38(1-2), 159-178. doi:10.1093/biomet/38.12.159

Basis for gating prewhitening selectively rather than applying it to every cell (see "Methodological details" above):

- Yue, S. and Wang, C.Y. (2002) Applicability of prewhitening to eliminate the influence of serial correlation on the Mann-Kendall test. Water Resources Research, 38(6), 4-1. doi:10.1029/2001WR000861

Official software implementation (independent re-implementation of the published method, not a port of this module's code):

- Eastman, J.R. (2016) TerrSet Geospatial Monitoring and Modeling System: Earth Trends Modeler. Clark Labs, Clark University, Worcester, MA.

This function is used (not authored) by the following study:

- Gutiérrez-Hernández, O. and García, L.V. (2025) Uncovering true significant trends in global greening. Remote Sensing Applications: Society and Environment, 37, 101377. doi:10.1016/j.rsase.2024.101377

See Also

Other Prewhitening functions: prewhiten_histograms(), prewhiten_maps(), prewhiten_summary()

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))

# Remove serial autocorrelation before testing for a trend -- only
# cells that actually show relevant autocorrelation are modified;
# the rest pass through unchanged.
result <- prewhiten(r, report = FALSE, verbose = FALSE)

# result$series is the prewhitened stack, same number of layers as the
# input -- feed this into trend_test()/slope_estimator()
# next, not the raw r.
terra::nlyr(result$series)
summary(result)
plot(result)   # Rho map and the DW-before/after comparison, real data

# Trend-free pre-whitening (Yue-Pilon): every valid cell is
# processed, its Theil-Sen trend preserved -- result_yp$series has
# one fewer layer than r (the classic transform loses the first
# time step to lag-1 differencing).
result_yp <- prewhiten(r, method = "TFPW_Y", report = FALSE,
                       verbose = FALSE)
terra::nlyr(result_yp$series)
terra::plot(result_yp$diagnostics$Beta_TheilSen,
            main = "Preserved Theil-Sen slope (Yue-Pilon)")
```

---

print.sptrends *Print a sptrends result*

Description

A quick, one-line-per-detail overview of any classed object this package returns – workflow_tst(), workflow_rta(), workflow_trends(), trend_test(), slope_estimator(), prewhiten(), and fdr_correction() all return an object with "sptrends" as (one of) its classes, and print(), summary.sptrends(), and plot.sptrends() all work the same way regardless of which one you have – the specific one-line summary shown depends on x's own class, listed below. This mirrors the convention used throughout terra itself (a single print(), whether x is a SpatRaster or a SpatVector): one predictable entry point per generic, not a different function name to remember for each result type. The API is organised around the *object* a function returns, not around remembering which reporting function goes with which: x <- workflow_tst(...), then print(x), summary(x), plot(x), regardless of what x actually is.

Usage

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

Arguments

x	An object of class "sptrends" (also one of "tst", "rta", "workflow_trends", "trend_test", "slope", "prewhiten", "fdr", "spatial_autocorrelation", "compare_detections", "sptrends_simulation", "sptrends_simulation_design", or "sptrends_benchmark"), from workflow_tst(), workflow_rta(), workflow_trends(), trend_test(), slope_estimator(), prewhiten(), fdr_correction(), spatial_autocorrelation(compare_detections(), sim_trend_stack(), simulation_design(), or benchmark_methods().
...	Ignored.

Details

Generic	Purpose
print()	Quick overview
summary()	Detailed textual report
plot()	Visual exploration

print() itself is intended for a quick inspection of an object at the console; use summary.sptrends() for more detailed textual reporting and plot.sptrends() for graphical exploration.

Function type: Reporting/derived function – presents an existing result and does not compute a new statistical estimate.

Value

x, invisibly.

Typical use

result <- workflow_tst(x); print(result) for a concise console overview.

Methodological details

Published workflow: "tst". Whether prewhitening ran, how many cells were tested, and how many are significant after FDR correction (if run). **Published workflow: "rta".** The Theil-Sen slope range, the trend test's cell count, and how many cells are significant after FDR-BH correction. **Configurable workflow: "workflow_trends".** The selected preprocessing, trend-test, slope and FDR stages, including skipped optional stages and the qualified Moran assessment when requested. **Trend estimation: "trend_test".** How many cells were tested, and how many are significant at the conventional alpha=0.05 threshold, uncorrected. **Trend estimation: "slope".** How many cells have a valid slope, and its range. **Diagnostic: "prewhiten".** How many cells were prewhitened, out of how many valid cells. **Diagnostic: "fdr".** How many cells are significant under each method that was requested (raw, BH, BKY, and BY, if it was explicitly requested – see ?fdr_correction's own method argument for why "BY" is opt-in, not part of its own default).

Diagnostic: `"spatial_autocorrelation"`**.** Global results show the observed statistic (Moran's I or Getis-Ord General G), its sign where applicable, and its permutation p-value. Local results show the valid-cell count and the exploratory number of cells below the raw `alpha`, followed by the route to `fdr_correction()` for BH, BKY or BY. **Validation:** `"compare_detections"`**.** The comparison table itself, printed as a plain data frame (this class also inherits from `"data.frame"`, so indexing, `$`, and so on all work exactly as they would on any other one). **Simulation and benchmarking.** Simulation objects report their dimensions, dependence model and known signal. Designs report scenario counts and varied factors. Benchmarks report their stage, methods, scenarios, replicates and elapsed time.

See Also

`summary.sptrends()` for detailed textual output and `plot.sptrends()` for graphical exploration.

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))
result <- workflow_tst(r, report = FALSE, verbose = FALSE)
print(result)  # dispatches to the "tst" case above
```

`read_netcdf_stack` *Read and chronologically order a single multi-temporal NetCDF file*

Description

Wraps `terra::rast()` for the common case of one NetCDF file holding an entire time series (e.g. reanalysis or climate model output), verifying that the layers come out in chronological order using the file's own time dimension (via `terra::time()`) rather than assuming the on-disk layer order is already correct.

Usage

```
read_netcdf_stack(path, var = NULL, report = TRUE, verbose = TRUE)
```

Arguments

`path`	Character. Path to the `.nc` file.
`var`	Character or `NULL`. Variable name to read, if the file has more than one. If `NULL` and the file has a single variable, that one is used; if it has several, the function stops and lists them.
`report`	Logical. If `TRUE` (default), draw a diagnostic plot of stack position vs. time value (a perfect diagonal line confirms correct ordering; deviations reveal layers that would otherwise have been read in the wrong temporal order).
`verbose`	Logical. Print progress messages and elapsed time.

Details

Why this matters more than it looks: a trend analysis assumes that successive raster layers represent the true chronological sequence. A NetCDF file's own internal layer order and its time dimension are two separate pieces of metadata, written independently – nothing in the file format itself guarantees they agree, and if they do not, every subsequent statistical result becomes invalid even though the analysis completes without any error. This function reorders by the time dimension explicitly rather than trusting the on-disk order, the same "surface a silent risk rather than let it pass unnoticed" instinct behind `read_ordered_stack()`'s own, stricter, filename-based check (see its own documentation for the fuller reasoning, which applies here too).

Function type: Data import function – builds an ordered `SpatRaster` from one multi-temporal NetCDF file. It performs no statistical inference.

Value

A `terra::SpatRaster`, ordered chronologically, with layer names taken from the time values, and the time dimension itself preserved as proper time metadata (readable via `terra::time()`) rather than discarded after the verification step above.

Typical use

```
NetCDF file with a time dimension
    |
read_netcdf_stack()
    |
chronologically ordered raster time series
    |
compute_anomalies() if seasonal, then a trend workflow
```

Methodological details

Temporal ordering

Layers are reordered explicitly from the NetCDF time coordinate; the function does not assume that physical layer order and temporal metadata already agree.

Monthly or seasonal data

This function only orders layers chronologically – it does not remove or otherwise account for a seasonal cycle (e.g. monthly reanalysis values with an annual signal, very common in NetCDF climate data). If the detected time step looks sub-annual (based on `terra::timeInfo()`), a `warning()` is issued reminding you to deseasonalise with `compute_anomalies()` *before* passing the result to `trend_test()`, `slope_estimator()`, or `workflow_tst()`, all of which assume a monotonic trend, not a periodic one. Ordering and deseasonalisation solve different problems.

Limitations

A usable, unambiguous time coordinate is required. With multiple variables, `var` must identify the intended field; this function does not guess among scientifically different variables.

Quality assurance

Tests verify time-coordinate extraction and ordering, variable selection, layer naming, preserved `terra` geometry/time metadata, and failures for absent or ambiguous NetCDF variables. See `?sptrends` for the package-wide release-check protocol.

See Also

Other Data import functions: read_ordered_stack()

Examples

```
if (requireNamespace("ncdf4", quietly = TRUE)) {
  # Convert the bundled environmental series to one temporary NetCDF.
  r <- read_ordered_stack(example_data("vhp_ndvi"))
  path <- tempfile(fileext = ".nc")
  terra::writeCDF(r, path, varname = "ndvi", overwrite = TRUE)
  s <- read_netcdf_stack(path)
  terra::nlyr(s)
  terra::time(s)
  unlink(path)
}
```

---

read_ordered_stack *Read and chronologically order a folder of raster files*

Description

list.files() sorts alphabetically, not numerically: with file names that lack leading zeros ("image1", ..., "image10"), "image10" sorts before "image2" – the same problem regardless of whether the numbering is a year, a time step, or any other sequential index. Using that order silently invalidates a trend analysis with no visible error anywhere in the code. This function instead extracts an explicit ordering number from each file name and sorts numerically by it, then prints and (optionally) plots a verification of the detected order.

Usage

```
read_ordered_stack(
  dir = NULL,
  pattern = "\\.(tif|tiff|nc|grd|img|vrt|asc)$",
  order_regex = NULL,
  candidate_regex = c("year([0-9]+)", "_A([0-9]{4})", "(19[0-9]{2}|20[0-9]{2})",
    "([0-9]{4})", "([0-9]+)"),
  var = NULL,
  report = TRUE,
  verbose = TRUE,
  files = NULL,
  time = NULL,
  cycle_type = NULL,
  start = NULL,
  end = NULL,
  time_anchor = "centre"
)
```

Arguments

`dir`	Character. Path to a folder containing one raster file per time step, all with the same extent, resolution, and CRS.
`pattern`	Character. Regular expression used to list candidate files. The default matches common raster extensions: `.tif`, `.tiff`, `.nc`, `.grd`, `.img`, `.vrt`, and `.asc`. Supply another expression for any additional format supported by `terra::rast()`/GDAL. The extension identifies candidate files; actual readability and compatible geometry are still validated when the stack is opened.
`order_regex`	Character or `NULL`. A regular expression with a single capture group, `"(...)"`, that extracts the ordering number from each file name – e.g. `"year([0-9]+)"` extracts 25 from `"year25.tif"`. If `NULL` (default), several common patterns are tried automatically (year-like `"year25"`, MODIS-style `"_A2000065"`, a bare 4-digit year, or any run of digits) and the first one that extracts a **unique** number from every file is used. If none does, the function stops rather than silently falling back to alphabetical order.
`candidate_regex`	Character vector of patterns tried automatically when `order_regex` is `NULL`. Most users should never need to modify this.
`var`	Character or `NULL`. For NetCDF input only: variable name to read from each file if each file has more than one variable. Leave `NULL` for ordinary single-variable raster formats.
`report`	Logical. If `TRUE` (default), draw a diagnostic plot of stack position vs. detected order number (a perfect diagonal line confirms correct ordering). Deviations from the diagonal immediately reveal files that would otherwise have been read in the wrong temporal order.
`verbose`	Logical. Print progress messages and elapsed time.
`files`	Character vector of file paths, in the exact order you want them read as layers – no sorting of any kind is ever applied. Use instead of `dir` whenever automatic detection from file names is not reliable enough, which in practice means: whenever the series is not simply annual. The same numeric shape in a file name can mean genuinely different things across real datasets (e.g. `"19820102"` is 2 January in some daily products, but the second half of January in PKU-GIMMS' own semimonthly convention) – automatic detection cannot resolve that safely, only declaring it explicitly can. Exactly one of `dir` or `files` must be supplied.
`time`	Optional. A `Date`, `POSIXct`, or numeric vector, one value per layer, in the same order as `files`. Must be strictly increasing with no missing or duplicated values. The supplied file order and time vector are treated as authoritative – checked for internal consistency (right length, valid dates, correct order), not that you paired the right date with the right file; that correspondence remains your responsibility. Mutually exclusive with `cycle_type`. Requires `files`.
`cycle_type`	Optional. One of `"annual"`, `"monthly"`, `"16-day"`, `"semimonthly"`, `"10-day"`, `"8-day"`, `"weekly"`, or `"daily"` – generates real calendar dates instead of you supplying `time` yourself, for series that genuinely follow one of these conventions (see "Supported cycle types" above for the generative rule and real-product examples behind each). If your series does not follow one of these exact conventions, supply `time` explicitly instead. Requires `files` and `start`.

`start` Required with `cycle_type`. A single `Date` marking the beginning of the first period. Must fall on a boundary valid for the declared `cycle_type` (day 1 for `"monthly"`; day 1 or 16 for `"semimonthly"`; day 1, 11 or 21 for `"10-day"`; year-day 1, 17, 33, ..., 353 for `"16-day"`; year-day 1, 9, 17, ..., 361 for `"8-day"`; year-day 1, 8, ..., 358 for `"weekly"`; 1 January for `"annual"`; any date for `"daily"`) – this checks that your own declaration is internally consistent, not that it matches your real product. If your product genuinely starts elsewhere, it does not follow this `cycle_type` convention – supply `files` + `time` instead.

`end` Optional, only with `cycle_type`. A single `Date`, the final inclusive end of the last declared period. If omitted, the number of layers found in `files` determines how many dates are generated from `start` onward. If supplied, the calendar expected between `start` and `end` is built first, and the number of layers found must match that expected count exactly – this is what detects an incomplete series (e.g. a declared full year missing one month's file), which omitting `end` cannot catch on its own.

`time_anchor` One of `"start"`, `"centre"` (default), or `"end"`. Only applicable with `cycle_type` = `"monthly"`, `"semimonthly"`, `"10-day"`, `"16-day"`, `"8-day"` or `"weekly"`. Annual layers are always dated 1 January; daily layers use their own date. Do not supply `time_anchor` for these two conventions. Controls which point within each period's own date range is assigned as that layer's date – irrelevant for MK/CMK (rank-based), but affects the numeric time values OLS/MMK use directly. For a period with an even number of days, `"centre"` deterministically picks the earlier of the two central days.

Details

Works with raster formats readable by `terra::rast()`, subject to the GDAL drivers available in the user's `terra` installation. The default pattern covers common GeoTIFF, NetCDF, native raster, ERDAS Imagine, virtual raster and ASCII-grid extensions; other supported formats can be selected through `pattern`. Each file must represent one time step. For a single multi-temporal NetCDF file instead, use `read_netcdf_stack()`.

Why this matters more than it looks: a trend analysis assumes that successive raster layers represent the true chronological sequence. If the temporal order is wrong, every subsequent statistical result becomes invalid even though the analysis completes without any error – there is nothing in a Mann-Kendall test's own output that could reveal a shuffled input series. Unlike generic file readers, this function deliberately refuses to proceed when the temporal order cannot be established unambiguously, rather than silently reverting to alphabetical file names. This same philosophy – surface a silent risk rather than let it pass unnoticed – recurs throughout this package: multiple-testing correction (`fdr_correction()`), the spatial-autocorrelation diagnostic behind it (`spatial_autocorrelation()`), and the monotonic-trend-only assumption checked informally in "Monotonic trends only" sections elsewhere (`trend_test()`, `slope_estimator()`) are the same instinct applied to different risks.

This function does not use `list.files()`'s own ordering, does not attempt to parse arbitrary or ambiguous date formats, and does not guess when no candidate pattern extracts a unique number from every file – it stops instead, on the view that it is better to stop than to silently analyse the wrong chronology.

Function type: Data import function – builds the ordered `SpatRaster` expected by the analytical functions. It performs no statistical inference.

Value

A `terra::SpatRaster` with one layer per time step, ordered chronologically – one layer per input file for ordinary single-layer formats (the typical case), but a single input file can contribute more than one layer for multi-layer formats (e.g. a NetCDF file with several time steps of its own; `files` and the explicit modes fully support this, tracking which layers came from which file for the verbose order-check table). Layer names are taken from the (de-duplicated) file names, and temporal order is stored as proper time metadata (`terra::time(result)`) rather than discarded after the verification step above – used, for instance, as the default `t` in `inspect_ts_cell()`, and by any future function in this package requiring explicit time coordinates.

Typical use

```
folder with one raster file per time step
    |
read_ordered_stack()
    |
chronologically ordered raster time series
    |
compute_anomalies() if seasonal, then a trend workflow
```

Methodological details

Temporal ordering

Ordering is derived from explicit numeric labels in file names, never from alphabetical order. Ambiguous, duplicated, or incomplete labels stop the import rather than trigger an undocumented fallback.

Explicit declaration (`files`, `time`, `cycle_type`)

Automatic detection above only extracts one ordering number per file name, which is reliable for genuinely annual series but not for finer cadences: the same numeric shape in a file name can mean genuinely different things across real datasets – `"19820102"` is 2 January in some daily products, but the second half of January under PKU-GIMMS NDVI's own semimonthly convention. `files` (an explicit, already correctly ordered vector) sidesteps this by never interpreting file names at all; combined with `time` (fully explicit dates) or `cycle_type` (a named calendar convention) it is recommended whenever the series is not simply annual.

Supported cycle types

Eight unambiguous calendar conventions, all built with genuine calendar arithmetic (leap years included) rather than naive interval division. Fixed compositing intervals that do not follow one of these exact conventions (e.g. a genuinely continuous 8-day or 16-day interval) are out of scope for `cycle_type` – supply `time` explicitly instead.

`"annual"` One date per year, 1 January. 1 sample/year. E.g. the bundled `example_data("vhp_ndvi")` dataset itself.

`"monthly"` One period per calendar month, beginning on the 1st. 12 samples/year. E.g. CRU TS, TerraClimate, ERA5 monthly means.

`"16-day"` 23 fixed periods per year, resetting every 1 January (never continuing across a year boundary), starting on year-day 1, 17, 33, ..., 353 – the last period of the year is shorter (13 or

14 days) to fit within the calendar year. Matches MODIS' own 16-day compositing convention (e.g. MOD13Q1), verified directly against real product file names.

`"semimonthly"` Two periods per calendar month, beginning on the 1st and the 16th. 24 samples/year. Matches PKU-GIMMS NDVI's own "half-month" convention – not the same cadence as a continuous 14-day interval.

`"10-day"` Three calendar periods per month, starting on the 1st, 11th and 21st (the last one running to the end of the month, so its own length varies: 8 to 11 days). 36 samples/year. The standard "dekad" convention, e.g. SPOT-VEGETATION and several FEWS NET agricultural products – not a continuous 10-day interval, which would not align with month boundaries and would give a different total (37, not 36).

`"8-day"` 46 fixed periods per year, resetting every 1 January, starting on year-day 1, 9, 17, ..., 361 – the last period of the year is shorter, capturing the remainder. Matches MODIS' own 8-day compositing convention (e.g. MCD15A2H, the LAI/FPAR product).

`"weekly"` 52 fixed, complete 7-day periods per year (Earth Trends Modeler's own number), starting on year-day 1, 8, ..., 358 – unlike `"8-day"`/`"16-day"`, the final one or two days of the year belong to no period and are skipped straight into next year's first period, rather than forming a shorter final period.

`"daily"` One date per real calendar day, including 29 February in leap years. 365 or 366 samples/year – the only one of these eight where the yearly total itself changes with leap years (the other seven keep a fixed count per year; only the exact date of late-period boundaries shifts). E.g. ERA5, CHIRPS daily precipitation.

