Package 'ggchangepoint'

September 25, 2026

Type Package

Title Combines Changepoint Analysis with 'ggplot2'

Version 0.5.0

Description A unified, tidy, 'ggplot2'-native interface to changepoint detection in R. Provides the 'ggcpt' S3 result class with 'broom'-style tidy/glance/augment methods, 'autoplot()' (with confidence intervals, significance regions, detector statistics, solution paths, scale space, and multivariate facets), composable geoms ('geom_changepoint()', 'geom_cpt_segment()', 'geom_cpt_ci()', 'geom_cpt_region()', 'geom_cpt_label()', 'geom_cpt_event()', 'stat_changepoint()'), and a 'cpt_detect()' dispatcher covering fifty methods with introspection via 'cpt_methods()': penalised/optimal partitioning, multiscale and search methods, nonparametric and kernel methods, Bayesian methods, high-dimensional, functional, covariance and network methods, regression breaks, seasonal-trend decomposition, the classical single-change tests, and robust detection under drift and autocorrelation. Adds inference (Narrowest Significance Pursuit regions, a unified 'cpt_confint()', post-detection tests), selection of the number of changes, influence and sensitivity diagnostics, supervised detection with learned penalties, consensus and method recommendation, event annotation and reproducible reports, benchmarks against the Turing Change Point Dataset, sequential monitoring with detection-delay accounting, power and study design, and an extension mechanism ('as_ggcpt()', 'cpt_register_method()') that brings external and non-CRAN detectors into the same grammar.

Depends R (>= 4.0.0)

License GPL (>= 3)

Encoding UTF-8

Imports changepoint, changepoint.np, dplyr, ecp, generics, ggplot2 (>= 3.5.0), lifecycle, Rdpack, stats, tibble, tools, utils

RdMacros Rdpack

RoxygenNote 7.3.2

Suggests rmarkdown, knitr, testthat (>= 3.1.7), withr, rjags, vdiffr, svglite, wbs, breakfast, not, mosum, fpop, IDetect, stepR, cpop, bcp, ocp, Rbeast, cpm, kcpRS, CptNonPar, DeCAFS, SNSeg, InspectChangepoint, ocd, changepoint.geo, strucchange, segmented, EnvCpt, fastcpd, nsp, mcp, HDCD, changepoints, fChange, KWCChangepoint, fabisearch, NMF, wbsts, bfast, trend, ChangePointTaylor, binsegRcpp, crossvalidationCP, changepoint.influence, penaltyLearning, zoo, xts, tsibble, jsonlite, gt, ggrepel, plotly, ggiraph, progressr, future, future.apply

Additional_repositories https://R-Forge.R-project.org

Config/testthat/edition 3

VignetteBuilder knitr

URL https://pursuitofdatascience.github.io/ggchangepoint/

BugReports https://github.com/PursuitOfDataScience/ggchangepoint/issues

Config/Needs/website rmarkdown

NeedsCompilation no

Author Youzhi Yu [aut, cre]

Maintainer Youzhi Yu <yuyouzhi666@icloud.com>

Repository CRAN

Date/Publication 2026-09-25 21:00:02 UTC

Contents

alarms . . . 5
annotate_segments . . . 6
as_cpt_labels . . . 7
as_cpt_series . . . 7
as_ggcpt . . . 8
augment.ggcpt . . . 10
autoplot.ggcpt . . . 11
bcp_wrapper . . . 13
beast_wrapper . . . 14
bfast_wrapper . . . 15
binsegrcpp_wrapper . . . 17
bocpd_wrapper . . . 18
cpm_wrapper . . . 19
cpop_wrapper . . . 20
cpt_annotate_events . . . 22
cpt_annotations . . . 23
cpt_batch . . . 24
cpt_benchmark . . . 26
cpt_cite . . . 28

cpt_confint . . . 29
cpt_consensus . . . 31
cpt_crops . . . 33
cpt_datasets . . . 34
cpt_delay . . . 35
cpt_detect . . . 37
cpt_gt . . . 39
cpt_influence . . . 40
cpt_install_engines . . . 42
cpt_labels . . . 43
cpt_label_error . . . 44
cpt_label_error_curve . . . 45
cpt_learn_penalty . . . 47
cpt_leverage . . . 49
cpt_load_tcpd . . . 50
cpt_methods . . . 51
cpt_metrics . . . 52
cpt_metrics_annotated . . . 54
cpt_min_detectable . . . 55
cpt_monitor . . . 57
cpt_penalty . . . 60
cpt_power . . . 61
cpt_recommend . . . 64
cpt_regions . . . 65
cpt_register_method . . . 66
cpt_replay . . . 68
cpt_report . . . 69
cpt_scale_space . . . 70
cpt_scenarios . . . 71
cpt_select . . . 72
cpt_sensitivity . . . 75
cpt_simulate . . . 76
cpt_solution_path . . . 78
cpt_stability . . . 79
cpt_statistic . . . 80
cpt_test . . . 81
cpt_update . . . 82
cpt_wrapper . . . 83
decafs_wrapper . . . 85
ecp_wrapper . . . 86
envcpt_wrapper . . . 87
esac_wrapper . . . 88
fabisearch_wrapper . . . 90
fastcpd_wrapper . . . 92
fcov_wrapper . . . 93
fmean_wrapper . . . 96
fpop_wrapper . . . 97
geomcp_wrapper . . . 98