`"weekly"`'s own remainder-discarding convention (unlike the other fixed-interval types here) is not this function's own inconsistency – it follows Earth Trends Modeler's own table exactly for each named type individually, and that table is not internally consistent on this specific point across its own listed conventions.

Monthly or seasonal data

This function only orders layers chronologically – it does not know or care whether the data has a seasonal cycle (e.g. monthly values with an annual signal). If it does, deseasonalise with `compute_anomalies()` *before* passing the result to `trend_test()`, `slope_estimator()`, or `workflow_tst()`, all of which assume a monotonic trend, not a periodic one. Ordering and deseasonalisation solve different problems.

Limitations

Files must share compatible raster geometry and each file must represent one time step. Automatic detection deliberately does not attempt to infer arbitrary calendar formats that are not captured by the supplied or candidate regular expressions – use `files` with `time` or `cycle_type` for those cases instead of expecting automatic detection to guess correctly.

Quality assurance

Tests cover year and year-month filename parsing, chronological ordering, duplicate and ambiguous labels, mixed geometries, variable selection, preserved `terra` geometry/time metadata, and informative failures for empty or invalid inputs. See `?sptrends` for the package-wide release-check protocol.

See Also

Other Data import functions: `read_netcdf_stack()`

Examples

```
# The bundled dataset contains one real NDVI GeoTIFF per year.
s <- read_ordered_stack(example_data("vhp_ndvi"))
terra::nlyr(s)
terra::time(s, "years")
terra::plot(s[[1]], main = "NDVI, first year")
```

`simulation_design` *Build a factorial design of simulation scenarios*

Description

Creates named argument lists for `sim_trend_stack()` by crossing temporal, spatial, signal and noise conditions. It separates experimental design from data generation so the complete scenario grid can be inspected and retained.

Usage

```
simulation_design(..., constants = list(), prefix = "scenario", verbose = TRUE)
```

Arguments

`...`	Named vectors or lists of factor levels to cross.
`constants`	Named list of arguments shared by every scenario.
`prefix`	Character prefix used for generated scenario names.
`verbose`	Logical. If `TRUE`, reports progress, elapsed time and the estimated time remaining while scenarios are assembled.

Details

Function type: Benchmarking function – defines simulation scenarios; it does not generate data or perform inference.

Value

A named list of argument lists suitable for the `scenarios` argument of `benchmark_methods()`, with classes `"sptrends_simulation_design"` and `"sptrends"` for unified printing, summaries, and plotting.

Typical use

Define the factors that should vary, add shared settings through `constants`, and pass the returned list to `benchmark_methods()`.

Methodological details

Experimental design

Every combination is retained. This makes comparisons across spatial and temporal dependence explicit and prevents methods from being evaluated on accidentally different scenario sets. Values that must remain grouped, such as a two-number `signal_size`, should be wrapped in a list.

Computational considerations

This function only constructs argument lists. Memory use grows with the product of the numbers of supplied factor levels; data generation remains deferred to `benchmark_methods()`.

Limitations

A full factorial design can become unnecessarily large. Users should vary scientifically relevant factors and keep fixed settings in `constants`.

Quality assurance

Tests verify factorial completeness, deterministic ordering, grouped values, validation failures, metadata retention and S3 presentation. The complete external simulation-cycle validation passed all 33 prespecified controls; see `inst/validation/` for the retained protocol and results.

See Also

`sim_trend_stack()`, `benchmark_methods()`

Other validation functions: `benchmark_methods()`, `benchmark_summary()`, `compare_detections()`, `plot_detection_comparison()`

Examples

```
design <- simulation_design(
  spatial_model = c("independent", "exponential"),
  spatial_rho = c(0.3, 0.7), ar1 = c(0, 0.5),
  trend_strength = c(0, 0.05),
  constants = list(nrow = 20, ncol = 20, n_time = 20,
                   constant_block = FALSE), verbose = FALSE)
length(design)
```

`sim_trend_stack` *Generate a synthetic gridded time series with known true trends*

Description

Generates benchmark datasets with a known ground truth, for evaluating spatiotemporal trend-detection methods. Concretely: a synthetic gridded time series, with the true, known slope at every cell kept alongside the data itself, for runnable `@examples`, vignettes, and unit tests, rather than as a realistic environmental simulation. Because the ground truth is known exactly (something no real dataset offers), this is the tool behind sptrends' own worked examples of what a trend-detection method actually gets right or wrong: compare `true_slope` against a fitted method's estimated slope, and against which cells it calls significant, to see estimation error and Type I/Type II error

directly instead of having to take a method's output on faith – the mechanism that lets power, false discovery rate, sensitivity, specificity, and robustness be demonstrated *objectively* for any method, not just described in prose.

Usage

```
sim_trend_stack(
  nrow = 15,
  ncol = 15,
  n_time = 10,
  trend_strength = 0.15,
 trend_shape = c("radial", "gradient", "block", "square", "rectangle", "ellipse",
    "custom"),
  trend_fraction = 0.9,
  ar1 = 0.3,
  noise_sd = 1,
  noise_dist = c("gaussian", "t"),
  t_df = 4,
  smooth_radius = 1L,
  spatial_rho = 1,
 spatial_model = c("legacy", "independent", "gaussian", "exponential", "matern"),
  spatial_range = 1,
  spatial_smoothness = 0.5,
  signal_size = NULL,
  signal_location = c("centre", "random"),
  signal_angle = 0,
  signal_axis_ratio = 1,
  custom_mask = NULL,
  constant_block = TRUE,
  break_type = c("none", "mean", "slope"),
  break_time = NULL,
  break_fraction = 0.3,
  break_magnitude = 2,
  seed = NULL,
  verbose = TRUE
)
```

Arguments

nrow, ncol	Integer. Raster dimensions in cells.
n_time	Integer. Number of time steps (layers).
trend_strength	Numeric. Maximum magnitude of the true slope (see "How the trend is generated – trend field" above for how it varies spatially according to trend_shape).
trend_shape	Spatial pattern of the true slope field: "radial", "gradient", "block", "square", "rectangle", "ellipse", or "custom"; see "How the trend is generated – trend field" above.
trend_fraction	Numeric in [0, 1]. Proportion of cells that keep a non-zero true slope; the rest are forced to exactly 0. Default 0.9. Set to 0 for a complete null field (no trend

anywhere), or 1 for a trend at every cell trend_shape assigns one to.

ar1 Numeric in (-1, 1). AR(1) autocorrelation coefficient of the noise term across time, applied independently within each cell.

noise_sd Numeric > 0. Standard deviation of the noise term – the main control over how easy or hard the true trend is to detect against the background noise (signal-to-noise ratio). Default 1.

noise_dist "gaussian" (default) or "t" – the distribution of the noise term, always rescaled to the standard deviation noise_sd regardless of shape. This matters for comparing methods: Mann-Kendall-based tests (and Theil-Sen) are rank-based and robust to heavy-tailed, outlier-prone noise, while a parametric method like OLS is the more *efficient* choice specifically when noise is Gaussian – the classic case for preferring a non-parametric method only shows up with non-Gaussian noise. "t" draws from a Student's t distribution with t_df degrees of freedom instead, giving heavier tails (more extreme outliers) at the same standard deviation.

t_df Numeric > 2. Degrees of freedom for the noise distribution when noise_dist = "t"; ignored otherwise. Lower values give heavier tails (more frequent/extreme outliers); values just above 2 are extreme, values above 30 are practically indistinguishable from Gaussian. Must exceed 2 for the distribution to have finite variance (required to rescale it to noise_sd).

smooth_radius Integer >= 0. Radius, in cells, of the focal mean window used to introduce spatial autocorrelation into the noise (see "How spatial autocorrelation is generated" above). 0 disables this: cells get independent noise with no spatial structure beyond the trend field itself. 1 (default) smooths over a 3x3 window; larger values give smoother, more strongly autocorrelated fields, at the cost of more computation.

spatial_rho Numeric in [0, 1]. For the legacy model, the weight assigned to focal-smoothed noise; this is not a target Moran's I. For Gaussian and exponential covariance models, the adjacent-cell correlation of the latent Gaussian field. It must be positive for these formal models. It is ignored by the independent and Matérn models.

spatial_model Spatial noise model. "legacy" preserves the focal smoother used by earlier versions. "independent", "gaussian", "exponential", and "matern" use a formal stationary covariance model simulated by circulant embedding and FFT.

spatial_range Positive covariance scale for the Matérn model.

spatial_smoothness
Positive Matérn smoothness parameter.

signal_size One number or a two-number vector giving the height and width, in cells, of block, square, rectangular, or elliptical signal regions. Defaults to half the raster in each dimension for "block" (its historical default), or a third of the raster for "square", "rectangle" and "ellipse".

signal_location
"centre" or "random"; location of an exact geometric signal region ("block", "square", "rectangle", "ellipse" or "custom"). Has no effect on "radial" or "gradient", which are not exact geometric regions.

signal_angle Rotation of a geometric signal region in degrees, or "random" to draw a new orientation in each realisation.

signal_axis_ratio
: Positive width multiplier for an ellipse.

custom_mask
: Logical/numeric matrix, vector, or single-layer SpatRaster defining the signal when trend_shape = "custom".

constant_block
: Logical. If TRUE, sets a small block of cells to a constant value (zero temporal variance) to exercise degenerate-case handling. Independent of trend_shape = "block" above, which is about the *trend* pattern, not a zero-variance edge case.

break_type
: "none" (default): no structural break, this function's original behaviour, unchanged. "mean": a step change in level at break_time – ground truth for a change-point method like Pettitt's test. "slope": a change in slope at break_time (continuous in level, a kink not a jump) – ground truth for distinguishing "did the trend change partway through" from "is there a trend at all". A break and a monotonic trend (trend_strength/trend_fraction above) compose independently on any given cell – a cell can have a trend only, a break only, both, or neither; the two are not mutually exclusive, nor does one replace the other.

break_time
: Integer or NULL. Time step *after* which the break takes effect (so break_time = 5 means steps 1-5 are unaffected, the break shows in step 6 onward). NULL (default): the middle of the series, round(n_time / 2). Must be in [1, n_time - 1] so there is at least one time step on each side. Ignored when break_type = "none".

break_fraction
: Fraction of the same spatially coherent blocks used for trend_fraction above that get a break, independently of which blocks (if any) got a trend. Default 0.3. Ignored when break_type = "none".

break_magnitude
: Numeric. For break_type = "mean": the size of the step added to the level from break_time onward. For break_type = "slope": the size of the *additional* slope (per time step) that kicks in from break_time onward, on top of whatever slope (possibly zero) that cell already had. Default 2. Ignored when break_type = "none".

seed
: Integer or NULL. Random seed for reproducibility.

verbose
: Logical. If TRUE, report simulation progress, elapsed duration and estimated time remaining. Set to FALSE inside large Monte Carlo experiments when benchmark_methods() provides the outer progress display.

Details

Function type: Benchmarking function – generates known-truth data for compare_detections() and external methods. It is not a component of the inferential workflows themselves.

Value

An object of class "sptrends_simulation" and "sptrends", supporting print(), summary(), and plot(), with:

series
: A terra::SpatRaster with n_time layers named "t1", "t2", ...; each layer represents one time step. Pass this to workflow_tst() or the individual trend functions.

true_slope	A single-layer terra::SpatRaster: the exact true slope used to generate each cell's series (0 where trend_fraction assigned no trend). Compare directly against a fitted slope_estimator() to see estimation error, or against direction_map() to see which cells a method calls significant vs. which cells truly have a trend.
true_signal	A binary SpatRaster equal to 1 where the exact true slope is non-zero.
true_direction	A SpatRaster containing -1, 0, or 1 for the known trend direction.
true_break	A single-layer terra::SpatRaster, 1 where a structural break was actually applied (per break_fraction), 0 elsewhere – ground truth for evaluating a change-point detection method the same way true_slope already lets you evaluate a trend test, via compare_detections(). Always present, even when break_type = "none" (all 0 in that case), so downstream code doesn't need to branch on whether the field exists.
break_time	The actual time step used as the break point (see break_time above), or NULL if break_type = "none" or break_fraction = 0 (no cell actually got a break).
parameters	The data-generating parameters retained with the realisation for reproducible benchmarking.
diagnostics	Simulation metadata, including the covariance model and target unit-lag correlation where applicable.

Typical use

Single-run benchmarking:

```
sim_trend_stack()
    |
run one or more methods on sim$series
(trend_test(), workflow_tst(), workflow_rta(), or your own)
    |
compare_detections()
```

For replicated benchmarking, repeat simulation and detection across seeds, collect the results, and aggregate them:

```
list of detections + list of ground truths
    |
compare_detections(replicates = TRUE)
```

See compare_detections()'s examples for both routes worked through in full. One call creates one replicate; Monte Carlo repetition remains outside this function so the same simulator can benchmark sptrends methods, future procedures, or external software. Use ar1 for temporal dependence and a formal spatial_model with its covariance parameters for spatial dependence.

Methodological details

Scope and limitations

This function is intended for methodological benchmarking rather than realistic environmental simulation. Its objective is to generate controlled datasets with a known ground truth, not to reproduce

the statistical properties of a particular environmental variable such as NDVI, temperature or precipitation. Formal Gaussian, exponential and Matérn covariance models and exact geometric signal regions provide controlled temporal and spatial structures for evaluating detection methods; they are not a calibrated environmental process model.

How the trend is generated – trend field

`trend_shape` sets the *pattern* of the true slope field:

- `"radial"` (default): slope is `trend_strength` at the centre cell and decays smoothly (exponentially) with distance from it, reaching close to zero towards the edges.
- `"gradient"`: slope varies linearly from `-trend_strength` at the left edge to `+trend_strength` at the right edge – decreasing on one side, increasing on the other, with no radial symmetry.
- `"block"`: `trend_strength` everywhere inside a centred square block covering about half the raster's area, `0` everywhere outside it – a sharp-edged region of trend against a flat background, rather than a smooth spatial gradient.

How the trend is generated – trend masking

`trend_fraction` then decides *how much of the raster actually has a trend at all*: the raster is partitioned into coarse, contiguous spatial blocks (not individual cells), a random `trend_fraction` proportion of *blocks* keep the slope values `trend_shape` assigned them, and the rest are forced to an exact true slope of `0`, regardless of shape. Acting on whole spatial patches rather than scattering individual cells at random matters here: `trend_test()`'s whole rationale is borrowing statistical strength from a cell's neighbours, which only helps when neighbouring cells are plausibly trending together – a cell-by-cell (salt-and-pepper) random mask would quietly defeat that assumption and penalise CMK for no good reason. `trend_fraction = 0` gives a complete null field (every cell's true slope is exactly `0` – there is no trend anywhere to find, the sharpest way to check a method's false-positive rate). `trend_fraction = 1` gives a trend everywhere `trend_shape` says there should be one. Among the blocks that do keep a trend, a random 10% have their sign flipped as a whole block, so the field is not purely "everything increases" even within the trending region, while still keeping each flipped patch spatially coherent.

How spatial autocorrelation is generated – the idea

For formal benchmarking, choose `spatial_model = "gaussian"`, `"exponential"`, or `"matern"`. Independent stationary fields are drawn at each time step by circulant embedding and FFT. If an admissible embedding is unavailable, modest grids use an exact covariance eigendecomposition. For Gaussian and exponential models, `spatial_rho` is the theoretical correlation between horizontally or vertically adjacent cells. For Matérn fields, `spatial_range` and `spatial_smoothness` define the covariance. The `"independent"` model provides the exact zero-dependence baseline.

`spatial_model = "legacy"` retains the earlier didactic smoother. Independent noise, with standard deviation `noise_sd`, is drawn per cell and per time step (with AR(1) correlation across time, controlled by `ar1`, but no spatial structure yet). If `smooth_radius > 0` and `spatial_rho > 0`, each time step's noise layer is smoothed with a focal mean filter. `smooth_radius` controls spatial scale, whereas `spatial_rho` controls how strongly the original and smoothed fields are blended. This is a pragmatic moving-average smoother, not a fitted CAR/SAR model or a Gaussian random field with a specified covariance function – it is meant to give `spatial_autocorrelation()` something genuine to detect in examples, not to match any particular real-world spatial process (see "What this simulator is not" above).

Computational considerations

How spatial autocorrelation is generated – implementation

The smoothing filter is a square window of side `2 * smooth_radius + 1` cells (via `terra::focal()`). The smoothed and blended noise are rescaled so spatial controls do not intentionally change `noise_sd`. `spatial_rho = 0` retains the independent field; `spatial_rho = 1` retains the fully smoothed field and reproduces the behaviour used before this parameter was added. Intermediate values are blend weights, not target Moran's I values. All `n_time` layers are smoothed in a single batched `terra::focal()` call on the whole multi-layer noise stack, rather than one call per layer – `terra::focal()` already smooths each layer of a multi-layer input independently (there is no cross-layer mixing), so this produces identical output while avoiding `n_time - 1` redundant round trips through `terra`'s internals. Matters mainly for a large `n_time`; for the modest-sized rasters this function is typically used to build (examples, tests), both versions are effectively instant either way. `terra::focal()` itself additionally requires the focal window (side `2 * smooth_radius + 1`) to be no more than twice the raster's own size in each direction – a small raster with a proportionally large `smooth_radius` (most sharply, any raster with `nrow` or `ncol` of 1, since even the smallest window, `smooth_radius = 1`, already has side 3) will not fit this. Rather than letting `terra::focal()`'s own internal error propagate up, this is checked beforehand and, if violated, a warning is issued and unsmoothed noise is used for that call instead.

Statistical assumptions and interpretation

Noise distribution and comparing methods

`noise_dist` controls the *shape* of the noise, independently of its standard deviation (`noise_sd`) or its temporal (`ar1`) and spatial (`smooth_radius`) correlation. This matters specifically for comparing a rank-based method (Mann-Kendall, Contextual Mann-Kendall, Theil-Sen) against a parametric one (e.g. ordinary least squares): under Gaussian noise (`noise_dist = "gaussian"`, the default), OLS is the more *efficient* estimator, so a comparison run only under Gaussian noise will not show the robustness advantage rank-based methods are typically chosen for. Set `noise_dist = "t"` with a low `t_df` (e.g. `3` or `4`) to generate heavy-tailed, outlier-prone noise instead, at the same `noise_sd` – this is the condition under which rank-based methods are expected to hold up better than OLS.

The simulation parameters control a synthetic data-generating process; they are not fitted environmental parameters. Under `spatial_model = "legacy"`, `spatial_rho` is a blend weight rather than a target correlation. Under Gaussian or exponential covariance with Gaussian noise, it is the target correlation between horizontally or vertically adjacent cells. With Student-t noise, it controls the latent Gaussian dependence used by the copula, so realised Pearson correlation need not equal it exactly. One call produces one realisation; use `benchmark_methods()` for a Monte Carlo experiment.

Quality assurance

Tests verify known slope and break fields, reproducibility, null and complete-signal cases, noise scale and distribution, degenerate grid sizes, focal-window safeguards, temporal/spatial controls and direct compatibility with `compare_detections()`. The `spatial_rho` tests specifically protect both endpoint semantics and the pre-0.89 default. Independent validation additionally compared the analytical Matérn correlation with `fields::Matern()`, evaluated 1,000 Gaussian fields per spatial model, and checked temporal AR(1), marginal distributions, exact truth fields, detection metrics and reproducibility. All 33 external validation controls passed in the recorded 0.96.3 full run. The retained script and numerical summaries are described under `inst/validation`. See `?sptrends` for the package-wide release-check protocol.

References

Dietrich, C.R. and Newsam, G.N. (1997). Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. *SIAM Journal on Scientific Computing*, 18(4), 1088-1107. doi:10.1137/S1064827592240555

See Also

Other example data functions: `example_data()`

Examples

```
# A small synthetic dataset with a known trend -- 8 time steps on a
# 12x12 grid. terra::global(..., "range") shows the true slope's
# minimum and maximum across all cells.
sim <- sim_trend_stack(nrow = 12, ncol = 12, n_time = 8, seed = 42)
terra::nlyr(sim$series)
terra::global(sim$true_slope, "range", na.rm = TRUE)

# A complete null field: every true slope is exactly zero.
sim_null <- sim_trend_stack(nrow = 12, ncol = 12, n_time = 8,
                            trend_fraction = 0, seed = 1)
terra::global(sim_null$true_slope, "range", na.rm = TRUE)

# Ground truth for a change-point method (e.g. Pettitt's test): a
# mean-shift break in 30% of the map, no monotonic trend at all.
sim_break <- sim_trend_stack(nrow = 12, ncol = 12, n_time = 10,
                             trend_fraction = 0, break_type = "mean",
                             break_fraction = 0.3, seed = 2)
sim_break$break_time
terra::global(sim_break$true_break, "sum", na.rm = TRUE)

# Spatial scale and intensity are separate: smooth_radius defines the
# focal scale, while spatial_rho blends independent and smoothed noise.
# Compare Moran's I for one time step at the two intensity extremes.

r0 <- sim_trend_stack(nrow = 20, ncol = 20, n_time = 1,
                      smooth_radius = 3, spatial_rho = 0,
                      seed = 1)$series[[1]]
r1 <- sim_trend_stack(nrow = 20, ncol = 20, n_time = 1,
                      smooth_radius = 3, spatial_rho = 1,
                      seed = 1)$series[[1]]
spatial_autocorrelation(r0, nperm = 99, seed = 1, verbose = FALSE,
                        report = FALSE)$statistic
spatial_autocorrelation(r1, nperm = 99, seed = 1, verbose = FALSE,
                        report = FALSE)$statistic


# Heavy-tailed (outlier-prone) noise instead of Gaussian, at the same
# standard deviation -- for comparing a rank-based method against OLS.
sim_heavy <- sim_trend_stack(nrow = 10, ncol = 10, n_time = 10,
                             noise_dist = "t", t_df = 3, seed = 1)
terra::nlyr(sim_heavy$series)
```

slope_estimator *Slope estimators for raster time series*

Description

Estimates the magnitude of a monotonic trend, independently of its statistical significance. `trend_test()` primarily tells you *whether* there is evidence of a trend and its direction; although its `"OLS"` branch necessarily returns a fitted coefficient, this function provides the dedicated and method-independent estimation of the *rate* of change, per cell, using one of three published estimators (see the `method` argument below).

Usage

```
slope_estimator(
  x,
  method = c("TS", "OLS", "RM"),
  t,
  max_pairs = 1e+05,
  seed = NULL,
  n_cores = 1,
  smooth_neighbourhood = FALSE,
  report = TRUE,
  verbose = TRUE,
  shared_cluster = NULL
)
```

Arguments

`x`	A `terra::SpatRaster`; each layer is one time step, in increasing chronological order.
`method`	Which slope estimator to use. `"TS"` (default): the classic Theil-Sen estimator – the median of all pairwise slopes per cell, in the same units as `x` per unit of `t`. Robust (outlier-resistant) up to a ~29% breakdown point, at the cost of the performance profile described in "Computational considerations" below. This package's own default – see "Methodological details" below for why. `"OLS"`: ordinary least squares – the closed-form linear regression slope, `cov(t, y) / var(t)` per cell, computed directly with a single vectorised matrix operation (no pairwise sampling, no iteration – `max_pairs`, `seed`, and `n_cores` below are all ignored). Faster, and the more familiar choice to most users and to readers used to classical linear regression. Worth choosing over the default specifically when: speed matters more than robustness (very large rasters or very long series – see "Computational considerations" below); the residuals are genuinely expected to be close to normally distributed with no substantial outlier

risk (in which case OLS is also the more statistically efficient of the two, with lower variance for the same data); or the result needs to be directly comparable to other analyses reported as classical linear-regression slopes, rather than a median-of-pairwise-slopes estimate.

`"RM"`: Siegel's (1982) repeated median estimator – for each cell, the median slope from every observation to every other observation is computed first (one median per observation), then the median of those per-observation medians is taken as the final slope. Like `"TS"`, median-based and highly robust – but with a 50% breakdown point rather than Theil-Sen's own ~29%, at a real computational cost: this is a naive O(n^2) port (see "Implementation notes" below), not the quasilinear-time algorithm the reference implementation this was ported from and verified against, `robslopes::RepeatedMedian()`, itself uses. Choose this over `"TS"` specifically when a dataset is suspected to have enough outliers or leverage points that Theil-Sen's own lower breakdown point might not fully resist them; `max_pairs`, `seed` and `n_cores` below are not used by this method (unlike `"TS"`, it does not sample pairs, and is not currently parallelised).

`t` Numeric vector of time points, one per layer. Defaults to `1:nlyr(x)`. Irregular spacing is supported, but values must be finite, unique and strictly increasing.

`max_pairs` Integer or `Inf`. Only used when `method = "TS"`. If the number of possible pairs (`n*(n-1)/2`) exceeds this, a random sample of `max_pairs` pairs is used per cell (see "Computational considerations" below). Default `100000`; set to `Inf` for the exact computation regardless of series length.

`seed` Integer or `NULL`. Only used when `method = "TS"` and subsampling actually happens (random seed for the pair sampling).

`n_cores` Integer. Only used when `method = "TS"`. Number of cores for the cell loop. `1` (default): sequential. `> 1`: uses a `parallel::makeCluster()` PSOCK cluster.

`smooth_neighbourhood`
Logical. Default `FALSE` – the estimated slope (whichever `method` was used) is left as computed, per cell independently. If `TRUE`, each cell's slope is replaced by the **median** of the already-estimated slopes in its queen 3x3 neighbourhood (itself and its 8 surrounding cells) – a queen-neighbourhood median filter applied to the per-cell slopes already computed, not a different way of estimating any individual cell's own slope. See the dedicated section below; read it before setting this to `TRUE`.

`report` Logical. If `TRUE` (default), automatically print the summary (`slope_summary()`) and draw the map (`slope_map()`) once the slope finishes computing.

`verbose` Logical. Print progress messages and elapsed time.

`shared_cluster` Advanced; most users never need this directly. An already-running `parallel::makeCluster()` PSOCK cluster (e.g. from `workflow_tst()`'s own `n_cores`) to reuse instead of building a new one from `n_cores` above. When supplied, `n_cores` is ignored – the shared cluster's own size was already decided by whoever built it. `NULL` (default): builds and tears down its own cluster from `n_cores`, exactly as before this argument existed.

Details

Function type: Companion function to `trend_test()` – one of the three quantities a standard

trend analysis normally reports (alongside significance and multiple-testing correction), not a mere supporting utility, though not one of `trend_test()` or `fdr_correction()`'s own inferential building blocks either.

Value

Returns an object of class `c("slope", "sptrends")`: a list with

`slope`	A single-layer `terra::SpatRaster`, named `"theilsen_slope"`, `"ols_slope"` or `"rm_slope"` depending on `method` (units: `x` per unit of `t`, e.g. per year if `t` is in years).
`intercept`	Only for `method = "RM"`: a single-layer `terra::SpatRaster` (named `"rm_intercept"`, `NA` for invalid cells), the repeated median estimator's own intercept – see "Implementation notes" below for how it is computed. Smoothed the same way `slope` itself is when `smooth_neighbourhood = TRUE`, for the same reason: a `SpatRaster`, matching `slope`'s own representation, rather than a plain numeric vector.
`method`	Character: which `method` produced this result (`"TS"`, `"OLS"` or `"RM"`).
`smoothed`	Logical: whether post-estimation neighbourhood smoothing (`smooth_neighbourhood = TRUE`) was applied to `slope` after estimation, not whether smoothing was itself part of how `slope` was computed.

Use `print()`/`summary()`/`plot()` – see `print.sptrends()`, `summary.sptrends()`, and `plot.sptrends()`.

Typical use

```
raster time series
    |
slope_estimator()
    |
change per time unit (`result$slope`)
```

Run this beside `trend_test()` on the same analytical series: the slope measures magnitude, whereas the trend test assesses evidence. If the test uses a prewhitened series, choose deliberately whether the scientific estimand is the slope of that transformed series or of the original series.

Methodological details

Why estimate the slope separately from the trend test?

Statistical significance and effect size answer different questions. A statistically significant trend may be negligible in magnitude, whereas a large estimated slope may fail to reach significance when uncertainty is high. For this reason, sptrends separates trend testing (`trend_test()`) from slope estimation (this function), allowing each quantity to be interpreted independently – neither substitutes for the other.

Methods and method selection

- **Original publication**: Theil (1950), later generalised by Sen (1968) into the median-of-pairwise-slopes form used here (linking it to Kendall's tau).

- **Main references**: Theil (1950) and Sen (1968), the two papers this estimator is directly named after. Full citations in "References" below.
- **Typical applications**: estimating the magnitude of a monotonic trend when the data may contain outliers or heavy-tailed noise – robust up to a ~29% breakdown point, unlike ordinary least squares.
- **Theil-Sen vs. OLS**: both quantify a linear rate of change, but they define and estimate that rate differently, under different statistical assumptions – Theil-Sen makes no distributional assumption about the noise and tolerates a substantial fraction of outliers before breaking down; OLS is highly sensitive to influential observations and heavy-tailed errors. Normality is not required to calculate the OLS slope itself, although conventional small-sample inference for it (its own standard errors and significance test) relies on additional distributional assumptions the point estimate does not need. See the `method` argument below for when each is the more defensible choice.

Implementation notes (`method = "RM"` specifically)

`"RM"` implements Siegel's (1982) repeated median estimator using the upper-median convention `robslopes::RepeatedMedian()` itself uses: for a vector of length `k`, the upper median is `sort(v)[floor((k + 2) / 2)]`; this differs from `stats::median()`, which averages the two central observations when `k` is even.

This implementation computes the estimator directly in `O(n^2)` time. It therefore reproduces the estimator itself, but not the faster, quasilinear-time algorithm (Matousek, Mount and Netanyahu, 1998) `robslopes` uses internally in C++.

The intercept follows the "direct" (also called "separate") repeated-median convention, applying the same nested upper-median operation to the pairwise intercepts, rather than the simpler "hierarchical" convention (`median(y - slope * t)`, using the already-computed overall slope) – both are legitimate conventions Siegel (1982) himself describes, but only the former matches `robslopes::RepeatedMedian()`'s own intercept output.

Statistical assumptions and interpretation

All methods require finite, unique and strictly increasing time values. Duplicates would produce undefined pairwise slopes for the robust estimators, while unordered values would no longer describe the chronological layer order; both are rejected before calculation. Valid cells must have a complete time series. Each estimator summarises change as one overall linear rate: `"OLS"` targets the least-squares slope, `"TS"` the median pairwise slope, and `"RM"` the repeated-median slope. The robust estimators do not require normally distributed errors; OLS can still be calculated without normality, but is more sensitive to outliers and influential observations.

Linear long-term change and seasonality

These estimators quantify an overall linear rate of change; they do not model seasonal cycles or nonlinear temporal structure. For strongly seasonal series, consider estimating the slope from anomalies (see `compute_anomalies()`) or using an appropriate seasonal model instead.

Computational considerations

Performance – read this before using long time series

The estimators have substantially different computational profiles. OLS is the fastest because it uses one vectorised closed-form calculation. Exact Theil-Sen is slower because it evaluates pairwise slopes, but it offers a strong practical balance between robustness, interpretation and computational cost and is therefore the recommended general-purpose estimator. The directly implemented

repeated median is usually much slower than both because it calculates nested medians for every observation in every cell; reserve RM for cases in which its higher breakdown point is scientifically needed and its additional cost is acceptable.

The exact estimator needs **every** pairwise slope, `n*(n-1)/2` of them per cell – unlike the Mann-Kendall S statistic, this cannot be accumulated as a running sum, so it does not benefit from the same O(n^2)-time-but-O(1)-memory trick. For a modest `n` (a few dozen time steps) this is fast. For long series (hundreds to thousands of steps – e.g. multi-decade monthly data) the number of pairs per cell can reach the hundreds of thousands, and computing an exact median that many times per cell, over every cell, gets slow. Two ways to cope:

- `n_cores > 1`: splits cells across a `parallel::makeCluster()` PSOCK cluster (each cell's pairs still computed exactly).
- `max_pairs`: if `n*(n-1)/2` exceeds this, a random sample of `max_pairs` pairs is used per cell instead of the full set (a standard approximation for Theil-Sen on long series). Random pair sampling approximates the exact Theil-Sen slope; increasing `max_pairs` generally reduces Monte Carlo variability between runs and improves agreement with the exact estimate, though it is not guaranteed to be exactly unbiased in every finite sample. `seed` below controls the reproducibility of that approximation across runs. Set to `Inf` to force the exact computation regardless of `n`.

If speed genuinely matters more than robustness to outliers – very large rasters, very long series, and residuals not expected to be heavy-tailed or outlier-prone – `method = "OLS"` sidesteps this entire performance question: a single closed-form matrix operation, with no pairwise sampling and no per-cell iteration at all.

Optional queen-neighbourhood smoothing – read the caveats first

This is an optional visual/post-processing step, not part of the published Theil-Sen estimator itself.

What it does: `smooth_neighbourhood = TRUE` replaces each cell's slope with the median of its own slope and its 8 queen neighbours (a 3x3 focal median), computed *after* the per-cell Theil-Sen estimation above – it does not change how any individual cell's own slope is estimated.

What it does not inherit: this is **not the same thing as** `trend_test()`'s `neighbourhood` argument, and does not carry the same justification. CMK's neighbourhood-adjusted statistic follows a published, validated method (Neeti & Eastman, 2011) for the specific question "is there a trend", where borrowing spatial evidence for a yes/no decision is on solid ground. Smoothing the *magnitude* this way has no equivalent literature backing here – it assumes neighbouring cells share similar true slopes, which is not always true (e.g. a valley cell next to a sunlit slope can have a genuinely different real trend from its neighbours), and in that case this would blend two different real signals into one, less accurate, number for both.

When it might make sense: purely as a display/visual-smoothing aid for a map that will be read at a glance, where averaging out cell-to-cell estimation noise is more valuable than preserving every individual cell's own independent estimate – not as a way to "improve" the underlying statistic.

Warnings: off by default for the reasons above. If you use it, validate it first for your own data and resolution using `sim_trend_stack()` and `compare_detections()` against known ground truth, rather than judging it only by whether the smoothed map looks visually more coherent – a smoother-looking map is not the same as a more accurate one.

One thing this smoothing does **not** do: extend the footprint of "has data" beyond where data actually exists. A cell with no complete time series of its own (its own slope is `NA`) stays `NA` after smoothing,

even if all 8 of its neighbours have valid slopes – `terra::focal()`'s own default behaviour would otherwise fill such a cell in from its neighbours, which would be presenting an estimate for a location that was never actually observed.

Limitations

These estimators describe one overall linear rate and do not identify breakpoints, nonlinear trajectories, or seasonal components. Cells with incomplete time series are returned as `NA`. Optional spatial smoothing changes the reported local magnitude and must not be interpreted as part of any of the three published estimators.

Quality assurance

Theil-Sen slopes are compared exactly with `trend::sens.slope()`, including tied data; OLS results are checked against `stats::lm()`; and repeated-median slopes and intercepts are checked against direct hand implementations. Automated tests also cover irregular time coordinates, missing and constant series, optional neighbourhood smoothing, raster return types, and sequential/parallel equivalence. See `?sptrends` for the common release-check protocol.

References

`method = "TS"`, original estimator:

- Theil, H. (1950) A rank-invariant method of linear and polynomial regression analysis. Indagationes Mathematicae, 12, 85-91 (Part I; published in three parts). No DOI available (pre-DOI-era publication).

Generalisation into the median-of-pairwise-slopes estimator used by this function, linking it to Kendall's tau:

- Sen, P.K. (1968) Estimates of the regression coefficient based on Kendall's tau. Journal of the American Statistical Association, 63, 1379-1389. doi:10.1080/01621459.1968.10480934

`method = "OLS"`: the classical least-squares regression slope, independently attributed to both of the following (no single original paper; both pre-DOI-era):

- Legendre, A.M. (1805) Nouvelles méthodes pour la détermination des orbites des comètes. Firmin Didot, Paris.
- Gauss, C.F. (1809) Theoria motus corporum coelestium in sectionibus conicis solem ambientium. Perthes und Besser, Hamburg.

`method = "RM"`, original estimator:

- Siegel, A.F. (1982) Robust regression using repeated medians. Biometrika, 69(1), 242-244. doi:10.1093/biomet/69.1.242

The quasilinear-time algorithm the reference implementation this was ported from and verified against, `robslopes::RepeatedMedian()`, itself uses (not this package's own, deliberately simpler `O(n^2)` port – see "Implementation notes" above):

- Matoušek, J., Mount, D.M. and Netanyahu, N.S. (1998) Efficient Randomized Algorithms for the Repeated Median Line Estimator. Algorithmica, 20(2), 136-150. doi:10.1007/PL00009190

Documents `robslopes` itself, including the exact formula and upper- median convention this port was verified against:

- Raymaekers, J. (2023) robslopes: Efficient Computation of the (Repeated) Median Slope. The R Journal, 15(1), 249-260. doi:10.32614/RJ2023012

This function is used (not authored) by both of this package's own integrated workflows, `workflow_tst()` and `workflow_rta()`, and by the studies behind them:

- Gutiérrez-Hernández, O. and García, L.V. (2025) Uncovering true significant trends in global greening. Remote Sensing Applications: Society and Environment, 37, 101377. doi:10.1016/j.rsase.2024.101377
- Gutiérrez-Hernández, O. and García, L.V. (2024) Robust Trend Analysis in Environmental Remote Sensing: A Case Study of Cork Oak Forest Decline. Remote Sensing, 16(20), 3886. doi:10.3390/rs16203886

See Also

`trend_test()` for statistical evidence of temporal change; `workflow_trends()`, `workflow_tst()`, and `workflow_rta()` for workflows combining significance and magnitude.