geom_changepoint . . . 100
geom_cpt_ci . . . 101
geom_cpt_event . . . 102
geom_cpt_label . . . 103
geom_cpt_region . . . 104
geom_cpt_segment . . . 106
ggcptplot . . . 107
ggcpt_compare . . . 108
ggcpt_compare_table . . . 109
ggcpt_eval . . . 110
ggcpt_interactive . . . 111
ggcpt_methods . . . 112
ggcpt_plot_methods . . . 113
ggcpt_posterior . . . 115
ggcpt_runlength . . . 115
ggecpplot . . . 116
glance.ggcpt . . . 117
hdcov_wrapper . . . 118
hdreg_wrapper . . . 120
idetect_wrapper . . . 121
inspect_wrapper . . . 122
is_ggcpt . . . 124
kcp_wrapper . . . 124
kwc_wrapper . . . 126
mcp_wrapper . . . 127
mosum_wrapper . . . 129
network_wrapper . . . 130
new_ggcpt . . . 132
not_wrapper . . . 134
npmojo_wrapper . . . 135
nsp_wrapper . . . 137
ocd_wrapper . . . 139
pilliat_wrapper . . . 141
print.ggcpt . . . 142
scale_colour_cpt . . . 143
scale_fill_cpt_label . . . 144
segmented_wrapper . . . 145
signal_blocks . . . 146
signal_fms . . . 147
signal_mix . . . 148
signal_stairs . . . 148
signal_teeth . . . 149
smuce_wrapper . . . 150
sn_wrapper . . . 151
stat_changepoint . . . 153
strucchange_wrapper . . . 154
summary.ggcpt . . . 155
taylor_wrapper . . . 156

tguh_wrapper 158
theme_ggcpt 159
tidy.ggcpt 160
trend_wrapper 160
var_wrapper 162
wbs2_wrapper 163
wbsts_wrapper 164
wbs_wrapper 166

Index **168**

`alarms` *The alarm log of a monitor*

Description

Every observation at which a sequential monitor crossed its threshold, in the order they fired. A monitor that never fired returns a zero-row tibble rather than `NULL`, so the result is always safe to `rbind()` or plot.

Usage

```
alarms(x, ...)

## S3 method for class 'ggcpt_monitor'
alarms(x, ...)
```

Arguments

`x`	A `ggcpt_monitor` object.
`...`	Ignored.

Value

A tibble with one row per alarm: `time` (the observation at which it fired, counted from the first one fed to the monitor), `statistic` and `threshold`.

See Also

`cpt_monitor()`, `cpt_delay()`.

Examples

```
set.seed(2026)
mon <- cpt_monitor("edetector", baseline = rnorm(100))
mon <- cpt_update(mon, c(rnorm(50), rnorm(50, 3)))
alarms(mon)
```

`annotate_segments` *Annotate segments with alternating shading*

Description

Adds alternating shaded rectangles to highlight segments between changepoints.

Usage

```
annotate_segments(cp, n, fill = c("grey90", "white"), alpha = 0.5, ...)
```

Arguments

`cp`	Changepoint indices (including 0 and n).
`n`	Length of the series.
`fill`	Colors for alternating segments. Defaults to c("grey90", "white").
`alpha`	Alpha for fill. Defaults to 0.5.
`...`	Additional arguments passed to `annotate`.

Value

A list of ggplot annotations.

See Also

Other result class: `as_cpt_series()`, `as_ggcpt()`, `cpt_annotations()`, `is_ggcpt()`, `new_ggcpt()`, `print.ggcpt()`

Examples

```
library(ggplot2)
set.seed(2026)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 4)), method = "pelt")
ggplot(fit$data, aes(index, value)) +
  annotate_segments(fit$changepoints$cp, n = nrow(fit$data)) +
  geom_line()
```

as_cpt_labels *Coerce annotations to changepoint labels*

Description

Turns a plain ground-truth changepoint set (the kind `cpt_metrics()` takes) into labelled regions, so one annotation can drive both the metric and the supervised machinery. Each true changepoint becomes a `"one_change"` region of width `2 * margin + 1`, and the stretches between them become `"no_change"` regions.

Usage

```
as_cpt_labels(truth, n, margin = 5, negatives = TRUE, series = NA_character_)
```

Arguments

`truth`	Integer vector of true changepoint positions.
`n`	Series length.
`margin`	Half-width of the positive regions. Defaults to `5`, matching `cpt_metrics()`'s default tolerance.
`negatives`	Add the `"no_change"` regions between the positives? Defaults to `TRUE`; without them a detector is never penalised for a false positive.
`series`	Optional series identifier.

Value

A `cpt_labels` tibble.

Examples

```
as_cpt_labels(c(50, 120), n = 200)
```

as_cpt_series *Coerce a time series object to values plus a time index*

Description

Detection itself runs on positions (every wrapped engine assumes an equally spaced sequence), but real series carry dates, and reporting a changepoint as "index 147" when the data are monthly rainfall is an unnecessary translation step for the user. `as_cpt_series()` is the one place that separates the two: it pulls the numeric values out of a `ts`, `xts`, `zoo`, `tsibble` or data frame and returns the time index alongside them, so `cpt_detect()` can detect on positions and report on dates.

Usage

```
as_cpt_series(x, index = NULL, check_regular = TRUE)
```

Arguments

x	A numeric vector or matrix, or a ts/mts, xts, zoo or tsibble object. The xts, zoo and tsibble paths need those packages installed (they are Suggests).
index	Optional explicit index, one value per observation. Overrides any index carried by x, and is the way to attach dates to a plain numeric vector.
check_regular	Warn when the index is not equally spaced? Defaults to TRUE. Every engine in the package assumes equal spacing, so an irregular index means the positions the engine sees are not the times the user means.

Value

A list with components values (a numeric vector, or a matrix for multivariate input), index (the time index, or NULL when there is none), index_label (a name for the x axis) and frequency (the seasonal frequency a ts carried, or NULL; cpt_detect() hands this to the engines that need one, such as bfast_wrapper()).