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))

# The rate of change per cell, in NDVI units per year (the raster's
# own time unit) -- this is "how fast", not "is it significant" (that
# is what trend_test()/workflow_tst() answer instead).
result <- slope_estimator(r, verbose = FALSE, report = FALSE)

# A "slope" object: result$slope is the SpatRaster itself; plot() and
# summary() give the diverging map and the descriptive statistics
# without reconstructing either by hand.
plot(result)
summary(result)

# Ordinary least squares: much faster (a single closed-form matrix
# operation, no pairwise sampling) -- see "Methodological details"
# for when this efficiency is, and is not, worth
# its own trade-offs relative to the two rank-based methods.
result_ols <- slope_estimator(r, method = "OLS", verbose = FALSE,
                              report = FALSE)
```

`spatial_autocorrelation`

Permutation-based spatial autocorrelation tests

Description

Provides one interface for global and local spatial-autocorrelation analyses of a single-layer raster. Global analysis computes Moran's I or Getis-Ord General G. Local analysis computes a local Moran's I or Getis-Ord Gi* statistic for every valid cell. Local significance is assessed by spatial permutation. Its p-value raster can subsequently be passed to `fdr_correction()` for BH, BKY or BY control.

Usage

```
spatial_autocorrelation(
  x,
  method = c("moran", "getis_ord"),
  scope = c("global", "local"),
  connectivity = c("queen", "rook"),
  nperm = 99,
  alternative = NULL,
  alpha = 0.05,
  seed = NULL,
  n_cores = 1,
  precomputed_neighbourhood = NULL,
  report = TRUE,
  verbose = TRUE
)
```

Arguments

`x`	A single-layer `terra::SpatRaster` containing finite numeric values or `NA`.
`method`	Which statistic to compute. `"moran"` (default): global Moran's I (Moran, 1950) – accepts any numeric values (positive, negative, or mixed), tests whether nearby cells tend to have *similar* values, whatever those values are. `"getis_ord"`: Getis-Ord General G (Getis & Ord, 1992) – **requires non-negative values** (errors otherwise; see "Methodological background" below for why), tests specifically whether *high* values cluster near other high values (or low near low) rather than just "similar near similar" – a genuinely different question from Moran's I, not a variant of the same one, and the two do not always agree. Choose Moran's I when the question is whether neighbouring values are generally similar; choose General G when the question specifically concerns clustering of high values.
`scope`	Spatial scope. `"global"` (default) computes one statistic for the complete raster. `"local"` computes one local Moran's I or Getis-Ord Gi* statistic and one permutation p-value per valid cell.
`connectivity`	`"queen"` (default) or `"rook"`.
`nperm`	Positive integer. Number of permutations. Start with `99` to check the analysis runs correctly, use `999+` for a result to report, and `9999` for publication-quality analyses when computationally feasible (p-value resolution is `1/(nperm + 1)`).
`alternative`	`NULL` (default) selects `"greater"` for global tests and `"two.sided"` for local tests. May also be supplied explicitly as `"greater"`, `"two.sided"`, or `"less"`.

The global default reflects the common question of whether nearby observations are more positively associated than expected under spatial randomisation.

`alpha` One finite number strictly between 0 and 1. Significance level used only for the exploratory, unadjusted `significant_raw` map returned by local analysis. It does not control multiplicity. Use `fdr_correction()` on the returned `p` raster and specify its target `q` for BH, BKY or BY inference.

`seed` One finite numeric value or `NULL`. Random seed for reproducibility (used for both sequential and parallel execution).

`n_cores` Positive integer. Number of cores for the permutation loop (each permutation is independent). `1` (default): sequential. `> 1`: uses a `parallel::makeCluster()` PSOCK cluster. The spatial topology (the expensive part for large rasters) is computed once and shared, not recomputed per permutation or per worker.

`precomputed_neighbourhood`
Optional. Skips recomputing the spatial adjacency (the same "expensive for large rasters" step `n_cores` above already shares across permutations) when this function is called repeatedly on the same raster geometry – most usefully, together with `trend_test()`'s own `precomputed_neighbourhood` argument, when both functions are run on the same geometry in the same analysis. Accepts the output of `prepare_cmk_neighbourhood()` directly when it uses the matching default 3 by 3 adjacency – despite the name, both functions then build the identical structure underneath. Broader CMK windows have a different signature and are rejected rather than silently substituted. Its `W` component is validated as a square matrix with one row and column per raster cell. If `NULL` (typical single-call use), it is computed internally.

`report` Logical. If `TRUE` (default), automatically print the scope-specific summary and draw its plot once the test finishes. Global output shows the null distribution; local output maps the statistic, empirical z, raw p-value and exploratory unadjusted significance decision.

`verbose` Logical. Print progress messages and elapsed time.

Details

This is a general-purpose spatial diagnostic, not an FDR-specific utility. It can be applied to environmental variables, model residuals, estimated coefficients, test statistics, probabilities or other numeric spatial fields.

Function type: Spatial diagnostic function – performs global or local spatial-autocorrelation inference on one spatial field. It is independent of the temporal trend workflows.

Value

For `scope = "global"`, returns an object of class `c("spatial_autocorrelation_global", "spatial_autocorrelation" "sptrends")`: a list with `statistic` (observed Moran's I or Getis-Ord G, depending on `method`), `method`, `sign` (`"positive"` or `"negative"`, `method = "moran"` only – General G is always non-negative for non-negative inputs), `scope`, `p` (permutation p-value), `alternative`, `nperm`, `null_dist` (the null distribution), and `N` (valid cell count). For `scope = "local"`, returns an object of class `c("spatial_autocorrelation_local", "spatial_autocorrelation", "sptrends")`. Its `statistic`, `z`, `p`, and `significant_raw` components are single-layer rasters. `null_mean` and `null_sd` store

cell-wise permutation summaries and the empirical standardised statistic without retaining the potentially enormous cell-by-permutation matrix. `permutation_seeds` records the generated streams for exact auditing of the Monte Carlo calculation. `N` is the valid-cell count and `N_tested` excludes valid cells without any valid neighbour; their local outputs are `NA` and `fdr_correction()` excludes them from the hypothesis family.

Typical use

```
one spatial raster (variable, residual, coefficient, or statistic)
    |
spatial_autocorrelation(scope = "global" or "local")
    |
observed statistic + permutation inference + null distribution
```

The input is one spatial field, not a raster time series. A trend workflow output can be analysed only after selecting one derived layer, such as a slope or test-statistic raster.

Methodological details

Applications

Typical applications include describing clustering or dispersion in environmental variables, detecting spatial structure in model residuals, and examining the spatial distribution of estimated trends or effect sizes. One optional application is diagnosing dependence in a field of cell-wise inferential results before selecting or interpreting a multiple-testing procedure. Moran's I does not formally test independence or the complete dependence conditions underlying BH, BKY or BY, and must not be used as an automatic selector of an FDR procedure.

Spatial autocorrelation is not itself a temporal trend test. The input represents one spatial distribution, not a time series. The optional `moran_check` argument of `fdr_correction()` is only one convenience use of this independent diagnostic and has the same interpretive limitation described above.

Local statistics

With `scope = "local"` and `method = "moran"`, the statistic for cell `i` is `I_i = z_i sum_j(w_ij z_j) / m_2`, where `z_i` is the deviation from the global mean and `m_2 = sum_i(z_i^2) / N`. The binary queen or rook weights are not row-standardised. Under this definition, `sum_i(I_i) = S0 * I_global`, which is checked directly in the test suite. Positive values indicate locally similar deviations; negative values indicate a focal deviation surrounded by deviations of the opposite sign. Significance still depends on the permutation test.

With `method = "getis_ord"`, the local Gi* statistic is `G_i* = sum_j(w_ij* x_j) / sum_j(x_j)`, where `w_ii* = 1`: the focal cell is included, which distinguishes Gi* from Gi. High values indicate local concentrations of high raw values. The same non-negative-input requirement as General G applies. This function uses each cell's permutation distribution rather than an analytic normal approximation. The returned `z` raster standardises each observed local statistic by its own permutation mean and standard deviation. It aids comparison across cells with different neighbour counts, but it is not assigned an analytic normal p-value; use the returned permutation inference. Valid cells with no valid spatial neighbour are returned as `NA` for local inference and are excluded from the multiple-testing family.

Methods and method selection

- **Original publications**: Moran (1950) for `method = "moran"`; Getis & Ord (1992) for `method = "getis_ord"`.
- **Main references**: Moran (1950)/Getis & Ord (1992) for the statistics themselves; Tobler (1970) for the conceptual grounding of why spatial dependence is the expected default in geographic data; Hope (1968) for the permutation-based significance test used here in place of either statistic's own analytic Z approximation. Full citations under "References" below.
- **Why Getis-Ord needs non-negative values, specifically**: General G is a ratio of a weighted sum of *products* of raw values (`sum(w_ij * x_i * x_j)`) to the sum of all pairwise products (`sum(x_i * x_j)`), for `i != j` – unlike Moran's I, it does not mean-centre the values first. With mixed-sign data, `x_i * x_j` can be negative for two high-magnitude values of opposite sign, which breaks the statistic's actual interpretation (a proportion of "high-value pairs found near each other") – not merely unusual input, a violated precondition. Matches the same restriction in ArcGIS's High/Low Clustering (Getis-Ord General G) tool and other established implementations.
- **Typical applications**: Moran's I for "are similar values near each other"; General G for "do high values specifically cluster together". Inputs may be raw environmental variables, residuals, coefficients or inferential fields. For mixed-sign statistics, Moran is usually meaningful directly; General G requires a scientifically justified non-negative input.

Statistical assumptions and permutation inference

The classic analytic formulas for both statistics' significance need either a normality assumption (the values are normally distributed) or a randomisation assumption (the values are exchangeable), and the resulting Z-score can be inaccurate whenever the actual data violate them (Hope, 1968). Environmental variables, residuals and inferential outputs can all be bounded, skewed or otherwise non-normal. A permutation test builds the null distribution by actually reshuffling the observed values across the raster's cells (preserving their real distribution, skew, and bounds, whatever it is) and recomputing the chosen statistic each time, so significance comes from resampling the data you actually have rather than from an assumption about a distribution it may not follow.

Local inference and multiple testing

A local statistic map without a null distribution is descriptive: it shows where local association is large or small, but it supplies no inferential *p*-value. Consequently, there is no meaningful "alpha without permutations" in this implementation. Local p-values are Monte Carlo p-values obtained by permuting the observed values.

The returned `significant_raw` raster applies `p_i <= alpha`, treating every cell as though it were the only hypothesis. It is exploratory and controls neither FDR nor FWER across the map. For multiple-testing inference, pass the returned `p` raster to `fdr_correction()`. That function provides BH, BKY and BY from one independently tested and documented implementation. BH and BKY require independent tests or suitable positive dependence; BY is valid under arbitrary dependence but is usually more conservative. In all three cases, *q* is the target FDR and is not another name for the unadjusted per-cell *alpha*.

Every permutation is global and joint: all valid values are reassigned across the raster once, and every local statistic is recomputed from that same reassignment. It is not the conditional LISA permutation used by some software, where the focal value is held fixed and potential neighbours are sampled (Anselin, 1995). The two randomisation hypotheses answer different questions and their cell-wise p-values need not coincide. Permutation validity requires exchangeability under the stated spatial-randomisation null; it is not assumption-free.

More permutations improve Monte Carlo resolution and stability; they do not mechanically make a result more significant. The smallest attainable p-value is `1 / (nperm + 1)` because both numerator and denominator use the standard plus-one correction (Phipson & Smyth, 2010), so a randomly sampled permutation test never reports zero. A gradual strategy is useful: use about 99 permutations for exploratory checking, 999 or more for a stable analysis, and 9999 when tail resolution is important and the computational cost is acceptable. Results based on only 99 permutations should be described as exploratory rather than final.

Computational considerations

Runtime grows with the number of valid cells and permutations; local analysis must update a statistic for every tested cell in every joint randomisation. Reusing `precomputed_neighbourhood` avoids rebuilding topology, while `n_cores > 1` distributes independent permutations. Increasing `nperm` improves p-value resolution but increases runtime approximately proportionally.

Limitations

Methodological: the magnitude of either statistic is not universally comparable across different weight matrices (Cliff & Ord, 1981), so this function does not categorise the result into "low/moderate/strong" by default (see `classify_moran()` for an explicitly non-standard convenience label, `method = "moran"` only – General G's natural range and expected value under H0 differ enough from Moran's I that the same low/moderate/strong thresholds would not transfer meaningfully, so no equivalent label is offered for `method = "getis_ord"`). With `scope = "global"`, the result is one number for the complete raster and does not identify where clusters occur. Local Moran's I and Getis-Ord Gi* require one test per cell and, consequently, explicit multiple-testing control for confirmatory maps. Apply BH, BKY or BY with `fdr_correction()` and its target `q`; do not interpret the raw per-cell `alpha` map as family-level inference. This diagnostic is evidence of spatial dependence, not proof that a downstream method's complete dependence assumptions hold.

Implementation: fixed queen/rook neighbourhood, not a generic distance-band or k-nearest definition. NA cells are excluded, not imputed.

Quality assurance

Moran's I is checked against direct matrix calculations and exact small-raster references. Permutation tests cover reproducible seeds, alternative hypotheses, queen/rook neighbourhoods, missing and constant rasters, reused adjacency objects, and sequential/parallel equivalence. Local Moran's I is checked through its exact algebraic identity with global Moran's I. Gi* is checked against hand-computed small-raster results. Raw permutation p-values, raster geometry, missing values, local S3 methods, and forwarding the local p-value raster to `fdr_correction()` have separate regression tests. BH, BKY and BY are independently tested in that function's own suite. See `?sptrends` for the package-wide release-check protocol.

References

Primary references for the statistics being tested:

- Moran, P.A.P. (1950) Notes on Continuous Stochastic Phenomena. Biometrika, 37(1-2), 17-23. doi:10.1093/biomet/37.12.17
- Getis, A. and Ord, J.K. (1992) The Analysis of Spatial Association by Use of Distance Statistics. Geographical Analysis, 24(3), 189-206. doi:10.1111/j.15384632.1992.tb00261.x
- Anselin, L. (1995) Local Indicators of Spatial Association – LISA. Geographical Analysis, 27(2), 93-115. doi:10.1111/j.15384632.1995.tb00338.x

- Ord, J.K. and Getis, A. (1995) Local Spatial Autocorrelation Statistics: Distributional Issues and an Application. Geographical Analysis, 27(4), 286-306. doi:10.1111/j.15384632.1995.tb00912.x

Conceptual grounding for why spatial dependence is the expected default in geographic data (motivates testing for it at all):

- Tobler, W. (1970) A Computer Movie Simulating Urban Growth in the Detroit Region. Economic Geography, 46(sup1), 234-240. doi:10.2307/143141

Extended treatment of Moran's I and related spatial autocorrelation statistics:

- Cliff, A.D. and Ord, J.K. (1981) Spatial Processes: Models and Applications. Pion, London.

Basis for using permutation rather than the analytic Z approximation to obtain the p-value (see "Why permutation" above):

- Hope, A.C. (1968) A simplified Monte Carlo significance test procedure. Journal of the Royal Statistical Society, Series B, 30(3), 582-598. doi:10.1111/j.25176161.1968.tb00759.x
- Phipson, B. and Smyth, G.K. (2010) Permutation P-values Should Never Be Zero: Calculating Exact P-values When Permutations Are Randomly Drawn. Statistical Applications in Genetics and Molecular Biology, 9(1), Article 39. doi:10.2202/15446115.1585

On FDR procedures and the positive-dependence setting this function can diagnose but cannot establish:

- Benjamini, Y., & Yekutieli, D. (2001) The control of the false discovery rate in multiple testing under dependency. Annals of Statistics, 29(4), 1165-1188. doi:10.1214/aos/1013699998
- Benjamini, Y. and Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x
- Benjamini, Y., Krieger, A.M. and Yekutieli, D. (2006) Adaptive Linear Step-Up Procedures that Control the False Discovery Rate. Biometrika, 93(3), 491-507. doi:10.1093/biomet/93.3.491

See Also

Other Spatial autocorrelation diagnostic functions: `classify_moran()`, `spatial_autocorrelation_null_plot()`, `spatial_autocorrelation_summary()`

Examples

```
# Use a small real-data region at the dataset's native resolution so the
# example remains suitable for routine package checks.
series <- read_ordered_stack(example_data("vhp_ndvi"))
complete <- which(stats::complete.cases(
  terra::values(series, mat = TRUE)
))
centre <- terra::xyFromCell(
  series, complete[ceiling(length(complete) / 2)]
)
resolution <- terra::res(series)
region <- terra::ext(c(
```

```
  centre[1] + c(-6, 6) * resolution[1],
  centre[2] + c(-6, 6) * resolution[2]
))
series <- terra::crop(series, region, snap = "near")
X <- terra::values(series, mat = TRUE)
ok <- stats::complete.cases(X)
r <- series[[1]]
r_values <- terra::values(r, mat = FALSE)
r_values[!ok] <- NA_real_
terra::values(r) <- r_values

# I > 0 means neighbouring cells tend to have similar values
# (positive spatial autocorrelation); result$p is the permutation
# p-value for that observed I.
# Nineteen permutations are sufficient for this interface example;
# substantive inference should use a larger value.
result <- spatial_autocorrelation(
  r, nperm = 19, seed = 1, report = FALSE, verbose = FALSE
)
result
plot(result)   # the null distribution, with the observed I marked

# Getis-Ord General G needs non-negative values -- abs() of a
# (possibly mixed-sign) trend statistic is a common way to get there
# when the question is "do high-magnitude values cluster together?"
r_abs <- abs(r)
result_g <- spatial_autocorrelation(r_abs, method = "getis_ord",
                                    nperm = 19, seed = 1,
                                    report = FALSE, verbose = FALSE)
result_g$statistic

# One optional inferential application: describing spatial association
# in the p-value raster from trend_test(). This does not prove PRDS
# or automatically choose an FDR procedure.
trend <- trend_test(series, report = FALSE, verbose = FALSE)
spatial_autocorrelation(
  trend$stats$p, nperm = 19, seed = 1,
  report = FALSE, verbose = FALSE
)

# --- Local inference, followed by the existing FDR module ---
# The statistic is descriptive by itself. The p raster is based on
# permutations. significant_raw uses the per-cell alpha without
# correcting for the number of cells tested.
local <- spatial_autocorrelation(
  r, scope = "local", nperm = 19, seed = 1,
  report = FALSE, verbose = FALSE
)
local$statistic
local$z
local$p
local$significant_raw
```

```
# BH, BKY and BY are all supplied by fdr_correction(); q is the
# target FDR, not the raw per-cell alpha above.
local_fdr <- fdr_correction(
  local$p, method = c("BH", "BKY", "BY"), q = 0.05,
  report = FALSE, verbose = FALSE
)
c(
  raw_alpha = terra::global(
    local$significant_raw, "sum", na.rm = TRUE
  )[1, 1],
  fdr_BH = terra::global(
    local_fdr$rasters$sig_BH, "sum", na.rm = TRUE
  )[1, 1],
  fdr_BKY = terra::global(
    local_fdr$rasters$sig_BKY, "sum", na.rm = TRUE
  )[1, 1],
  fdr_BY = terra::global(
    local_fdr$rasters$sig_BY, "sum", na.rm = TRUE
  )[1, 1]
)
plot(local)
plot(local_fdr)

# Both functions build the identical spatial adjacency structure
# underneath -- computing it once and reusing it across both calls
# (rather than letting each recompute it independently) matters for
# large rasters specifically, where that step is the expensive part.
nb <- prepare_cmk_neighbourhood(series, ok)
spatial_autocorrelation(r, precomputed_neighbourhood = nb,
                        nperm = 19, seed = 1,
                        report = FALSE, verbose = FALSE)
trend_test(series, precomputed_neighbourhood = nb,
           report = FALSE, verbose = FALSE)
```

`summary.sptrends` *Summarise a sptrends result*

Description

The detailed textual report behind `print()`'s own one-line overview – see `print.sptrends()` for the full class list and the rationale for one shared entry point per generic. Each class here calls its own underlying reporting function directly, listed below, rather than duplicating what `print()` already shows.

Usage

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

Arguments

object	An object of class "sptrends" (also one of "tst", "rta", "workflow_trends", "trend_test", "slope", "prewhiten", "fdr", "spatial_autocorrelation", "compare_detections", "sptrends_simulation", "sptrends_simulation_design", or "sptrends_benchmark"), from workflow_tst(), workflow_rta(), workflow_trends(), trend_test(), slope_estimator(), prewhiten(), fdr_correction(), spatial_autocorrelation(compare_detections(), sim_trend_stack(), simulation_design(), or benchmark_methods().
...	Passed on to the underlying reporting function (e.g. path; alpha for "tst"/"rta"/"trend_test" objects).

Details

Function type: Reporting/derived function – summarises an existing result and does not recompute its statistical analysis.

Value

Invisibly, whatever the underlying reporting function itself returns – see each section below.

Typical use

`result <- workflow_tst(x); summary(result)` for a detailed textual report.

Methodological details

Published workflow: "tst". The full detail behind the "tst" case of print.sptrends(): the uncorrected trend summary table and, if FDR correction was run, the FDR summary. Returns a list with trend and fdr (or NULL). **Published workflow: "rta".** The full detail behind the "rta" case of print.sptrends(): the uncorrected trend summary table, the Theil-Sen slope summary, and the FDR-BH summary. Returns a list with trend and fdr. **Configurable workflow: "workflow_trends".** The uncorrected trend table, optional slope summary and selected FDR summary. **Trend estimation: "trend_test".** Cell counts and increase/decrease/no-change breakdown at multiple alpha levels; calls trend_summary() internally. **Trend estimation: "slope".** Valid cells, range, median, mean, and the increasing/decreasing/flat breakdown; calls slope_summary() internally. **Diagnostic: "prewhiten".** Valid cells, cells prewhitened, mean rho among them, and median Durbin-Watson; calls prewhiten_summary() internally. **Diagnostic: "fdr".** Significant/not-significant counts and percentages for every method requested; calls fdr_summary() internally. **Diagnostic: "spatial_autocorrelation".** Global results report the observed statistic, permutation distribution and empirical summary statistics. Local results report the statistic range, minimum permutation p-value and exploratory raw-significance count. **Validation: "compare_detections".** Which method scores best on each numeric metric in the table – a small table of its own, metric/best_method, not part of what compare_detections() itself computes. **Simulation and benchmarking.** Simulation summaries quantify the true signal, true-null proportion and slope range. Design summaries count levels per factor. Benchmark summaries retain scenario factors and aggregate performance over independent Monte Carlo replicates, including empirical FDR and FWER where available.

See Also

print.sptrends() for a concise overview and plot.sptrends() for graphical exploration.

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))
result <- workflow_tst(r, report = FALSE, verbose = FALSE)
summary(result)  # dispatches to the "tst" case above
```

`trend_test` *Trend tests for raster time series*

Description

Applies a trend test to a raster time series – the general step this function exists for, regardless of which specific test is used underneath. `trend_test()` is the core inferential step of sptrends. In a standard workflow it follows optional preprocessing (`compute_anomalies()`, `prewhiten()`) and precedes slope estimation (`slope_estimator()`) and multiple-testing correction (`fdr_correction()`).