See Also

Other result class: annotate_segments(), as_ggcpt(), cpt_annotations(), is_ggcpt(), new_ggcpt(), print.ggcpt()

Examples

```
as_cpt_series(1:10)$index
s <- as_cpt_series(1:10, index = as.Date("2020-01-01") + 0:9)
s$index
```

---

as_ggcpt *Turn external changepoints into a ggcpt result*

Description

Wraps a set of changepoint locations (from a detector this package does not wrap, a Python tool called through **reticulate**, a neural detector, a published paper's reported breaks, or an analyst's own annotations) into a first-class ggcpt object. Everything built on the ggcpt contract then applies: autoplot(), the composable geoms, tidy()/glance()/augment(), cpt_metrics(), cpt_consensus(), cpt_report().

Usage

```
as_ggcpt(
  cp,
  x,
  fitted = NULL,
  method = "custom",
  change_in = "mean",
  ci = NULL,
  regions = NULL,
  penalty = NULL,
  cp_convention = c("left", "right"),
  index = NULL,
  fit = NULL,
  extra = NULL
)
```

Arguments

cp	Integer vector of changepoint locations. Out-of-range, duplicated and missing values are dropped and the result is sorted, the same contract every built-in wrapper is held to. But unlike a wrapper, which is normalising an engine's output, this function is given yours, so **anything it drops it warns about**, with the values and the range they had to fall in. A fractional index is included in that: it is truncated rather than rounded, which makes `50.5` into a changepoint at 50.
x	The series the changepoints refer to: a numeric vector, or a matrix/data frame (rows are time points) for a multivariate result. A one-column matrix or data frame is read as a univariate series.
fitted	Optional length-n fitted signal, used by `autoplot(show_fit = TRUE)` and `augment()`.
method	Method label. Defaults to `"custom"`.
change_in	What the changepoints are changes in. Defaults to `"mean"`.
ci	Optional two-column matrix or data frame of location confidence intervals, one row per changepoint, giving lower and upper bounds as positions in the same `cp_convention` as `cp`: with `"right"` they are converted along with `cp`, so an interval keeps bracketing its changepoint.
regions	Optional two-column matrix or data frame of significance regions (`start`, `end`); see `nsp_wrapper()` for the interval-valued case this exists to serve. A region with a missing bound is dropped with a warning.
penalty	Optional penalty descriptor: a number, a string, or a list with `type` and `value`.
cp_convention	`"left"` (the changepoint is the last index of the left segment, this package's convention) or `"right"` (the first index of the right segment, which is converted on the way in).
index	Optional time index, one value per observation.
fit	Optional raw upstream object to carry along.
extra	Optional named list of per-changepoint columns, each of the same length as `cp`, appended to the changepoints tibble.

Value

A ggcpt object, with the same components as any detector in the package returns (see new_ggcpt() for the full list, and cpt_detect() for the summary).

See Also

cpt_register_method() to make cpt_detect() dispatch to an external detector by name.

Other result class: annotate_segments(), as_cpt_series(), cpt_annotations(), is_ggcpt(), new_ggcpt(), print.ggcpt()

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 4))
fit <- as_ggcpt(c(60), x, method = "my_detector")
fit
tidy(fit)
```

---

augment.ggcpt *Augment a ggcpt object*

Description

Returns the original data with added columns: seg_id, .fitted, .resid, and is_changepoint.

Usage

```
## S3 method for class 'ggcpt'
augment(x, ...)
```

Arguments

x	A ggcpt object.
...	Additional arguments (ignored).

Details

For a multivariate result every coordinate is returned, but the changepoints are shared across them, so seg_id and is_changepoint apply to the whole row while .fitted and .resid describe the **univariate series the result carries**: $data$value, the same series $segments$param_estimate summarises, so .resid is always value - .fitted. For most multivariate engines that series is the first coordinate; fmean, fcov, kwc and fabisearch store the cross-sectional mean rowMeans() instead, and for those .fitted/.resid describe that mean rather than any one column. Either way the two columns agree with each other, which is what makes .resid a residual. When an engine supplies its own fitted signal that signal is used for .fitted in place of the segment means, and rides along in a fitted column of its own, so for those engines the two columns agree. The engines that do this are exactly the ones cpt_methods() marks in its fitted column: smuce, hsmuce, cpop, bcp, beast, decafs, segmented, mcp and bfast.

Value

A tibble with one row per observation: the data as the result carries it, plus four added columns. The data half depends on the result: `index` and `value` for a univariate one, `index` plus **one column per coordinate** (named as the input's columns were) for a multivariate one, and an extra `fitted` column for the engines that supply their own fitted signal. The added four are always the same:

`seg_id` which segment the observation falls in, counting from 1.

`.fitted` the segment's `param_estimate`: the segment **mean**, for every method in the package, or the engine's own fitted signal where there is one. See the details below for the multivariate case.

`.resid` `value - .fitted`, against the univariate series the result carries.

`is_changepoint` `TRUE` at each detected location, under the result's `cp_convention`.

Measured on a `pelt` fit the columns are `index`, `value`, `seg_id`, `.fitted`, `.resid`, `is_changepoint`.

`autoplot.ggcpt` *Autoplot a ggcpt object*

Description

Renders a changepoint detection result as a ggplot. The raw series is drawn as a line (with optional points), changepoints are shown as vertical lines, and (optionally) fitted segment levels, the engine's fitted signal, and changepoint-location confidence intervals are overlaid. Multivariate results (from `ecp`, `inspect`, `geomcp`, ...) are drawn as faceted small-multiples with shared changepoint rules.

Usage