Usage

```
trend_test(
  x,
  method = c("CMK", "MK", "OLS", "MMK"),
  ties = FALSE,
  t = NULL,
  window_size = 3L,
  connectivity = c("queen", "rook"),
  precomputed_neighbourhood = NULL,
  continuity = FALSE,
  n_cores = 1,
  alpha = c(0.1, 0.05, 0.01),
  report = TRUE,
  verbose = TRUE,
  shared_cluster = NULL
)
```

Arguments

`x`	A `terra::SpatRaster`; each layer is one time step, in increasing chronological order.
`method`	Which trend test to run. `"CMK"` is the default for spatially coherent gridded data; `"MK"`, `"OLS"`, and `"MMK"` address the distinct cases described below and are not equally weighted substitutes. `"CMK"` (default): full Contextual Mann-Kendall using a square queen neighbourhood. `window_size = 3L` reproduces the region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module, with at most `m = 9`: the focal cell plus eight valid neighbours. Larger odd windows extend the same CMK

equations to a broader region, as explicitly anticipated by Neeti and Eastman (2011). `"MK"`: classic cell-by-cell Mann-Kendall (no spatial averaging) – the foundational statistic `"CMK"` builds on. `"OLS"`: the classical parametric alternative – a per-cell ordinary least squares regression against `t`, with the standard t-test on the slope coefficient (`n - 2` degrees of freedom); like `"MK"`, no spatial averaging. `"MMK"`: the modified Mann-Kendall test of Hamed and Rao (1998) – corrects the classical MK test's own variance formula for temporal autocorrelation directly (an effective sample size, estimated from the significant autocorrelation of the Sen-detrended series' own ranks), rather than transforming the series the way `prewhiten()` does; like `"MK"`, no spatial averaging.

`"MMK"` **is an alternative to prewhitening, not a complement to it: do not run** `prewhiten()` **before using** `method = "MMK"`**.** Both exist to solve the same problem – temporal autocorrelation inflating the classical MK test's significance – by different means; `"MMK"` corrects the test statistic's own variance directly, without transforming the data first. Running it on an already-prewhitened series applies that correction to data with little autocorrelation left to detect (at best redundant), or compounds two corrections for the same issue (at worst biased) – see "Methodological details" below.

Use `"CMK"` when neighbouring cells are expected to exhibit coherent trends (the usual case for environmental rasters, and this function's own recommendation for that case – see "Methodological details" below). Use `"MK"` when reproducing the classic Mann-Kendall test, or when spatial pooling is not appropriate for the data at hand (e.g. cells that are not genuinely spatially related to one another). Use `"OLS"` when the residuals are genuinely expected to be close to normally distributed with no substantial outlier risk (in which case OLS is the more statistically efficient of the three), or when the result needs to be directly comparable to other analyses reported as classical linear-regression trend tests – see `slope_estimator()`'s own `method` argument for the same robust-vs-parametric trade-off applied to slope estimation instead of significance testing. Use `"MMK"` as an alternative to prewhitening when temporal autocorrelation is suspected but a spatial-pooling assumption (`"CMK"`'s own) is not appropriate, or when comparing against published studies reporting this particular correction.

`ties` Logical. Only used when `method` is `"CMK"`, `"MK"` or `"MMK"`. `FALSE` (default): closed-form variance (Eq. 3). `TRUE`: correct for tied (repeated) values within each cell's time series (Eq. 4) – relevant for low numeric-resolution variables. For CMK, every cell's resulting `VarS` is propagated through the complete RAMK covariance sum, so neighbouring series may have different tie-corrected variances. This option is slower because tie groups are calculated row by row. Most continuous environmental variables (NDVI, temperature, precipitation) rarely need this correction, since exact ties are uncommon at typical measurement precision.

`t` Numeric vector of time points, one per layer. Only used when `method` is `"OLS"` or `"MMK"` (`"CMK"`/`"MK"` are rank-based and need only the chronological *order* of layers, not their actual spacing; `"MMK"` needs `t` for its own Sen-slope detrending step, even though its significance test that follows is, like `"MK"`, rank-based). Defaults to `1:nlyr(x)` (evenly-spaced steps) if `NULL`. Supply real time values here if layers are *not* evenly spaced (e.g. a gap for a missing year) – unlike `"CMK"`/`"MK"`, OLS's own slope and its significance depend on the actual spacing,

not just the order. Values must be finite, unique and strictly increasing.

`window_size` Odd integer greater than or equal to `3`. Width and height, in raster cells, of the square neighbourhood used only by `method = "CMK"` (see `connectivity` below for its shape). The default `3L` preserves the CMK region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module. Larger values such as `5L` or `7L` change the spatial scale of the contextual test; choose them from the process scale and raster resolution, not as automatic upgrades. A non-default value with another `method` is rejected.

`connectivity` `"queen"` (default, 8 neighbours) or `"rook"` (4 neighbours, no diagonals). Only used when `method = "CMK"`. `"queen"` preserves the CMK region described by Neeti and Eastman (2011); `"rook"` is a narrower neighbourhood, useful when diagonal-only adjacency is not a meaningful spatial relationship for the process being tested, or to check how sensitive a result is to the choice of neighbourhood shape. A non-default value with another `method` is rejected.

`precomputed_neighbourhood`

Optional output of `prepare_cmk_neighbourhood()`, to avoid recomputing the adjacency when this function is called repeatedly on the same raster (e.g. in a permutation loop). It does not alter the statistical method in any way; it only skips rebuilding the spatial adjacency structure that `method = "CMK"` needs. Only used when `method = "CMK"`. If `NULL` (typical single-call use), it is computed internally. Reuse is accepted only when matrix dimensions, raster geometry, the requested connectivity, `window_size`, and the complete-case cell pattern all match `x`; incompatible objects produce an error rather than a result built from the wrong neighbours.

`continuity` Logical. Only used when `method = "CMK"`. Default `FALSE`: `Zm` is computed as `Sm / sqrt(VarSm)`, Eq. 10 exactly as published (Neeti and Eastman, 2011) – see "Implementation notes" below for why the continuity correction reserved for the single-cell classic MK statistic (Eq. 5) is not applied to the neighbourhood-averaged `Sm`. Set to `TRUE` to apply it anyway (`Zm = (Sm +- 1) / sqrt(VarSm)`), matching `ConMK`'s own convention instead – see "External validation" below, including `inst/validation/` for a reproducible, cell-by-cell comparison confirming the match for neighbourhoods without zero-variance series. At `Sm = 0`, this option deliberately follows `ConMK`'s `+1` branch rather than forcing `Z = 0`. This is an opt-in for matching another implementation's convention, not a recommendation to prefer it over this function's own default.

`n_cores` Integer. Number of cores to use for the S-statistic computation (the O(n^2) loop over pairs of time steps). Only used when `method` is `"CMK"`, `"MK"` or `"MMK"` – `"OLS"`'s own per-cell fit is already a single vectorised matrix operation across every cell at once, with no comparable loop to parallelise. `1` (default): sequential, identical to the original implementation. `> 1`: splits the outer loop across a `parallel::makeCluster()` PSOCK cluster (works on Windows, macOS, and Linux). Only worth it for long time series (large `n`) and/or very large rasters, since each worker needs a copy of the full data matrix.

`alpha` Numeric vector of significance thresholds. **What it controls**: only used when `report = TRUE`, passed to `trend_summary()` as-is. **How it is used for maps**: the single threshold for `trend_maps()` is `0.05` if it is one of the values in `alpha` (the default vector includes it), otherwise the strictest (smallest) value supplied.

The default `c(0.1, 0.05, 0.01)` reports all three side by side for context, but they are **not interchangeable defaults** – `0.05` is the conventional standard and the one used for the map; `0.1` is a more liberal threshold not unusual in exploratory trend studies (more power to detect a trend, at the cost of more false positives); `0.01` is markedly more conservative. **Why not a final result**: every value here is **uncorrected** for multiple testing – see "Methodological details" above; do not treat cells crossing `alpha` here as a final significance map.

`report` Logical. If `TRUE` (default), automatically print the summary table (`trend_summary()`) and draw the diagnostic histograms and maps (`trend_histograms()`, `trend_maps()`) after computing. Set to `FALSE` for programmatic use (e.g. inside a loop, or when called from `workflow_tst()`).

`verbose` Logical. Print progress messages and elapsed time.

`shared_cluster` Advanced; most users never need this directly. An already-running `parallel::makeCluster()` PSOCK cluster (e.g. from `workflow_tst()`'s own `n_cores`) to reuse instead of building a new one from `n_cores` above. When supplied, `n_cores` is ignored – the shared cluster's own size was already decided by whoever built it. `NULL` (default): builds and tears down its own cluster from `n_cores`, exactly as before this argument existed.

Details

Four tests are available through one interface: Contextual Mann-Kendall (`"CMK"`, the default), classic Mann-Kendall (`"MK"`), ordinary least squares (`"OLS"`), and modified Mann-Kendall (`"MMK"`). They answer the same broad question – whether change over time is statistically distinguishable from no trend – but make different assumptions. See `method` and "Methods and method selection" below.

Function type: Core function – one of the core building blocks of TST and RTA. Typically followed by `fdr_correction()`, once this function's own p-values exist for every spatial cell.

Value

Returns a list of class `c("trend_test", "sptrends")`, with:

`stats` A 3-layer `SpatRaster`: `Sm`, `VarSm`, `p` for `method = "CMK"`; `S`, `VarS`, `p` for `method = "MK"` or `"MMK"` (`VarS` is the Hamed-and-Rao-corrected variance for `"MMK"`, the uncorrected closed-form variance for `"MK"`); `beta` (the OLS slope estimate), `se_beta` (its standard error), `p` for `method = "OLS"`.

`neighbourhood` Logical: `TRUE` if `method = "CMK"` (spatial averaging was used), `FALSE` for `"MK"`, `"MMK"` and `"OLS"`.

`window_size` The odd CMK neighbourhood size used, or `NULL` for non-CMK methods.

`method` The requested trend-test method.

`connectivity` For CMK, the selected queen or rook connectivity.

Use `print()`/`summary()`/`plot()` – see `print.sptrends()`, `summary.sptrends()`, and `plot.sptrends()` – rather than accessing `$stats` directly for anything beyond programmatic use.

Typical use

```
raster time series
    |
trend_test()
    |
test statistics + raw p-value raster (`result$stats$p`)
    |
fdr_correction()
```

Use `prewhiten()` first when serial correlation requires treatment, except when choosing `method = "MMK"`, which is itself the alternative variance correction; analyse the prewhitened `result$series` with the other methods. Estimate change magnitude separately with `slope_estimator()`; significance and slope answer different questions.

Methodological details

What CMK tests

CMK tests the null hypothesis of no monotonic trend for every focal raster cell while using the focal cell's local spatial context. It does not smooth or replace the observed values. Instead, it combines the Mann-Kendall evidence from the focal cell and its immediate neighbours, then adjusts the variance for correlation among those series. A positive pooled statistic (`Sm`) indicates a predominantly increasing local region; a negative value indicates a predominantly decreasing one. The returned `p` tests whether that local regional statistic is compatible with no monotonic trend.

The geographical rationale is that many environmental processes are spatially continuous: if a trend is real, nearby cells will often contain related evidence. CMK can therefore gain power over isolated cell-wise MK when this assumption is justified. It must not be used merely because the input is a raster: boundaries, fragmented habitats, categorical mosaics, or other abrupt spatial discontinuities can make neighbouring trends legitimately different. Use `"MK"` in those cases.

CMK and prewhitening – two separate operations

`trend_test(method = "CMK")` **does not prewhiten the series.** This point is easy to misunderstand from Neeti and Eastman (2011). Their article presents prewhitening and contextual significance testing together in one broader analytical procedure: the discussion and flowchart first address serial autocorrelation and then apply CMK to the resulting series. That presentation can make prewhitening look like an internal part of CMK. It is not. The defining CMK equations (their Eqs. 7-13) contain the local average of Kendall statistics, the cross-correlation-adjusted variance, and its standardisation; they contain no prewhitening transformation.

The two operations address different dependencies:

- **serial autocorrelation** is dependence through time within one cell; diagnose and, when justified, treat it before this function with `prewhiten()`, or use `"MMK"` as a non-prewhitening alternative;
- **spatial cross-correlation** is contemporaneous dependence between different cells; CMK accounts for it in `VarSm`.

Prewhitening one cell at a time does not remove the lag-zero cross-correlation between neighbouring cells. Conversely, CMK's spatial variance adjustment does not remove serial autocorrelation. A workflow that needs both therefore calls `prewhiten()` first and `trend_test(method = "CMK")`

second. Keeping them as separate, composable functions makes the statistical role of each step explicit.

Relationship between CMK and RAMK

CMK is the moving-window raster application of the analytical Regionally Averaged Mann-Kendall (RAMK) test of Douglas, Vogel and Kroll (2000). The mathematical logic is shared:

1. calculate Kendall's `S` for every series in a region;
2. average those statistics to obtain the regional statistic;
3. include all within-region cross-covariances in its variance;
4. standardise the regional statistic and obtain a p-value.

RAMK applies this calculation once to a predefined group of stations. CMK moves the region across a raster and returns one result per focal cell. With the default complete 3 by 3 queen neighbourhood, the local region contains nine series: the focal cell plus eight neighbours. Larger odd `window_size` values use the corresponding square region. At raster edges or beside missing cells, `m` is the number actually available, so incomplete regions are not silently treated as complete. This function is therefore an analytical RAMK calculation for each local raster region under the common-length conditions and covariance formulation developed in both papers; it is not a general RAMK interface for arbitrary station regions.

How CMK is calculated

For each valid cell, `trend_test()` performs the following steps.

1. Compute the ordinary Mann-Kendall statistic `S` from all ordered pairs of time steps. No slope magnitude enters this statistic.
2. Define the local square region from the focal cell and the valid queen neighbours within `window_size`.
3. Compute `Sm` as the mean of the `S` statistics in that region (Neeti and Eastman Eq. 7; Douglas et al. Eq. 7).
4. Estimate every lag-zero cross-correlation among the regional series and include the corresponding covariance terms in `VarSm` (Neeti and Eastman Eqs. 11-13; Douglas et al. Eqs. 11-13). When ties are corrected, each pair uses `rho[i,j] * sqrt(VarS[i] * VarS[j])`; thus different tie patterns may give different cell-wise variances without imposing the focal cell's variance on the rest of the region.
5. By default, calculate `Zm = Sm / sqrt(VarSm)` and its two-sided normal-approximation p-value.

The covariance adjustment is essential. Treating neighbouring series as independent would underestimate the uncertainty of their pooled statistic and create too many apparently significant trends. The correction is analytical and relies on the assumptions stated in the source methods; it should be interpreted as a model-based variance adjustment, not as proof that all forms of spatial dependence have disappeared.

Statistical assumptions and interpretation

- for every method, layers are ordered chronologically, represent comparable time steps, and valid cells have complete series;
- for `"CMK"`, `"MK"`, and `"MMK"`, the trend of interest is monotonic, not necessarily linear, and the normal approximation is adequate for the available record;

- for "CMK", local spatial coherence is scientifically defensible;
- for "CMK" and "MK", serial autocorrelation has been assessed separately; "MMK" instead adjusts the rank-test variance for it;
- for "OLS", the temporal relationship is linear and the usual regression-inference assumptions concerning residual independence, constant variance, and approximate normality are defensible;
- raw cell-wise p-values are followed by a multiple-testing procedure such as fdr_correction() before producing a final significance map.

Sm describes direction and strength on the Mann-Kendall rank scale, not change per year. Use slope_estimator() for magnitude. A small p supports a locally coherent monotonic trend; it does not establish causation, practical importance, linearity, or a particular slope.

This is an independent implementation of the published equations. It is not affiliated with the original authors and does not port code from TerrSet's Kendall module.

Methods and method selection

- **Original publication**: Neeti & Eastman (2011), the "Contextual Mann-Kendall" (CMK) extension of the classic Mann-Kendall test.
- **Why "CMK" is this function's own default and recommendation**: gridded environmental data (the case this package is built around) routinely shows spatial autocorrelation – a real trend at one cell is expected, not merely permitted, to look similar to its immediate neighbours (Tobler's first law). CMK is designed to use that expected structure to its advantage, borrowing statistical strength across the neighbourhood while adjusting the test statistic's own variance to keep the significance decision formally valid – see "Methodological details" above for the mechanism. "MK" is the right choice specifically when that spatial expectation does not hold for the data at hand (see "Why method = \"mk\" remains available" below), not as an equally-good default.
- **Why method = "MK" remains available**: classic Mann-Kendall is the reference method in the literature, with the longest track record and the simplest assumptions (no spatial pooling at all). "CMK" is not a strict replacement for it – see the method argument below for when each is the more defensible choice.
- **Why method = "OLS" is offered too**: "CMK"/"MK" are both rank-based (built on the Mann-Kendall S statistic), robust to outliers and non-normal noise – this package's own default assumption about gridded environmental data, and the reason "CMK"/"MK" are recommended ahead of "OLS" in general. "OLS" is the classical parametric alternative – the standard significance test for a linear regression slope – offered for the specific, narrower case where that robustness genuinely is not needed (see the method argument below), the same reason slope_estimator() offers Theil-Sen, OLS, and Siegel's repeated median rather than only one robust option: this package aims to be a platform for comparing such methods (see compare_detections()), not a vehicle for only one statistical philosophy.
- **Why method = "MMK" is offered, and why it is an alternative to prewhitening, not a complement**: temporal (not spatial) autocorrelation is a separate problem from the one CMK addresses, and this package's other answer to it is prewhiten() – which transforms the series before testing it. "MMK" (Hamed and Rao, 1998) solves the same problem differently: it leaves the series untouched and corrects the classical MK test's own variance formula directly, using an effective sample size derived from the autocorrelation remaining in a Sen-detrended version of the ranks. Because both correct for the same underlying issue by different mechanisms,

applying both together (prewhitening the series and then also using "MMK") is not additive – see the method argument below for when "MMK" is the more appropriate choice on its own.

- **Main references**: Douglas, Vogel & Kroll (2000) for the Regionally Averaged Mann-Kendall (RAMK) statistic and its cross-correlation-adjusted variance; Neeti & Eastman (2011) for applying that analytical RAMK logic as CMK in a moving 3 by 3 raster neighbourhood, while explicitly noting that the technique can be extended to any neighbourhood size (CMK itself does not prewhiten); Mann (1945) and Kendall (1975) for the foundational statistic "CMK"/"MK" extend; Tobler (1970) for the spatial-coherence rationale behind averaging with the neighbourhood; Legendre (1805) and Gauss (1809) for the classical least-squares regression "OLS" is built on; Hamed and Rao (1998) for "MMK". Full citations appear under "References" below.
- **Typical applications**: trend detection in gridded remote sensing and climate time series, where a genuine trend is expected to be spatially coherent rather than isolated in a single pixel.

Monotonic trends only – deseasonalise first if needed

Mann-Kendall (and its contextual variant here) tests for a **monotonic** trend: a consistent tendency to increase or to decrease over time. It is not designed to detect or accommodate a periodic/seasonal cycle – if the input has one (e.g. raw monthly data with an annual cycle), that cycle itself will dominate the ranking of values and can produce a meaningless or misleading trend result. Deseasonalise first (see `compute_anomalies()`) and run this function on the anomalies, not on the raw seasonal series.

Implementation notes

Eq. 10 (`Zm = Sm / sigma_m`) is implemented as written, **without** the continuity correction: confirmed directly against Neeti and Eastman's own text for Eqs. 7-10, which introduce `Sm`, its mean and variance, and `Zm` with no +-1 term anywhere in that derivation – the continuity correction belongs to the single-cell classic MK statistic (Eq. 5) specifically, whose discrete step-2 structure motivates it; `Sm`, being an average across a neighbourhood, loses that structure. Cells with a constant time series (zero variance) would otherwise produce `NaN` correlations that propagate to their valid neighbours; here the zero standard deviation is replaced by `Inf` before dividing, giving a well-defined zero correlation. The neighbour-neighbour covariance term is computed via a closed-form identity that avoids an O(k^2) loop per cell, and the whole variance adjustment uses sparse matrix multiplication (package `Matrix`) rather than a `tapply()` loop.

External validation

`ConMK` **(Antiphon, GitHub, https://github.com/antiphon/ConMK, not on CRAN)**: the strongest external check available for this function, since its C++ source was inspected directly rather than trusted from its own documentation. `S`/`Sm` matched to floating-point precision on 100 cells of simulated data. The bundled frozen comparison records `S`/`Sm` and `p`; the accompanying script also records both variance estimates when the comparison is re-run. `p` differed for 97 of the 100 cells (the remaining 3 have `Sm` near zero, where the correction below crosses zero and produces the opposite-signed effect instead), traced to a specific line in `ConMK`'s own source (`Z = (Sm +- 1) / sqrt(s2)`, a continuity correction this function deliberately omits – see "Implementation notes" above, confirmed directly against Eqs. 7-10). Not a bug in either implementation: `ConMK` applies the correction meant for the single-cell classic MK statistic to the neighbourhood-averaged one instead, a different choice from what Eq. 10 itself specifies. Set `continuity = TRUE` (see that argument above) to reproduce `ConMK`'s continuity convention. An automated regression test matches its frozen `p` values for the 91 cells whose neighbourhoods contain no constant series. **One narrow, documented exception**:

for the 9 of 100 cells in that comparison affected by a constant (zero-variance) neighbouring cell, the cross-correlation term either implementation needs is undefined (0/0); this package's own convention (zero standard deviation replaced with `Inf`, giving zero correlation – see "Implementation notes" above) is not asserted to match `ConMK`'s own, unknown, handling of the same undefined case.

TerrSet's own Kendall module (Eastman's own reference implementation): `method = "MK"` (no spatial pooling) matched TerrSet's own classic Kendall module exactly on the same 100 cells, including `p`. The contextual case (`method = "CMK"`, TerrSet's own `KENDALL_Crosscorrelation` module) did not match as cleanly, and with no consistent, explainable pattern the way the `ConMK` difference has – but TerrSet is closed-source, and its own intermediate values (the pooled statistic and its own variance, analogous to this function's `Sm`/`VarSm`) are not exposed for inspection, only the final `Z`/`p`. Becoming free to use in 2024 (as "liberaGIS") did not make its own source available – what is published on GitHub is the installer and user guide, not the analytical modules' own code. Without the intermediate values, this difference cannot be traced to a specific formula choice the way the `ConMK` one was, and is reported here as an unresolved, inconclusive observation, not a confirmed discrepancy – closed, unverifiable code is not treated as a ground truth this function's own output is expected to match.

Independent re-confirmation (2026): a separate, freshly-run comparison against an installed copy of `ConMK` – not a reuse of the frozen comparison above – reproduced both findings on new simulated data. On a 10x10 grid (no continuity correction): `S` matched to floating-point precision (correlation 1, maximum absolute difference 0) across every cell; `p` correlated at 0.998, with a maximum absolute difference of 0.087 in the general case (consistent with the continuity-correction difference already documented above, not a new discrepancy). On a separate 15x15 null field with `continuity = TRUE`: of the cells landing exactly on `Sm == 0`, `ConMK`'s own `p` matched this package's `p` to seven decimal places at every one of them, directly confirming (not just tracing to source code, as before) that `continuity = TRUE` reproduces `ConMK`'s own convention exactly at this specific edge case, not only in the aggregate.

Computational considerations

A complete `window_size` by `window_size` region contains up to `window_size^2` series including the focal cell. Increasing the window therefore enlarges the sparse adjacency and every regional covariance calculation; runtime and memory do not remain constant. Start with the default 3 by 3 region and benchmark broader scales on representative data.

Observed execution time versus TerrSet: in the like-for-like validation run, using the same input raster, this implementation of CMK completed substantially faster than TerrSet's contextual Kendall module. This is a practically important advantage for large raster series. It is reported as an observed comparison rather than a universal benchmark: absolute speed and the ratio between programs depend on hardware, raster dimensions, storage, software versions, and the parallel settings used.

Quality assurance

The implementation is exercised at several independent levels:

- hand-calculated tests verify `S`, tie-corrected `VarS`, `Sm`, `VarSm`, and their p-values;
- an equation-level test verifies that the centre cell of a 3 by 3 CMK window equals a direct analytical RAMK calculation for the same region;
- regression tests verify that the default and explicit 3 by 3 results are identical, and that 5 by 5 regions use the expected valid cells;
- classic MK is compared with `Kendall::MannKendall()` and `trend::mk.test()`;

- CMK is compared cell by cell with the open-source ConMK implementation, with its continuity convention tested separately;
- rkt independently verifies the regional score aggregation (S / m = Sm); its Hirsch-Slack corrected variance is deliberately not required to equal CMK's analytical RAMK variance;
- the classic TerrSet Kendall result is recorded as an exact external match; this is only a partial TerrSet validation because its closed-source contextual result, which exposes neither Sm nor VarSm, is retained as an inconclusive comparison;
- automated tests cover ties, constant series, missing and isolated cells, raster edges, queen and rook adjacency objects, serial versus parallel execution, continuity choices, reporting, and returned object structure.

Frozen external results and their reproducible script are stored in inst/validation/. Package-wide checks also include the complete testthat suite, coverage inspection, R CMD check, goodpractice, lintr, and spelling review; see ?sptrends for the overall quality policy. A tool being part of that policy does not imply that a particular source archive has passed its latest run: release-specific results are recorded only after an actual run in cran-comments.md.

Limitations and scope

It does not prewhiten the series for serial correlation (see prewhiten()), does not correct for multiple testing (see fdr_correction()), and does not compute a separate robust estimate of trend magnitude such as the Theil-Sen slope. The "OLS" branch necessarily returns its fitted slope (beta), but robust and method-independent magnitude estimation belongs to slope_estimator().

CMK defaults to the local queen 3 by 3 region described in the source paper, as implemented in TerrSet's Kendall module. Neeti and Eastman (2011) explicitly state that the technique can be extended to any neighbourhood size, so larger odd window_size values apply the same equations at a broader spatial scale. This changes the scale of inference: larger windows can stabilise broad regional signals but may dilute small or spatially heterogeneous trends. At edges and beside invalid cells the available region becomes smaller. "MK", "MMK", and "OLS" do not perform spatial pooling. None of the four methods, by itself, controls multiplicity across the returned raster.

Multiple testing: the p-values and alpha thresholds are uncorrected

This applies identically to all four methods ("CMK", "MK", "OLS", and "MMK") – the problem below is about *how many tests were run*, not about which test statistic each one used.

What alpha **is**: the significance threshold (e.g. 0.05) is the pre-specified Type I error rate of a test under a true null hypothesis. It is not the posterior probability that a detected trend is false, and its single-test guarantee does not automatically extend to a raster-wide family of tests.

The error that follows from using it directly here: this function returns one test for every valid cell – a raster with thousands of cells means thousands of simultaneous tests, not one. CMK results are also spatially related through overlapping local regions; "cell-wise" does not mean statistically independent. Reading alpha against p cell by cell, with no adjustment for that total count, is a form of selective inference: some cells will cross any fixed alpha purely by chance, regardless of whether a real trend is present anywhere, and deciding significance cell by cell without accounting for how many tests were run leaves both the family-wise error rate and the false discovery rate uncontrolled. Consequently, an unknown fraction of the cells called "significant" this way may be false discoveries. The summary and maps produced when report = TRUE describe this *uncorrected* result only, not a defensible final significance map, whichever method was used to produce them.

What to do about it: always run `fdr_correction()` on `p` before reporting which cells are "significant" – see Gutiérrez-Hernández & García (2025) below for the full statistical argument (this exact problem, in this exact context, is what that paper is about).

References

Foundational statistic this method builds on (the `S`/`VarS` this function computes, before the neighbourhood adjustment below):

- Mann, H.B. (1945) Nonparametric tests against trend. Econometrica, 13(3), 245-259. doi:10.2307/1907187
- Kendall, M.G. (1975) Rank Correlation Methods (4th edn). Charles Griffin, London. No DOI available (pre-DOI-era publication).

Primary method reference:

- Neeti, N. and Eastman, J.R. (2011) A Contextual Mann-Kendall Approach for the Assessment of Trend Significance in Image Time Series. Transactions in GIS, 15(5), 599-611. doi:10.1111/j.14679671.2011.01280.x

`method = "OLS"`: the classical least-squares regression this test's own t-test is built on, independently attributed to both of the following (no single original paper; both pre-DOI-era):

- Legendre, A.M. (1805) Nouvelles méthodes pour la détermination des orbites des comètes. Firmin Didot, Paris.
- Gauss, C.F. (1809) Theoria motus corporum coelestium in sectionibus conicis solem ambientium. Perthes und Besser, Hamburg.

`method = "MMK"`:

- Hamed, K.H. and Rao, A.R. (1998) A modified Mann-Kendall trend test for autocorrelated data. Journal of Hydrology, 204(1-4), 182-196. doi:10.1016/S00221694(97)00125X

Conceptual grounding for the spatial-coherence rationale behind averaging with the neighbourhood:

- Tobler, W. (1970) A Computer Movie Simulating Urban Growth in the Detroit Region. Economic Geography, 46(sup1), 234-240. doi:10.2307/143141

Source of the Regionally Averaged Mann-Kendall (RAMK) regional statistic and its analytical cross-correlation variance correction; CMK applies the same equations to a moving local raster region:

- Douglas, E.M., Vogel, R.M. and Kroll, C.N. (2000) Trends in Floods and Low Flows in the United States: Impact of Spatial Correlation. Journal of Hydrology, 240(1-2), 90-105. doi:10.1016/S00221694(00)00336X

Official software implementation (this function is an independent re-implementation of the published equations, not a port of this module's code):

- Eastman, J.R. (2016) TerrSet Geospatial Monitoring and Modeling System: Kendall module. Clark Labs, Clark University, Worcester, MA.

On why the uncorrected alpha/p from this function should not be read as a final significance result (see "Methodological details" above):

- Gutiérrez-Hernández, O. and García, L.V. (2025) The ghost of selective inference in spatiotemporal trend analysis. Science of The Total Environment, 958, 177832. doi:10.1016/j.scitotenv.2024.177832

This function is used (not authored) by the following studies, which can serve as applied examples of the Contextual Mann-Kendall test in practice:

- Gutiérrez-Hernández, O. and García, L.V. (2025) Uncovering true significant trends in global greening. Remote Sensing Applications: Society and Environment, 37, 101377. doi:10.1016/j.rsase.2024.101377
- Gutiérrez-Hernández, O., & García, L.V. (2024) Robust Trend Analysis in Environmental Remote Sensing: A Case Study of Cork Oak Forest Decline. Remote Sensing, 16(20), 3886. doi:10.3390/rs16203886
- Gutiérrez-Hernández, O. and García, L.V. (2025, September 17) Multiple Testing in Remote Sensing: Addressing the Elephant in the Room. Available at SSRN: https://ssrn.com/abstract=4891512. doi:10.2139/ssrn.4891512
- Gutiérrez-Hernández, O., & García, L.V. (2025) False discovery rate estimation and control in remote sensing: reliable statistical significance in spatially dependent gridded data. Remote Sensing Letters, 16(5), 537-548. doi:10.1080/2150704X.2025.2478664
- Gutiérrez-Hernández, O., & García, L.V. (2025) Implementing the Linear Adaptive False Discovery Rate Procedure for Spatiotemporal Trend Testing. Mathematics, 13(22), 3630. doi:10.3390/math13223630
- Gutiérrez-Hernández, O., & García, L.V. (2026) Intensified and Extended Growing Seasons in Abies marocana Forests (2000-2024): A Robust Seasonal Trend Analysis Using 16-Day MODIS EVI Time Series. Remote Sensing, 18(12), 2052. doi:10.3390/rs18122052

See Also

prewhiten() for temporal preprocessing, slope_estimator() for trend magnitude, and fdr_correction() for multiplicity across valid cells; workflow_trends() combines these stages in one configurable workflow.

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))

# Test every cell for a monotonic trend, borrowing strength from each
# cell's spatial neighbourhood (method = "CMK", the default).
trend <- trend_test(r, report = FALSE, verbose = FALSE)
trend
summary(trend)

# Significance and direction, rather than the raw trend statistic.
plot(trend, panels = c("significance", "direction"))

# A larger odd window applies the CMK equations at a broader scale.
```

```
trend_5 <- trend_test(r, window_size = 5L, report = FALSE,
                      verbose = FALSE)
trend_5$window_size
```

`workflow_rta` *Robust Trend Analysis (RTA): the full pipeline in one call*

Description

Implements the complete Robust Trend Analysis workflow for monotonic trends in gridded raster time series. One of sptrends' two entry points: chains `slope_estimator()`, `trend_test()`, and `fdr_correction()` (`method = "BH"` only), in that order, into the **Robust Trend Analysis (RTA)** workflow. Each step's underlying parameters remain available directly on the individual functions – `workflow_rta()` does not replace them, it saves wiring the calls together for the common case, and returns a single `"rta"` object recognised by `print.sptrends()`. RTA is a **different, shorter** published workflow from this package's other integrated pipeline, `workflow_tst()` – see "Methodological comparison with TST" below before choosing between them. Both are implemented here because both are genuinely published methods, not because one supersedes the other – this package aims to be a platform for comparing such methods (see `compare_detections()`), not a vehicle for only its own authors' preferred one.

Usage

```
workflow_rta(
  x,
  cmk_args = list(),
  theil_sen_args = list(),
  alpha = 0.05,
  q = 0.05,
  report = TRUE,
  verbose = TRUE,
  n_cores = 1
)
```

Arguments

`x`	A `terra::SpatRaster`; each layer is one time step, in increasing chronological order. Used directly, unprewhitened – see "Comparison with TST" under "Methodological details" below.
`cmk_args`	A named list of extra arguments passed to `trend_test()` (e.g. `list(window_size = 5L)` for a broader CMK region, or `list(method = "MK", n_cores = 4)`). The default empty list preserves the 3 by 3 CMK region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module; changing it creates an RTA-inspired variant rather than an exact reproduction of the published workflow.

Known limitation with irregular time spacing: this workflow has no top-level `t` argument of its own – if your observation times are irregularly spaced and you need a genuine time axis for `trend_test()`/`slope_estimator()` rather than the default `1:n` layer index, you must supply the same correct `t` yourself in both `cmk_args` and `theil_sen_args`, and keep them consistent by hand.

`theil_sen_args` A named list of extra arguments passed to `slope_estimator()` (e.g. `list(max_pairs = 20000, n_cores = 4)`). `smooth_neighbourhood` is left at `slope_estimator()`'s own default (`FALSE`, same as `workflow_tst()`) unless you set it yourself – see "Computational considerations" above. See `cmk_args` above for a known limitation with irregular time spacing if you pass `t` here.

`alpha` Numeric vector of significance thresholds used for reporting the (uncorrected) trend result – passed to `trend_summary()` as `alpha`, and used as the single threshold for `trend_maps()`: `0.05` if it is one of the values in `alpha` (the default vector includes it), otherwise the strictest (smallest) value supplied. The three default values are not interchangeable: `0.05` is the conventional standard and the one actually used for the map; `0.1` is a more liberal threshold not unusual in exploratory trend studies; `0.01` is markedly more conservative. All three are shown side by side in `trend_summary_table` for context, not as equally valid choices.

`q` Numeric. Target FDR level, passed to `fdr_correction()` – not an adjusted or renamed `alpha`: it limits the expected false discovery proportion among rejected hypotheses. See `fdr_correction()` for the full distinction.

`report` Logical. If `TRUE` (default), each step prints its own summary table and draws its diagnostic plots as it runs (same effect as calling each function directly with `report = TRUE`). Set to `FALSE` for silent, plot-free programmatic use.

`verbose` Logical. Print progress messages and elapsed time for the complete workflow. Per-stage times are also returned in `timing`.

`n_cores` Integer. `1` (default): every step runs sequentially. `> 1`: builds **one** `parallel::makeCluster()` PSOCK cluster, shared across every parallel step below (currently the CMK and Theil-Sen steps) instead of each step creating and tearing down its own – avoiding the repeated process-spawn overhead of doing that two or three times in a row for what is, from the caller's own perspective, one parallel request. The cluster is closed automatically when `workflow_rta()` returns. Setting `n_cores` inside `cmk_args`/`theil_sen_args` directly still works exactly as before (each step falls back to building its own cluster from that value) – but only when this top-level `n_cores` is left at its default of `1`; when both are set, this one wins and the per-step `n_cores` inside `cmk_args`/`theil_sen_args` is ignored, since a single shared cluster and per-step separate ones cannot both apply to the same call. Mirrors `workflow_tst()`'s own identical `n_cores` argument exactly.

Details

When should I choose RTA?: want to reproduce the published 2024 workflow exactly -> use `workflow_rta()`. Want the more recent workflow, including selective prewhitening and the adaptive BKY correction -> use `workflow_tst()`. See "Methodological comparison with TST" below for the reasoning behind each design choice.

Moran's I (spatial_autocorrelation()) is **not** part of this pipeline, for the same reason it is not part of workflow_tst(): it is a separate diagnostic for the spatial dependence assumption behind FDR-BH, meant to be run independently (before or after) rather than chained automatically, and pulling it in here would add a dependency this function does not otherwise need – see the package vignette.

Function type: Core function – one of the two integrated published workflows in sptrends.

Value

An object of class c("rta", "sptrends") (the second, shared with workflow_tst()'s own return value, is for print()/summary()/ plot() – see print.sptrends()): a list with

theil_sen — The $slope field of slope_estimator()'s own output (a SpatRaster) – not the full return value. Not smoothed by default – check theil_sen_smoothed if you need to know whether it was (e.g. because you set it yourself).

theil_sen_smoothed — Logical: whether theil_sen was computed with smooth_neighbourhood = TRUE. FALSE by default.

trend — The $stats field of trend_test()'s own output (a SpatRaster) – not the full return value, which also includes neighbourhood/window_size.

trend_summary_table — The output of trend_summary() (invisible data frame).

fdr — The output of fdr_correction(), method = "BH" only.

timing — A named list of elapsed seconds per step (theil_sen, CMK, fdr), each measured with Sys.time() around that step's own function call. Coarse (whole-step, not line-by-line) and not a substitute for a proper profiler, but enough for noticing which step dominates on your own data.

Use print.sptrends() for a one-line summary.

Typical use

```
raster time series
    |
workflow_rta()
    |
Theil-Sen slope + CMK significance + FDR-BH
    |
one `rta` result containing every stage
```

RTA analyses the supplied series without prewhitening. If the input has a seasonal cycle, first use compute_anomalies() and pass its anomalies raster.

Methodological details

How it works.