```
## S3 method for class 'ggcpt'
autoplot(
  object,
  show_segments = FALSE,
  show_ci = FALSE,
  show_fit = FALSE,
  show_regions = NULL,
  cptline_alpha = 1,
  cptline_color = "blue",
  cptline_type = "solid",
  cptline_linewidth = 0.5,
  show_points = NULL,
  show_line = TRUE,
  index = NULL,
  labels = NULL,
  type = c("series", "statistic", "path", "scale_space"),
  ...
)
```

Arguments

object	A ggcpt object.
show_segments	Logical. Whether to draw the fitted segment means. Defaults to FALSE.
show_ci	Logical. Whether to draw confidence intervals for changepoint locations, when the engine provides them (columns ci_lower/ci_upper on the changepoints tibble, supplied by the engines cpt_methods() marks in its ci column: smuce, hsmuce, strucchange, segmented, mcp, bfast and taylor). nsp is marked there as well but is not drawn by this argument: it reports region_start/region_end rather than an interval around an estimate, which show_regions draws. Drawn as horizontal whiskers near the bottom of the panel. Defaults to FALSE.
show_fit	Logical. Whether to draw the engine's fitted signal (the fitted column of $data, provided by the engines cpt_methods() marks in its fitted column: smuce, hsmuce, cpop, bcp, beast, decafs, segmented, mcp and bfast). Defaults to FALSE.
show_regions	Logical. Whether to shade the significance regions an interval-valued method returns (the regions slot; currently nsp_wrapper()). Each band is an interval that contains at least one changepoint at the stated global level; it is not a confidence interval around a point estimate. Defaults to TRUE when the result carries regions, and is ignored otherwise.
cptline_alpha	Alpha for changepoint lines. Defaults to 1.
cptline_color	Color for changepoint lines. Defaults to "blue".
cptline_type	Linetype for changepoint lines. Defaults to "solid".
cptline_linewidth	Linewidth for changepoint lines. Defaults to 0.5.
show_points	Logical. Whether to draw data points. Auto-off above 500 obs.
show_line	Logical. Whether to draw the line. Defaults to TRUE.
index	Optional vector of x-axis values (e.g. dates) of the same length as the series. Defaults to the time index carried by the result (see the index argument of cpt_detect()), and to the observation position when there is none.
labels	Optional cpt_labels() tibble. When supplied, the labelled regions are shaded behind the series and coloured by the outcome cpt_label_error() gives them (correct, false positive, false negative), so scoring a segmentation against expert labels becomes a picture rather than a table. Not drawn for a multivariate result, which warns instead.
type	Which view to draw. "series" (default) is the series with its changepoints; "statistic" and "path" delegate to ggcpt_statistic() and ggcpt_solution_path(), which error with the list of supporting engines when this one does not expose the internals. "scale_space" delegates to ggcpt_scale_space(), which needs nothing from the engine: it sweeps a multiscale detector over the same series, so it draws for any result.
...	Unknown arguments are ignored with a warning, except when type is not "series", in which case they are passed to the delegate.

Value

A ggplot object.

See Also

ggcpt_plot_methods for the base plot() methods that delegate here, and theme_ggcpt() and geom_changepoint() for restyling and re-assembling the layers by hand.

---

bcp_wrapper *Bayesian changepoint wrapper (Barry-Hartigan product partition model)*

Description

Wraps bcp::bcp(), the MCMC implementation (Erdman and Emerson, 2007) of the Barry and Hartigan (1993) product partition model. The engine returns a *posterior probability of a change-point at every location*; locations whose posterior probability reaches prob_threshold are reported as changepoints, and the full probability profile is kept so that ggcpt_posterior() can draw the classic two-panel posterior plot.

Usage

```
bcp_wrapper(x, prob_threshold = 0.5, burnin = 50, mcmc = 500, seed = NULL, ...)
```

Arguments

x	A numeric vector.
prob_threshold	Posterior probability cutoff in $(0, 1)$ above which a location is reported as a changepoint. Defaults to 0.5.
burnin	Number of burn-in MCMC iterations. Defaults to 50.
mcmc	Number of post-burn-in MCMC iterations. Defaults to 500.
seed	Optional seed for reproducibility of the MCMC run. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to bcp::bcp().

Value

A ggcpt object. The changepoints tibble carries a posterior_prob column, and the data tibble carries the posterior mean in its fitted column.

References

Barry D, Hartigan JA (1993). "A Bayesian analysis for change point problems." *Journal of the American Statistical Association*, **88**(421), 309–319.

Erdman C, Emerson JW (2007). "bcp: An R package for performing a Bayesian analysis of change point problems." *Journal of Statistical Software*, **23**(3), 1–13.

See Also

Other changepoint engines: beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
res <- bcp_wrapper(c(rnorm(60), rnorm(60, 4)), seed = 2026)
res$changepoints
ggcpt_posterior(res)
```

---

beast_wrapper *BEAST wrapper: Bayesian estimation of abrupt change, seasonality, and trend*

Description

Wraps `Rbeast::beast()` (Zhao et al., 2019), a Bayesian model-averaging ensemble that estimates the number and location of trend changepoints together with their posterior occurrence probabilities. Locations whose posterior probability reaches `prob_threshold` are reported; the probability profile renders via ggcpt_posterior().

Usage

```
beast_wrapper(x, prob_threshold = 0.5, seed = NULL, ...)
```

Arguments

`x`	A numeric vector (treated as a non-seasonal series).
`prob_threshold`	Posterior probability cutoff in $(0, 1)$ above which a candidate trend changepoint is reported. Defaults to `0.5`.
`seed`	Optional seed for the engine's MCMC sampler (passed to `Rbeast::beast()` as `mcmc.seed`). The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`...`	Additional arguments passed to `Rbeast::beast()`.

Value

A ggcpt object. The `changepoints` tibble carries `posterior_prob`, and the `data` tibble carries the posterior mean trend in its `fitted` column.

References

Zhao K, Wulder MA, Hu T, Bright R, Wu Q, Qin H, Li Y, Toman E, Mallick B, Zhang X, Brown M (2019). "Detecting change-point, trend, and seasonality in satellite time series data to track abrupt changes and nonlinear dynamics: A Bayesian ensemble algorithm." *Remote Sensing of Environment*, **232**, 111181.

See Also

Other changepoint engines: bcp_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
# try(): Rbeast intermittently returns an all-NaN fit and the condition
# can persist for a session, so a check must not fail on it -- the
# wrapper reports it by name when it happens.
res <- try(beast_wrapper(c(rnorm(60), rnorm(60, 4)), seed = 2026),
           silent = TRUE)
if (!inherits(res, "try-error")) res$changepoints
```

---

bfast_wrapper *BFAST wrapper: breaks for additive season and trend*

Description

Wraps the **bfast** family (Verbesselt et al.), the standard tool in remote sensing and land-cover monitoring. BFAST decomposes a seasonal series into trend and seasonal components and detects breaks in each separately, which is the right question for satellite time series where a shift in phenology and a shift in level mean different things.

Usage

```
bfast_wrapper(
  x,
  frequency = 12,
  change_in = c("mean", "slope", "seasonality"),
  h = 0.15,
  season = c("harmonic", "dummy", "none"),
  max_iter = 5,
  ...
)
```

Arguments

x	A numeric vector, or a ts; a ts is strongly preferred, because BFAST needs the seasonal frequency and cannot guess it. A bare vector is turned into a ts with frequency.
frequency	Observations per season, used when x carries none. Defaults to 12. cpt_detect() reads the frequency off a ts and passes it here, so the two calls agree; the panel helpers (cpt_batch(), ggcpt_compare(), cpt_benchmark()) normalise their input to bare numeric vectors before dispatching, so name frequency explicitly there.
change_in	Which component's breaks to report: "mean" or "slope" give the trend breaks (the usual choice), "seasonality" gives the seasonal ones.
h	Minimal segment size as a fraction of the series. Defaults to 0.15.
season	Seasonal model: "harmonic" (default), "dummy" or "none".
max_iter	Maximum iterations of the trend/season loop. Defaults to 5.
...	Additional arguments passed to bfast::bfast().

Value

A ggcpt object whose fitted column holds the estimated trend component (so autoplot(show_fit = TRUE) draws it) and whose changepoints carry ci_lower/ci_upper from **strucchange**'s break-date intervals when available.

References

Verbesselt J, Hyndman R, Newnham G, Culvenor D (2010). "Detecting trend and seasonal changes in satellite image time series." *Remote Sensing of Environment*, **114**(1), 106–115. doi:10.1016/j.rse.2009.08.014.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
season <- rep(sin(seq(0, 2 * pi, length.out = 12)), 10)
y <- stats::ts(c(rnorm(60, 1), rnorm(60, 5)) + season,
               frequency = 12, start = c(2000, 1))
bfast_wrapper(y)
```

`binsegrcpp_wrapper` *Fast binary segmentation across loss functions*

Description

Wraps `binsegRcpp::binseg()` (Hocking): a C++ binary segmentation that runs in $O(n \log n)$ for the best case and supports several loss functions, including ones no other engine here offers (Poisson, ℓ_1). It is the package's *performance* path for binary segmentation on long series, and it returns the whole nested family of segmentations, so it also feeds `cpt_solution_path()`.

Usage

```
binsegrcpp_wrapper(
  x,
  change_in = c("mean", "meanvar"),
  distribution = NULL,
  max_segments = NULL,
  n_segments = NULL,
  min_segment_length = NULL
)
```

Arguments

`x`	A numeric vector.
`change_in`	`"mean"` (Gaussian, the default) or `"meanvar"`, mapped to the engine's `mean_norm` and `meanvar_norm` distributions. **binsegRcpp** has no variance-only cost, so `"var"` is not offered here; use `"meanvar"`, or `cpt_detect(method = "pelt", change_in = "var")` for a variance-only change.
`distribution`	Loss function, overriding the mapping from `change_in`. Run `binsegRcpp::get_distribution_info()` for the list (`"mean_norm"`, `"meanvar_norm"`, `"poisson"`, `"l1"`, ...).
`max_segments`	Largest number of segments searched. Defaults to `min(20, floor(n / 5))`.
`n_segments`	Number of segments to report. When `NULL` (the default) it is chosen by BIC over the nested family the engine returns.
`min_segment_length`	Minimum segment length. Passed through when supplied.

Value

A `ggcpt` object.

References

Hocking TD (2024). "Finite sample complexity analysis of binary segmentation." *arXiv preprint arXiv:2410.08654*. doi:10.48550/arXiv.2410.08654.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
binsegrcpp_wrapper(c(rnorm(100), rnorm(100, 3)))
```

---

bocpd_wrapper *Bayesian online changepoint detection wrapper (BOCPD)*

Description

Wraps ocp::onlineCPD(), an implementation of Bayesian Online Changepoint Detection (Adams and MacKay, 2007). BOCPD recursively updates a posterior over the current *run length* (time since the last change); the maximum a posteriori set of changepoints is reported, and the full run-length posterior is kept so that ggcpt_runlength() can draw the signature run-length heatmap.

Usage