```
Input raster
    |
```

```
slope_estimator    -- how fast? (Theil-Sen)
    |
trend_test         -- is there a monotonic trend? (CMK)
    |
FDR-BH correction  -- which cells survive multiple testing?
    |
"rta" object
```

All three steps always run – unlike `workflow_tst()`, none of them is optional here, matching the published RTA method exactly (see "Comparison with TST" under "Methodological details" for what is deliberately different between the two workflows). Each step's own output is kept in full on the returned object (see "Value" below) – nothing is discarded once a later step begins.

Statistical assumptions: monotonicity and seasonality. Every step here (the Contextual Mann-Kendall test, Theil-Sen) is designed around a **monotonic** trend – a consistent tendency to increase or decrease – not a periodic/seasonal cycle. If `x` has a seasonal cycle (e.g. raw monthly data with an annual signal), remove it first with `compute_anomalies()` and pass the anomalies to `workflow_rta()`, not the raw seasonal series.

Computational considerations. Unlike the Mann-Kendall S statistic, the Theil-Sen slope needs every pairwise slope in the series (`n*(n-1)/2` per cell) to take their median – it cannot be accumulated as a running sum, so it does not scale the same way. For short series (a few dozen time steps) this is unnoticeable; for long series (hundreds to thousands of steps, e.g. multi-decade monthly data) it can become the slowest step in the whole workflow. Tune `theil_sen_args` (`max_pairs` to subsample pairs, `n_cores` to parallelise) – see `slope_estimator()`. `smooth_neighbourhood` is left at `slope_estimator()`'s own default (`FALSE`) unless you set it yourself – neither RTA nor TST originally included any such smoothing (it is this package's own optional addition on top of both published methods), so `workflow_tst()` does not default to it either, for the same reason.

Statistical assumptions: `alpha` and `q`. This is one of the most common mistakes in gridded trend analysis, so it is worth spelling out: `alpha` (e.g. `0.05`) is defined for a *single* hypothesis test. A raster is not one test – it is one test per cell, potentially thousands of them run simultaneously. Applying `alpha` cell by cell, as if each cell were the only test being run, is exactly the multiple-testing error this package's FDR step exists to fix (see the "Warning" section of `?trend_test`); it is not a harmless simplification.

It is common practice to set `q` equal to `alpha` (e.g. both `0.05`) for convenience and comparability between the uncorrected and corrected results reported here – this function does not enforce that, `alpha` and `q` are independent arguments. But equal values does not mean equal meaning: `alpha` bounds the error rate of each individual cell's test, while `q` bounds the expected proportion of false positives among all the cells called significant after correction (see `fdr_correction()` for the full distinction). Do not read `trend_summary_table` (based on `alpha`, uncorrected) and the FDR results (based on `q`) as answering the same question just because the numbers match. `q = 0.05` has little reason to change: it is the standard target FDR level in the literature this package builds on, and lowering it (e.g. to `0.01`) mainly costs statistical power rather than offering a meaningfully different guarantee.

Comparison with TST. RTA (Gutiérrez-Hernández & García, 2024) and TST (Gutiérrez-Hernández & García, 2025; see `workflow_tst()`) share two pillars – Theil-Sen and Contextual Mann-Kendall – but differ in two deliberate, independent ways that this function keeps faithful to the published RTA method, rather than silently reusing TST's later choices.

Difference 1: prewhitening. TST's first step, selective AR(1) prewhitening, does not appear in RTA. Whether to prewhiten before a Mann-Kendall-family test is a genuine, unresolved methodological debate, not a settled question with one correct answer. Yue & Wang (2002) find that prewhitening can substantially reduce power, particularly when a real trend and real autocorrelation coexist: it can remove part of the trend signal together with the autocorrelation. Bayazit & Önöz (2007) argue the opposite case: skipping prewhitening when autocorrelation is genuinely present can inflate the false-positive rate. RTA does not prewhiten; TST prewhitens selectively, touching only cells whose own Durbin-Watson statistic crosses a gating threshold (see `prewhiten()`), specifically to limit unnecessary power loss. Neither position is implemented here as universally correct.

Difference 2: FDR-BH only, not adaptive BKY. `workflow_tst()` defaults to the two-stage adaptive BKY correction, which estimates how many tested hypotheses are likely genuinely non-null ($\hat{\pi}_0$) and relaxes its threshold when substantial real signal is detected; see `fdr_bky()`. RTA instead uses the original, non-adaptive Benjamini-Hochberg procedure, which does not estimate π_0 and is derived to control FDR under the global null: the least favourable case in which every tested hypothesis could be truly null. It therefore keeps a more conservative guarantee that does not relax as more signal is found. See `fdr_correction()` for the procedure itself.

These differences are independent: RTA's choice on one does not imply or require its choice on the other. They happen to be the less adaptive options in this package's two workflows, not because either dictates the other.

Limitations. RTA is intentionally faithful to its published method: it does not prewhiten, it always estimates a Theil-Sen slope, and it uses FDR-BH rather than exposing alternative corrections. Use `workflow_trends()` when those stages or methods need to be configured.

Quality assurance. CMK, slope estimation, and FDR correction are validated independently in their module functions. Integration tests verify the RTA stage sequence, argument forwarding, shared parallel resources, timing fields, S3 return structure, reporting, and agreement between direct module calls and workflow outputs. See `?sptrends` for the complete internal and external quality-assurance strategy.

References

Source of this workflow (primary reference – this pipeline is a direct implementation of it):

- Gutiérrez-Hernández, O. and García, L.V. (2024) Robust Trend Analysis in Environmental Remote Sensing: A Case Study of Cork Oak Forest Decline. Remote Sensing, 16(20), 3886. doi:10.3390/rs16203886

Step 1, Theil-Sen slope (see `slope_estimator()` for the full reference list):

- Theil, H. (1950) A rank-invariant method of linear and polynomial regression analysis. Indagationes Mathematicae, 12, 85-91 (Part I; published in three parts). No DOI available (pre-DOI-era publication).
- Sen, P.K. (1968) Estimates of the regression coefficient based on Kendall's tau. Journal of the American Statistical Association, 63, 1379-1389. doi:10.1080/01621459.1968.10480934

Step 2, Contextual Mann-Kendall (see `trend_test()` for the full reference list, including the foundational Mann-Kendall statistic it builds on):

- Neeti, N. and Eastman, J.R. (2011) A Contextual Mann-Kendall Approach for the Assessment of Trend Significance in Image Time Series. Transactions in GIS, 15(5), 599-611. doi:10.1111/j.14679671.2011.01280.x

Step 3, FDR-BH correction (see `fdr_correction()` for the complete reference list):

- Benjamini, Y. and Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x

On whether to prewhiten before a Mann-Kendall-family trend test (see "Comparison with TST" under "Methodological details" above):

- Yue, S. and Wang, C.Y. (2002) Applicability of prewhitening to eliminate the influence of serial correlation on the Mann-Kendall test. Water Resources Research, 38(6), 4-1-4-6. doi:10.1029/2001WR000861
- Bayazit, M. and Önöz, B. (2007) To prewhiten or not to prewhiten in trend analysis? Hydrological Sciences Journal, 52(4), 611-624. doi:10.1623/hysj.52.4.611

See Also

Other pipeline functions: `workflow_trends()`, `workflow_tst()`

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))

# Run the full workflow: Theil-Sen -> Contextual Mann-Kendall ->
# FDR-BH (only BH -- see "Comparison with TST" above
# for why this function does not offer BKY as an option).
result <- workflow_rta(r, report = FALSE, verbose = FALSE)

# An "rta" object: printing it gives a one-line-per-step summary
# (the Theil-Sen slope range, the trend test's cell count, and how
# many cells are significant after FDR-BH correction).
result

# The same plot() methods used throughout this package, rather than
# reconstructing either view by hand: which cells are significant
# after FDR-BH, and how fast those cells are changing.
plot(result, which = "significance")
plot(result, which = "slope")
plot(result)
```

workflow_trends *Configure a monotonic or linear trend-analysis workflow*

Description

This function analyses monotonic or linear temporal trends using a user-selected trend test – available methods include the Mann-Kendall family (`"CMK"`, `"MK"`, `"MMK"`) and ordinary least-squares regression (`"OLS"`); it does not detect abrupt changes, periodicity, or general nonlinear temporal patterns. Like `workflow_tst()` and `workflow_rta()`, each of which implements one specific published analytical workflow, this function analyses that same class of trend. Unlike those two, `workflow_trends()` lets you choose your own combination of prewhitening, trend testing, slope estimation and multiple-testing correction methods – useful when no single published workflow matches what a given analysis needs, or when comparing how sensitive a result is to that choice (see the package's own validation framework, `sim_trend_stack()`/ `compare_detections()`, for a way to evaluate that sensitivity against data with a known answer, rather than only this function's own runtime warnings below).

Usage

```
workflow_trends(
  x,
  prewhiten_method = c("TFPW_WS", "TFPW_Y", "TFPW_Z", "VCTFPW", "none"),
  trend_method = c("CMK", "MK", "OLS", "MMK"),
  slope_method = c("TS", "OLS", "RM"),
  fdr_method = c("BKY", "BH", "BY"),
  prewhiten_args = list(),
  trend_args = list(),
  slope_args = list(),
  fdr_args = list(),
  n_cores = 1,
  report = TRUE,
  verbose = TRUE
)
```

Arguments

`x`	A `terra::SpatRaster` with one layer per time step, ordered chronologically. All layers must share the same geometry. Temporal spacing is assumed regular unless explicit time coordinates are supplied through the relevant underlying function's own arguments (e.g. `trend_args = list(t = ...)`, `slope_args = list(t = ...)`).
`prewhiten_method`	Which prewhitening method to apply before trend testing, or `"none"` to skip this step entirely. One of `"TFPW_WS"` (default; Wang and Swail, 2001), `"TFPW_Y"` (Yue, Pilon and Cavadias, 2002), `"TFPW_Z"` (Zhang, Vincent, Hogg and Niitsoo, 2000), `"VCTFPW"` (Wang, Chen, Becker and Liu, 2015), or `"none"`. See `prewhiten()`'s own `method` argument for the full comparison between these

four, and "Combinations this function warns about" below for the one combination involving trend_method worth knowing about before combining.

trend_method — Which trend test to run. One of "CMK" (default; Neeti and Eastman, 2011), "MK" (classic Mann-Kendall), "OLS" (tests whether an ordinary-least-squares regression slope differs from zero, rather than a rank-based statistic – see trend_test()'s own "OLS" entry for the specific p this returns and the assumptions its own inference relies on), or "MMK" (Hamed and Rao, 1998). See trend_test()'s own method argument for the full comparison between these four.

slope_method — Which slope estimator to run, or NULL to skip this step entirely (matching workflow_tst()'s own theil_sen = FALSE). One of "TS" (default; Theil-Sen), "OLS", "RM" (Siegel's repeated median), or NULL. See slope_estimator()'s own method argument for the full comparison between the three.

fdr_method — Which multiple-testing correction to apply, or NULL to skip this step. One of "BKY" (default; Benjamini, Krieger and Yekutieli, 2006), "BH" (Benjamini and Hochberg, 1995), or "BY" (Benjamini and Yekutieli, 2001). "BY" is an opt-in safeguard when control under arbitrary dependence is scientifically required; it is generally more conservative than "BH" and adaptive "BKY". See fdr_correction()'s own method argument, and fdr_by()'s own documentation, for the full reasoning.

prewhiten_args, trend_args, slope_args, fdr_args — Named lists of extra arguments forwarded to the corresponding underlying function, beyond method itself (which this function's own arguments above already control) – e.g. trend_args = list(ties = TRUE, window_size = 5L), fdr_args = list(q = 0.1). For CMK, omitting window_size preserves the 3 by 3 CMK region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module. Arguments already controlled directly by workflow_trends() itself – x, method, report, verbose, and cluster- management arguments (n_cores/shared_cluster) – must not be repeated in these lists; a clear error is raised instead of silently overriding the workflow configuration.

n_cores — Integer. Number of workers made available to stages that support parallel execution – a single PSOCK cluster is created once and reused where possible, not one per step. Not every method actually uses it: slope_method = "OLS"'s own vectorised computation and the current "RM" implementation, for instance, do not. 1 (default): sequential. Requesting more cores than are actually available falls back to the number detected, with a message (see ?trend_test's own n_cores for the exact behaviour every step here shares).

report — Logical. Forwarded to every step's own report argument – if TRUE, each analytical stage may print its own component-level summary and draw its own diagnostic plots as it runs, not only a single consolidated summary of the whole workflow at the end. Set FALSE to suppress this intermediate output and inspect the returned object directly instead.

verbose — Logical. Forwarded to every step's own verbose argument, and controls this function's own step-by-step progress messages and elapsed time. Per-stage times are returned in timing.

Details

Function type: Core function – a configurable composition of the same analytical stages used by

the published workflows.

Value

A list of class `c("workflow_trends", "sptrends")`:

`prewhiten`	The output of `prewhiten()`, or `NULL` if `prewhiten_method = "none"`.
`trend`	The multilayer `SpatRaster` returned as `trend_test()`'s own `$stats` (the test statistic, raw p, and any method-specific layers – see `?trend_test`'s own "Value" section for exactly which, since this differs by `trend_method`).
`trend_summary_table`	The output of `trend_summary()` on `trend`.
`slope`	The single-layer `SpatRaster` extracted from `slope_estimator()`'s own `$slope` (not the full `"slope"` object that function itself returns), or `NULL` if `slope_method = NULL`.
`fdr`	The output of `fdr_correction()`, or `NULL` if `fdr_method = NULL`.
`timing`	A named list of per-step elapsed seconds, only for steps that actually ran.

Typical use

```
raster time series
    |
workflow_trends()
    |
optional prewhitening -> trend test -> optional slope -> optional FDR
    |
one `sptrends` result containing every computed stage
```

Use this configurable workflow when the fixed published combinations in `workflow_tst()` and `workflow_rta()` do not match the analysis. For seasonal input, first pass the `$anomalies` component returned by `compute_anomalies()`.

Methodological details

Relationship to TST. True Significant Trends (TST) is the methodological origin of sptrends and supplies the architecture of the complete workflow: temporal preprocessing, trend testing, slope estimation and multiple-testing correction. TST includes all these stages, but not every method now available here. `workflow_trends()` generalises the architecture with additional methods and optional stages. A complete configuration retains the analytical philosophy inherited from TST; the name TST remains reserved for the published methods and sequence reproduced by `workflow_tst()`.

Every method available on the individual functions this orchestrates – `prewhiten()`, `trend_test()`, `slope_estimator()`, `fdr_correction()` – can be freely combined here, with two exceptions this function warns about explicitly rather than silently allowing (see "Combinations this function warns about" below). The trend test and slope estimator answer different questions and need not use the same statistical framework – mixing a robust trend test with a parametric slope estimator (or vice versa) is unconventional, but not warned about here. Nevertheless, users remain responsible for ensuring the selected combination is appropriate for their own data and scientific objective; other

combinations this function does not itself flag can still be questionable for a given dataset (e.g. `"CMK"` on an already spatially smoothed series – see "Combinations this function warns about" below for the one combination this function does check for explicitly).

How it works.

1. (optional) `prewhiten` – removes serial autocorrelation.
2. `trend_test` – is there a monotonic trend?
3. (optional) `slope_estimator` – how fast? and (optional) FDR correction – which cells survive multiple testing? These two are independent branches after the trend test, not a sequence: FDR correction depends only on the test's own p-values, not on whether a slope was estimated at all.

The same ordering logic as the two published workflows applies: prewhitening (when requested) happens before the trend test, since the test assumes independent observations; the slope (when requested) is estimated on the same (optionally prewhitened) series the test itself used; and FDR correction needs the test's own p-values to exist first, but nothing else – it does not depend on whether a slope was estimated at all. Unlike prewhitening and FDR correction, which were already optional here from the start, slope estimation being skippable too (`slope_method = NULL`) was added later, to bring this function to full parity with `workflow_tst()`'s own `theil_sen = FALSE` – see NEWS.md for when.

Statistical assumptions: `alpha` and `q`. As with `workflow_tst()`/`workflow_rta()`: q (inside `fdr_args`, e.g. `fdr_args = list(q = 0.1)`) bounds the expected proportion of false positives among cells called significant *after* correction – it is not the same quantity as an uncorrected per-cell α. Applying a per-cell significance threshold across many cells without multiplicity control can substantially increase the number of false-positive detections, the multiple-testing problem `fdr_method` exists to address. See `?workflow_tst`'s own "A note on alpha and q" section for the fuller explanation, and `fdr_correction()` for the distinction in full.

Statistical assumptions: monotonicity and seasonality. As with `workflow_tst()`/`workflow_rta()`: every method this function can combine is designed around a **monotonic** trend, not a periodic/seasonal cycle. If `x` has a seasonal cycle (e.g. raw monthly data with an annual signal), remove it first with `compute_anomalies()` and pass the anomalies here, not the raw seasonal series.

Computational considerations. Runtime depends primarily on the selected methods and raster size. Theil-Sen and repeated-median slopes are more expensive than OLS; CMK and parallel-capable stages can reuse one workflow-level PSOCK cluster. Per-stage elapsed times are retained in the returned object.

Limitations. Configurability does not make every combination scientifically interchangeable. The caller remains responsible for matching the chosen assumptions to the data; the explicit safeguards below cover important known conflicts, not every possible misuse.

Methodological safeguards. Combining any prewhitening method with `trend_method = "MMK"` is warned against because both address temporal autocorrelation by different mechanisms. Applying both is redundant and may compound the correction. The warning is issued before computation; it does not forcibly stop an explicitly requested analysis.

Quality assurance. Integration tests exercise every available method family, optional stages, invalid and questionable combinations, protected argument forwarding, BH/BKY/BY display paths, exact warning behaviour, shared parallel resources, timing fields, and returned S3 structure. Component results are compared with direct calls to `prewhiten()`, `trend_test()`, `slope_estimator()`, and `fdr_correction()`. See `?sptrends` for the complete internal and external quality strategy.

References

Primary method reference (`prewhiten_method = "TFPW_WS"`, the default):

- Wang, X.L. and Swail, V.R. (2001) Changes of Extreme Wave Heights in Northern Hemisphere Oceans and Related Atmospheric Circulation Regimes. Journal of Climate, 14(10), 2204-2221.

Primary method reference (`prewhiten_method = "TFPW_Y"`):

- Yue, S., Pilon, P., Phinney, B. and Cavadias, G. (2002) The influence of autocorrelation on the ability to detect trend in hydrological series. Hydrological Processes, 16(9), 1807-1829. doi:10.1002/hyp.1095

Primary method reference (`prewhiten_method = "TFPW_Z"`):

- Zhang, X., Vincent, L.A., Hogg, W.D. and Niitsoo, A. (2000) Temperature and precipitation trends in Canada during the 20th century. Atmosphere-Ocean, 38(3), 395-429. doi:10.1080/07055900.2000.9649654

Primary method reference (`prewhiten_method = "VCTFPW"`):

- Wang, W., Chen, Y., Becker, S. and Liu, B. (2015) Variance Correction Prewhitening Method for Trend Detection in Autocorrelated Data. Journal of Hydrologic Engineering, 20(12), 04015033. doi:10.1061/(ASCE)HE.19435584.0001234

Primary method reference (`trend_method = "CMK"`, the default):

- Neeti, N. and Eastman, J.R. (2011) A Contextual Mann-Kendall Approach for the Assessment of Trend Significance in Image Time Series. Transactions in GIS, 15(5), 599-611. doi:10.1111/j.14679671.2011.01280.x

Primary method reference (`trend_method = "MMK"`):

- Hamed, K.H. and Rao, A.R. (1998) A modified Mann-Kendall trend test for autocorrelated data. Journal of Hydrology, 204(1-4), 182-196. doi:10.1016/S00221694(97)00125X

Primary method reference (`fdr_method = "BH"`):

- Benjamini, Y. and Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x

Primary method reference (`fdr_method = "BKY"`, the default):

- Benjamini, Y., Krieger, A.M. and Yekutieli, D. (2006) Adaptive Linear Step-Up Procedures that Control the False Discovery Rate. Biometrika, 93(3), 491-507. doi:10.1093/biomet/93.3.491

Primary method reference (`fdr_method = "BY"`, not recommended by default – see the `fdr_method` argument above for why):

- Benjamini, Y. and Yekutieli, D. (2001) The control of the false discovery rate in multiple testing under dependency. Annals of Statistics, 29(4), 1165-1188. doi:10.1214/aos/1013699998

trend_method = "MK", trend_method = "OLS", and slope_method = "TS"/"OLS"/"RM" do have their own foundational references (Mann, 1945 and Kendall, 1975; Legendre, 1805 and Gauss, 1809; Theil, 1950 and Sen, 1968; Siegel, 1982, respectively) – cited in full under ?trend_test and ?slope_estimator rather than repeated here.