```
bocpd_wrapper(x, hazard = 100, ...)
```

Arguments

x	A numeric vector.
hazard	The λ of **ocp**'s constant hazard: the expected run length between changes, so the hazard rate itself is $1/\lambda$ and larger values mean changes are expected less often. Defaults to 100 (the upstream default).
...	Additional arguments passed to ocp::onlineCPD().

Value

A ggcpt object with the MAP changepoint set. The full ocp fit (including the run-length posterior) is kept in $fit.

References

Adams RP, MacKay DJ (2007). "Bayesian online changepoint detection." *arXiv preprint arXiv:0710.3742*.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
res <- bocpd_wrapper(c(rnorm(60), rnorm(60, 4)))
res$changepoints
ggcpt_runlength(res)
```

---

cpm_wrapper *Sequential change point model wrapper (CPM)*

Description

Wraps `cpm::processStream()` (Ross, 2015) for distribution-free sequential changepoint detection via repeated two-sample tests (Mann-Whitney for location, Mood for scale, Lepage, Kolmogorov-Smirnov and Cramer-von-Mises for general changes, and parametric Student/Bartlett/GLR variants). Although the engine is designed for streams, it is run here over the full series in one pass, mimicking online monitoring with average run length `arl0`.

Usage

```
cpm_wrapper(x, cpm_type = "Mann-Whitney", arl0 = 500, startup = 20, ...)
```

Arguments

x	A numeric vector.
cpm_type	Test statistic, passed to `cpm::processStream()` as `cpmType`. Distribution-free: `"Mann-Whitney"` (location, the default), `"Mood"` (scale), `"Lepage"`, `"Kolmogorov-Smirnov"`, `"Cramer-von-Mises"`. Parametric: `"Student"`, `"Bartlett"`, `"GLR"` (Gaussian), `"Exponential"` (positive data), `"FET"` (Fisher's exact test, for 0/1 Bernoulli data; this one also needs a `lambda` value passed through `...`, e.g. `lambda = 0.3`).
arl0	Target in-control average run length (how many observations, on average, before a false alarm). Defaults to `500`. **cpm** ships thresholds only for a fixed grid (100, 200, 300, 370, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 20000, 30000, 40000 and 50000), and any other value is refused, because the engine answers it by printing an error and reporting no changepoints. The grid is the same for every `cpm_type`, and 50000 is the ceiling: a long series cannot be given an `arl0` proportional to its length indefinitely.

`startup`	Number of observations after each restart before monitoring begins. Defaults to `20`.
`...`	Additional arguments passed to `cpm::processStream()`.

Value

A `ggcpt` object. The `changepoints` tibble carries a `detection_time` column: the index at which the sequential test flagged each change (always later than the estimated location).

References

Ross GJ (2015). "Parametric and nonparametric sequential change detection in R: The cpm package." *Journal of Statistical Software*, **66**(3), 1–20.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
res <- cpm_wrapper(c(rnorm(100), rnorm(100, 3)))
res$changepoints
```

---

cpop_wrapper *CPOP wrapper: optimal change-in-slope detection*

Description

Wraps `cpop::cpop()` (Fearnhead, Maidstone and Letchford, 2019; Fearnhead and Grose, 2024): exact penalised estimation of a *continuous* piecewise-linear mean via dynamic programming with functional pruning. This is the engine behind `cpt_detect(change_in = "slope")`. The fitted broken line is stored in the `fitted` column and renders via `autoplot(show_fit = TRUE)`.

Usage

```
cpop_wrapper(x, penalty = NULL, sd = NULL, ...)
```

Arguments

x	A numeric vector.
penalty	Penalty for adding a changepoint. Defaults to `2 * log(length(x))`. `cpt_detect` resolves its own `"MBIC"` default to a stronger numeric value, so the two entry points need not agree unless `penalty` is given.
sd	Noise standard deviation; when `NULL` it is estimated from the data by the engine's default difference-based estimator.
...	Additional arguments passed to `cpop::cpop()`.

Value

A `ggcpt` object with `change_in = "slope"`. For a continuous fit the reported location is the kink point itself.

References

Fearnhead P, Maidstone R, Letchford A (2019). "Detecting changes in slope with an L0 penalty." *Journal of Computational and Graphical Statistics*, **28**(2), 265–275.

Fearnhead P, Grose D (2024). "cpop: Detecting changes in piecewise-linear signals." *Journal of Statistical Software*, **109**(7), 1–30.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
y <- cumsum(c(rep(0.3, 100), rep(-0.4, 100))) + rnorm(200)
res <- cpop_wrapper(y)
res$changepoints
ggplot2::autoplot(res, show_fit = TRUE)
```

`cpt_annotate_events` *Match detected changepoints to known events*

Description

Joins a table of real-world events to a detection result and reports all three outcomes: changepoints an event explains, changepoints no event explains, and events no changepoint found. Matching reuses the same tolerance rule as `cpt_metrics()` and `cpt_consensus()`, so the package has one notion of "close enough" throughout.

Usage

```
cpt_annotate_events(
  object,
  events,
  location = NULL,
  label = NULL,
  tolerance = 5
)

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

## S3 method for class 'ggcpt_events'
tidy(x, ...)

## S3 method for class 'ggcpt_events'
autoplot(object, repel = NULL, ...)
```

Arguments

`object`	A `ggcpt_events` object (for `autoplot()`).
`events`	A data frame of events with a location column and a label column. The location may be on the position scale or, when the result carries a time index, on the index scale (dates, say); which one is detected automatically from the column's type and reported. An event whose location cannot be placed on the series (a missing value, text that is not a number, or a date string against a `Date` index) is left out of all three outcomes, with a warning that names it.
`location`	Name of the location column. Defaults to the first column whose type matches the result's index (or the first numeric column).
`label`	Name of the label column. Defaults to the first character or factor column.
`tolerance`	Matching window in positions. Defaults to `5`.
`x`	A `ggcpt_events` object.
`...`	Ignored.
`repel`	Use **ggrepel** for the event labels? Defaults to `TRUE` when it is installed.