See Also

Other pipeline functions: workflow_rta(), workflow_tst()

Examples

```
# Annual mean NDVI from the bundled environmental dataset -- reduced
# to a small window around a cell confirmed to have a complete
# series (found here programmatically, rather than assumed by
# coordinates, which risks landing on a region with no valid
# coverage at all) purely to keep this example fast; the full
# dataset works identically, just with more cells.
r_full <- read_ordered_stack(example_data("vhp_ndvi"))
ok <- stats::complete.cases(terra::values(r_full, mat = TRUE))
rc <- terra::rowColFromCell(r_full, which(ok)[1])
row_lo <- max(1, rc[1] - 10)
row_hi <- min(terra::nrow(r_full), rc[1] + 10)
col_lo <- max(1, rc[2] - 10)
col_hi <- min(terra::ncol(r_full), rc[2] + 10)
r <- terra::crop(r_full, terra::ext(
  terra::xFromCol(r_full, col_lo), terra::xFromCol(r_full, col_hi),
  terra::yFromRow(r_full, row_hi), terra::yFromRow(r_full, row_lo)))

# A combination neither TST nor RTA offers on its own: Yue-Pilon
# prewhitening, classic Mann-Kendall, OLS slope, standard (BH) FDR.
result <- workflow_trends(r, prewhiten_method = "TFPW_Y",
                          trend_method = "MK", slope_method = "OLS",
                          fdr_method = "BH",
                          report = FALSE, verbose = FALSE)
result
plot(result, which = "significance")
plot(result, which = "slope")

# Siegel's repeated median as the slope estimator instead of OLS --
# useful when a dataset is suspected to have enough outliers or
# leverage points that even Theil-Sen's own ~29% breakdown point
# might not fully resist them (see ?slope_estimator's own "RM" entry
# under `method`).
result_rm <- workflow_trends(r, prewhiten_method = "TFPW_Y",
                             trend_method = "MK", slope_method = "RM",
                             fdr_method = "BH",
                             report = FALSE, verbose = FALSE)
plot(result_rm, which = "slope")

# Skipping optional stages entirely -- slope_method/fdr_method both
# NULL leaves $slope/$fdr NULL too, rather than some placeholder
# value, making the modularity concrete rather than just described.
```

```
result_test_only <- workflow_trends(r, prewhiten_method = "none",
                                     trend_method = "MMK",
                                     slope_method = NULL,
                                     fdr_method = NULL,
                                     report = FALSE, verbose = FALSE)
is.null(result_test_only$slope)
is.null(result_test_only$fdr)


## Not run:
# A combination this function warns about: MMK plus prewhitening.
# Not run automatically (unlike the examples above) specifically
# because its only purpose here is to demonstrate the warning
# itself, which would otherwise fire on every R CMD check and every
# example() call.
r <- read_ordered_stack(example_data("vhp_ndvi"))
result_warns <- workflow_trends(r, trend_method = "MMK",
                                 report = FALSE, verbose = FALSE)

## End(Not run)
```

`workflow_tst` *True Significant Trends (TST): the full pipeline in one call*

Description

Implements the complete workflow for robust statistical inference of monotonic trends in gridded raster time series. The main entry point of sptrends: chains `prewhiten()`, `trend_test()`, `slope_estimator()`, and `fdr_correction()`, in that order, into the True Significant Trends (TST) workflow. Each step is optional and all underlying parameters remain available directly on the individual functions – `workflow_tst()` does not replace them, it saves wiring the calls together for the common case, and returns a single `"tst"` object with its own `print()`, `summary()`, and `plot()` methods.

Usage

```
workflow_tst(
  x,
  prewhiten = TRUE,
  prewhiten_args = list(),
  export_dw = FALSE,
  cmk_args = list(),
  theil_sen = TRUE,
  theil_sen_args = list(),
  alpha = 0.05,
  moran_check = FALSE,
  fdr_method = c("BKY", "BH", "BY"),
  q = 0.05,
```

```
  bky_implementation = c("multtest", "original"),
  report = TRUE,
  verbose = TRUE,
  n_cores = 1
)
```

Arguments

x	A terra::SpatRaster; each layer is one time step, in increasing chronological order.
prewhiten	**(Preprocessing)** Logical. If TRUE (default), run prewhiten() first. If FALSE, x is supplied to the trend test unmodified.
prewhiten_args	A named list of extra arguments forwarded to prewhiten() (e.g. list(dw_method = "test")). Workflow-managed transport and reporting arguments cannot be overridden here. **Known limitation with irregular time spacing and TFPW_Y:** this workflow has no top-level t argument of its own – if your observation times are irregularly spaced (not one evenly-spaced step per layer) and you need a genuine time axis for trend_test()/slope_estimator() rather than the default 1:n layer index, you must supply the correct t yourself in prewhiten_args, cmk_args and theil_sen_args separately, and keep them consistent by hand. TFPW_Y specifically drops the first observation (see prewhiten()'s own "Value" section), so the t you pass to cmk_args/theil_sen_args after using TFPW_Y must itself have its own first element dropped to stay aligned with the one-layer-shorter prewhitened series – this workflow does not do that adjustment for you. Not an issue for TFPW_WS (default), which keeps every layer.
export_dw	Logical. If TRUE, include the Durbin-Watson prewhitening diagnostics (diagnostics from prewhiten()) in the return value, even though the trend test itself does not need them. Ignored if prewhiten = FALSE. Default FALSE.
cmk_args	**(Trend detection)** A named list of extra arguments forwarded to trend_test() (e.g. list(window_size = 5L) for a broader CMK region, or list(method = "MK", n_cores = 4)). The default empty list preserves the 3 by 3 CMK region described by Neeti and Eastman (2011), as implemented in TerrSet's Kendall module; changing it creates a TST-inspired variant rather than an exact reproduction of the published workflow. See prewhiten_args above for a known limitation with irregular time spacing and TFPW_Y if you pass t here.
theil_sen	**(Slope estimation)** Logical. If TRUE (default), also compute the Theil-Sen slope (magnitude of change) via slope_estimator() – see "Computational considerations" above before leaving this on for long time series.
theil_sen_args	A named list of extra arguments forwarded to slope_estimator() (e.g. list(max_pairs = 20000, n_cores = 4)). Ignored if theil_sen = FALSE. smooth_neighbourhood stays at slope_estimator()'s own default (FALSE) unless set here (e.g. list(smooth_neighbourhood = TRUE)) – see "Theil-Sen has a real computational cost" above for why. See prewhiten_args above for a known limitation with irregular time spacing and TFPW_Y if you pass t here.
alpha	Numeric vector of significance threshold(s) used for reporting the (uncorrected) trend result – supplied to trend_summary() as alpha, and used as the single

threshold for trend_maps(): 0.05 if it is one of the values in alpha, otherwise the strictest (smallest) value supplied. Defaults to a single 0.05, matching q's own default – a normal analysis picks one alpha and one q, not several at once. Pass a vector explicitly (e.g. alpha = c(0.1, 0.05, 0.01)) to compare several thresholds side by side in trend_summary_table instead. See "Statistical assumptions: alpha and q" above.

moran_check **(Multiple testing)** Logical. Forwarded to fdr_correction() as-is – if TRUE, runs spatial_autocorrelation() on the trend p-value raster as part of the FDR step. Default FALSE (Moran's I stays a separate, deliberate diagnostic – see the package vignette).

fdr_method Character vector supplied as method to fdr_correction(). The Usage displays all supported values: "BKY", "BH" and "BY"; when omitted, "BKY" remains the default originally used in the TST methodology (Gutiérrez-Hernández & García, 2025): it adapts to the estimated proportion of true nulls, gaining statistical power over BH while targeting false-discovery-rate control. Supply one method or a vector of methods; BY is the conservative option for arbitrary dependence. Set to NULL to skip FDR correction entirely.

q Numeric. Target FDR level, forwarded to fdr_correction() – not an adjusted or renamed alpha: it limits the expected false discovery proportion among rejected hypotheses. See fdr_correction() and "Statistical assumptions: alpha and q" above.

bky_implementation
"multtest" (default, unchanged from previous versions) or "original", forwarded to fdr_bky() via fdr_correction() – see fdr_bky()'s own documentation for the difference between the two. Ignored if fdr_method does not include "BKY".

report **(Reporting)** Logical. If TRUE (default), each step prints its own summary table and draws its diagnostic plots as it runs (same effect as calling each function directly with report = TRUE). Set to FALSE for silent, plot-free programmatic use.

verbose Logical. Print progress messages and elapsed time for the complete workflow. Per-stage times are also returned in timing.

n_cores Integer. 1 (default): every step runs sequentially. > 1: builds **one** parallel::makeCluster() PSOCK cluster, shared across every parallel step below (currently the CMK and Theil-Sen steps) instead of each step creating and tearing down its own – avoiding the repeated process-spawn overhead of doing that two or three times in a row for what is, from the caller's own perspective, one parallel request. The cluster is closed automatically when workflow_tst() returns. Setting n_cores inside cmk_args/theil_sen_args directly still works exactly as before (each step falls back to building its own cluster from that value) – but only when this top-level n_cores is left at its default of 1; when both are set, this one wins and the per-step n_cores inside cmk_args/theil_sen_args is ignored, since a single shared cluster and per-step separate ones cannot both apply to the same call.

Details

Unlike many trend-analysis workflows, TST separates preprocessing, hypothesis testing, effect-size estimation, and multiple-testing correction into independent but composable steps. `workflow_tst()` simply orchestrates those steps into a reproducible pipeline without hiding any of their parameters.

Moran's I (`spatial_autocorrelation()`) is **not** part of this pipeline: it is an independent, general spatial diagnostic. Its optional use on inferential fields can reveal dependence relevant to FDR interpretation, but cannot verify all assumptions of an FDR procedure – see the package vignette.

Function type: Core function – the complete published TST workflow.

Value

An object of class `c("tst", "sptrends")` (the second, shared with `workflow_rta()`'s own return value, is for `print()`/`summary()`/ `plot()` – see `print.sptrends()`): a list with

`prewhiten`	The full output of `prewhiten()`, or `NULL` if `prewhiten = FALSE`.
`dw_diagnostics`	The prewhitening `diagnostics` raster, only if `export_dw = TRUE`.
`trend`	The `$stats` field of `trend_test()`'s own output (a `SpatRaster` with the trend statistics themselves) – not the full `trend_test()` return value, which also includes `neighbourhood`/`window_size`.
`trend_summary_table`	The output of `trend_summary()` (invisible data frame).
`theil_sen`	The `$slope` field of `slope_estimator()`'s own output (a `SpatRaster`), or `NULL` if `theil_sen = FALSE`. Not the full `slope_estimator()` return value. Not smoothed by default (see the `theil_sen_args` note above) – check `theil_sen_smoothed` if you need to know whether it was (e.g. because you set it yourself).
`theil_sen_smoothed`	Logical: whether `theil_sen` was computed with `smooth_neighbourhood = TRUE`. `FALSE` by default; `NULL` if `theil_sen = FALSE`.
`fdr`	The output of `fdr_correction()`, or `NULL` if `fdr_method = NULL`.
`timing`	A named list of elapsed seconds per step actually run (`prewhiten`, `CMK`, `theil_sen`, `fdr` – only the ones that ran are present), each measured with `Sys.time()` around that step's own function call. Coarse (whole-step, not line-by-line) and not a substitute for a proper profiler, but enough for noticing which step dominates on your own data.

Use `print()` for a one-line summary, `summary()` for the full detail, and `plot()` for a map – see `plot.sptrends()`.

Typical use

```
raster time series
    |
workflow_tst()
    |
selective prewhitening -> CMK -> Theil-Sen -> adaptive FDR
    |
one `tst` result containing every stage
```

For seasonal input, first use compute_anomalies() and pass its anomalies raster. For long series, consider setting max_pairs through theil_sen_args; see "Computational considerations" below.

Methodological details

How it works.

```
Input raster
    |
(optional) prewhiten          -- removes serial autocorrelation
    |
trend_test                    -- is there a monotonic trend? (CMK)
    |
(optional) slope_estimator    -- how fast? (Theil-Sen or OLS)
    |
(optional) FDR correction     -- which cells survive multiple testing?
    |
"tst" object
```

The four steps run in this fixed order because each one's own assumptions depend on what came before it: prewhitening needs to happen before the trend test, since the test assumes independent observations; the slope is estimated on the same (optionally prewhitened) series the test itself used, so the two describe the same data; and FDR correction needs the test's own p-values to exist first. Prewhitening, slope estimation, and FDR correction are each individually optional – not every analysis needs all three (a quick exploratory look at significance alone might skip slope and FDR entirely; data already known to have negligible serial correlation might skip prewhitening) – but the trend test itself is not optional, since every other step either feeds into it or consumes its output. The returned object keeps every intermediate result it computed (see "Value" below), not only the final one, so that any step's own output can be inspected or reused without recomputing the whole pipeline.

Statistical assumptions: alpha and q. This is one of the most common mistakes in gridded trend analysis, so it is worth spelling out: alpha (e.g. 0.05) is defined for a *single* hypothesis test. A raster is not one test – it is one test per cell, potentially thousands of them run simultaneously. Applying alpha cell by cell, as if each cell were the only test being run, is exactly the multiple-testing error this package's FDR step exists to fix (see the "Warning" section of ?trend_test); it is not a harmless simplification.

It is common practice to set q equal to alpha (e.g. both 0.05) for convenience and comparability between the uncorrected and corrected results reported here – this function does not enforce that, alpha and q are independent arguments. But equal values does not mean equal meaning: alpha bounds the error rate of each individual cell's test, while q bounds the expected proportion of false positives among all the cells called significant after correction (see fdr_correction() for the full distinction). Do not read trend_summary_table (based on alpha, uncorrected) and the FDR results (based on q) as answering the same question just because the numbers match. Unlike alpha, which this function reports at three conventional levels for context (see alpha below), q = 0.05 has little reason to change: it is the standard target FDR level in the literature this package builds on, and lowering it (e.g. to 0.01) mainly costs statistical power rather than offering a meaningfully different guarantee.

Statistical assumptions: monotonicity and seasonality. Every step here (prewhitening, the Contextual Mann-Kendall test, Theil-Sen) is designed around a **monotonic** trend – a consistent tendency to increase or decrease – not a periodic/seasonal cycle. If `x` has a seasonal cycle (e.g. raw monthly data with an annual signal), remove it first with `compute_anomalies()` and pass the anomalies to `workflow_tst()`, not the raw seasonal series.

Computational considerations. Unlike the Mann-Kendall S statistic, the Theil-Sen slope needs every pairwise slope in the series (`n*(n-1)/2` per cell) to take their median – it cannot be accumulated as a running sum, so it does not scale the same way. For short series (a few dozen time steps) this is unnoticeable; for long series (hundreds to thousands of steps, e.g. multi-decade monthly data) it can become the slowest step in the whole workflow. `theil_sen = TRUE` by default, to match the published TST workflow, but for long series consider `theil_sen = FALSE`, or tune `theil_sen_args` (`max_pairs` to subsample pairs, `n_cores` to parallelise) – see `slope_estimator()`. `smooth_neighbourhood` stays at `slope_estimator()`'s own default (`FALSE`) unless you set it yourself via `theil_sen_args = list(smooth_neighbourhood = TRUE)`: that mechanism is not part of the published TST method (neither TST nor RTA originally included any such smoothing – it is this package's own optional addition on top of both), so there is no principled reason for `workflow_tst()` and `workflow_rta()` to default to different values for it – see `?slope_estimator`'s "Optional queen-neighbourhood smoothing" section for exactly what it does and why it is off by default.

Limitations. The workflow targets monotonic trends and does not detect abrupt breaks, periodicity, or general nonlinear change. Optional stages permit exploratory variants, but only the complete default configuration reproduces the published TST workflow.

Quality assurance. Each component is validated independently in its own function. Workflow-level tests additionally verify stage order, optional prewhitening, argument forwarding, shared-cluster behaviour, sequential/parallel equivalence, timings, S3 structure, summaries, plots, and propagation of raw and FDR-corrected results. See `?sptrends` for the internal release protocol and external numerical controls.

References

Source of this workflow (primary reference – this pipeline is a direct implementation of it):

- Gutiérrez-Hernández, O. and García, L.V. (2025) Uncovering true significant trends in global greening. Remote Sensing Applications: Society and Environment, 37, 101377. doi:10.1016/j.rsase.2024.101377

Step 1, selective AR(1) prewhitening (see `prewhiten()` for the full reference list, including the Durbin-Watson gate):

- Wang, X.L. and Swail, V.R. (2001) Changes of Extreme Wave Heights in Northern Hemisphere Oceans and Related Atmospheric Circulation Regimes. Journal of Climate, 14(10), 2204-2221.

Step 2, Contextual Mann-Kendall (see `trend_test()` for the full reference list, including the foundational Mann-Kendall statistic it builds on):

- Neeti, N. and Eastman, J.R. (2011) A Contextual Mann-Kendall Approach for the Assessment of Trend Significance in Image Time Series. Transactions in GIS, 15(5), 599-611. doi:10.1111/j.14679671.2011.01280.x

Step 3, Theil-Sen slope (see `slope_estimator()` for the full reference list):

- Theil, H. (1950) A rank-invariant method of linear and polynomial regression analysis. Indagationes Mathematicae, 12, 85-91 (Part I; published in three parts). No DOI available (pre-DOI-era publication).
- Sen, P.K. (1968) Estimates of the regression coefficient based on Kendall's tau. Journal of the American Statistical Association, 63, 1379-1389. doi:10.1080/01621459.1968.10480934

Step 4, FDR correction – BKY (default) and BH (see `fdr_bky()` and `fdr_correction()` for the full reference lists):

- Benjamini, Y., & Hochberg, Y. (1995) Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing. Journal of the Royal Statistical Society: Series B, 57, 289-300. doi:10.1111/j.25176161.1995.tb02031.x
- Benjamini, Y., Krieger, A. M., & Yekutieli, D. (2006) Adaptive Linear Step-Up Procedures that Control the False Discovery Rate. Biometrika, 93(3), 491-507. doi:10.1093/biomet/93.3.491

See Also

Other pipeline functions: `workflow_rta()`, `workflow_trends()`

Examples

```
# Annual mean NDVI from the bundled environmental dataset.
r <- read_ordered_stack(example_data("vhp_ndvi"))

# Run the full workflow: prewhiten -> Contextual Mann-Kendall ->
# Theil-Sen -> FDR-BKY (only BKY, not BH -- see the "fdr_method"
# argument below for why).
result <- workflow_tst(r, report = FALSE, verbose = FALSE)

# A "tst" object: printing it gives a one-line-per-step summary
# (cells modified by prewhitening, the trend test's cell count, the
# Theil-Sen slope range, and how many cells are significant after
# FDR-BKY correction).
result

# Three reports worth seeing: how much of the map is significant at
# all, how fast the significant cells are changing, and which way.
plot(result, which = "significance")
plot(result, which = "slope")
plot(result)
```

Index