Value

A ggcpt_events object: a list with

matched one row per matched pair: cp, event, event_value (the event's own location, on the index scale when it was given as one), event_position (that location as a position in the series) and distance.

unexplained detected changepoints with no event (cp).

undetected events with no changepoint (event, event_value, event_position).

matched and unexplained carry cp_index, the changepoint on the original scale, when (and only when) the result carries a time index, so "cp_index" %in% names(x) is the test for it.

tidy() flattens those three slots into **one** table with a status column, and the row count is therefore the number of changepoints *plus* the number of events, not either one alone. The three values are "matched" (a pair: both cp and event filled, with distance), "unexplained_changepoint" (a changepoint with no event: cp filled, event and distance NA) and "undetected_event" (an event with no changepoint: event and position filled, cp and distance NA).

Filter on status, not on is.na(cp). An "unexplained_changepoint" row has a non-missing cp, so subset(tidy(x), !is.na(cp)) returns the matched pairs *and* the unexplained changepoints, which is the natural mistake to make, and it silently overstates how many changepoints an event explains. With print(), tidy() and autoplot().

See Also

geom_cpt_event(), cpt_report().

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 4))
dates <- as.Date("2020-01-01") + 0:119
fit <- cpt_detect(x, method = "pelt", index = dates)
events <- data.frame(when = as.Date(c("2020-03-01", "2020-04-15")),
                     what = c("policy change", "supply shock"))
cpt_annotate_events(fit, events)
```

---

cpt_annotations *Per-annotator ground truth for a benchmark dataset*

Description

Returns the annotation sets of a dataset one row per (annotator, changepoint), so the disagreement between human annotators is visible instead of averaged away.

Usage

```
cpt_annotations(dataset)
```

Arguments

dataset	A dataset in the cpt_datasets() shape (a list with annotations), or a named list of them.

Value

A tibble with dataset, annotator and cp.

See Also

cpt_metrics_annotated().

Other result class: annotate_segments(), as_cpt_series(), as_ggcpt(), is_ggcpt(), new_ggcpt(), print.ggcpt()

Examples

```
cpt_annotations(cpt_datasets(n = 200, names = "step"))
```

---

cpt_batch *Batch changepoint detection over many series*

Description

Runs one detector over every series in a collection: the panel-data loop that methodological and applied work both need constantly. Accepts a matrix/data frame (one column per series) or a named list of numeric vectors. Honours future::plan() for parallel execution when the **future.apply** package is available, with parallel-safe RNG.

Usage

```
cpt_batch(
  x,
  method = "pelt",
  change_in = "mean",
  index = NULL,
  seed = NULL,
  keep_fit = TRUE,
  ...
)

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

## S3 method for class 'ggcpt_batch'
tidy(x, ...)

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

Arguments

x	For cpt_batch(), a numeric matrix or data frame (columns are series) or a list of numeric vectors; for the print() and tidy() methods, a ggcpt_batch object.
method	Detection method, passed to cpt_detect().
change_in	What to detect change in, passed to cpt_detect().
index	Optional time index shared by every series in the panel (a vector of dates, say), or a named list of one index per series. A list has to cover every series, with indices of one type, because tidy() and autoplot() stack them into one table and one axis. Carried onto each result and used by tidy() and autoplot(), so a faceted plot of fifty series shows dates rather than positions.
seed	Optional seed for reproducible parallel execution (passed to future.apply::future_lapply() as future.seed; applied via set.seed() when running sequentially). The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
keep_fit	Keep each engine's raw fit in result[[i]]$fit? Defaults to TRUE, which is what makes a batch result as inspectable as a single one. Set it to FALSE for a large panel: a few engines return fits far bigger than the data they were given. strucchange keeps a triangular $O(n^2)$ RSS matrix, so a single 2000-point series costs about 135 MB, and bfast and bocpd are in the tens of MB, and a panel multiplies that by the number of series. Everything else on the object, including tidy() and autoplot(), is unaffected; only accessors that read $fit (cpt_statistic(), ggcpt_posterior(), cpt_confint(engine = \"native\")) need it.
...	Additional arguments passed to every cpt_detect() call.
object	A ggcpt_batch object (for autoplot()).

Value

A ggcpt_batch object: a tibble with one row per series and columns series, n_changepoints, changepoints (a list-column of tidy tibbles), and result (a list-column of ggcpt objects). Methods: print(), tidy() (one row per changepoint across all series, with columns series, cp and cp_value), and autoplot() (faceted small-multiples with each series' changepoints).

Examples

```
set.seed(2026)
X <- cbind(a = c(rnorm(60), rnorm(60, 4)), b = rnorm(120))
batch <- cpt_batch(X, method = "pelt")
batch
tidy(batch)
ggplot2::autoplot(batch)
```

cpt_benchmark *Benchmark detectors across datasets*

Description

Runs a grid of methods over a collection of labelled series and scores every cell with `cpt_metrics()` (or `cpt_metrics_annotated()` when a dataset carries several annotators). An engine that errors on one dataset records the message and the grid continues.

Usage

```
cpt_benchmark(
  datasets,
  methods = c("pelt", "binseg", "wbs"),
  metrics = c("covering", "f1"),
  tolerance = 5,
  change_in = "mean",
  parallel = TRUE,
  progress = TRUE,
  seed = NULL,
  ...
)

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

## S3 method for class 'ggcpt_benchmark'
tidy(x, ...)

## S3 method for class 'ggcpt_benchmark'
autoplot(
  object,
  plot_type = c("heatmap", "ranks", "critical_difference"),
  metric = NULL,
  alpha = 0.05,
  ...
)
```

Arguments

`datasets`	A named list of datasets. Each element is either a plain numeric vector (no ground truth, so only descriptive columns are filled) or a list with `series` and one of `truth`, `changepoints` (an integer vector) or `annotations` (a list of integer vectors, one per annotator). A list carrying none of those is scored `NA` and warns. A `series` with several columns stays a matrix, so only the multivariate methods can score it. `cpt_datasets()` builds a ready-made collection.
`methods`	Character vector of method names.

metrics	Which metrics to keep. Defaults to c("covering", "f1"), the pair van den Burg and Williams (2020) established as the benchmark standard.
tolerance	Matching window passed to the metrics. Defaults to 5.
change_in	Passed to every detector.
parallel	Use future::plan() when **future.apply** is available? Defaults to TRUE.
progress	Show a **progressr** progress bar when that package is installed and a handler is enabled? Defaults to TRUE.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to every cpt_detect() call.
x	A ggcpt_benchmark object.
object	A ggcpt_benchmark object (for autoplot()).
plot_type	"heatmap" (method by dataset, coloured by the metric), "ranks" (mean rank per method) or "critical_difference" (the Demšar diagram: mean ranks with the Nemenyi critical distance, the standard way this literature says "method A beats method B").
metric	Which metric to plot. Defaults to the first one scored.
alpha	Level for the critical distance, in (0, 1). Defaults to 0.05. Only used by plot_type = "critical_difference", which also needs at least two methods to compare.

Value

A ggcpt_benchmark object: a tibble with one row per (dataset, method): dataset, method, n, n_annotators (how many ground-truth sets the dataset supplied), n_cp, the requested metrics, runtime and error, with print(), tidy() and autoplot() ("heatmap", "ranks", "critical_difference").

Reading the critical-difference diagram

Rank 1 is best. Each method's mean rank is taken over the datasets, in the direction the metric calls for: higher is better for covering, f1, precision, recall and rand_index; lower for hausdorff, annotation_error, mae_matched and rmse_matched. A metric that came back NA takes the *worst* rank on that dataset rather than being dropped, so a method that failed there is penalised for it instead of quietly scoring on a smaller sample; ties share the average rank.

The bar is the Nemenyi critical distance

$$CD = q_\alpha \sqrt{k(k+1)/(6N)}$$

for k methods over N datasets, where q_α is the Studentised range statistic at α divided by $\sqrt{2}$ (Demsar, 2006; the constants agree with that paper's Table 5 to three decimals for $k = 2, \ldots, 10$). Two methods whose mean ranks differ by less than CD are not distinguished at that level.

One caveat the diagram cannot show. Nemenyi is a *post-hoc* procedure, and the convention is to run it only after a Friedman test has rejected the null that all methods rank equally. autoplot() does not run that omnibus test (it draws the diagram it is asked for), so a gap wider than CD on a grid where Friedman would not have rejected is not the significant difference it looks like. With the handful of datasets cpt_datasets() supplies, N is small and CD correspondingly wide; read the diagram as a descriptive summary unless N is large enough to support the test.

References

van den Burg GJJ, Williams CKI (2020). "An evaluation of change point detection algorithms." *arXiv preprint arXiv:2003.06222*. doi:10.48550/arXiv.2003.06222.

Demšar J (2006). "Statistical comparisons of classifiers over multiple data sets." *Journal of Machine Learning Research*, **7**, 1–30.

See Also

`cpt_datasets()`, `cpt_load_tcpd()`, `cpt_metrics()`.

Examples

```
bm <- cpt_benchmark(cpt_datasets(n = 200, seed = 1),
                    methods = c("pelt", "binseg", "amoc"),
                    progress = FALSE)
bm
ggplot2::autoplot(bm)
```

`cpt_cite` *Cite the method behind a result*

Description

Returns the bibliographic reference(s) for the method behind a `ggcpt` result (or a method name), so an analysis can cite the right methodological paper without leaving R.

Usage

```
cpt_cite(x)
```

Arguments

`x`	A `ggcpt` object, a method name (e.g. `"pelt"`), or missing, in which case references for every known method are returned. A method registered with `cpt_register_method()` returns the citation that registration supplied, or a plain statement that none was given.

Value

A tibble with columns `method` and `reference`, invisibly; the references are also printed.

Examples

```
cpt_cite("pelt")
res <- cpt_detect(c(rnorm(50), rnorm(50, 5)), method = "pelt")
cpt_cite(res)
```

cpt_confint *Confidence intervals for changepoint locations*

Description

Answers "where could this changepoint be?" for any result, and says which of four routes it used. `show_ci = TRUE` in `autoplot()` works only for the handful of engines that ship intervals of their own; this generic covers the rest, and (because the four routes mean genuinely different things) reports the provenance in a `source` column rather than presenting them as interchangeable.

Usage

```
cpt_confint(
  object,
  level = 0.95,
  method = c("auto", "native", "nsp", "bootstrap", "posterior"),
  B = 200,
  seed = NULL,
  ...
)
```

Arguments

`object`	A `ggcpt` object.
`level`	Confidence/credible level. Defaults to `0.95`. Honoured by the routes that compute an interval (`"bootstrap"`, `"posterior"` and `"nsp"`) and **ignored by** `"native"`, which reports the interval the engine already computed at whatever level it was asked for. Note that `method = "auto"` resolves to `"native"` whenever the engine supplied one, so an explicit `level` can go unused there too: it is reported in the `level` column either way, and supplying a level the answer does not carry now warns rather than passing silently. To choose the level yourself, name a computing route.
`method`	Which route to use: `"auto"` (default) native if the engine supplied intervals, else posterior if it supplied one, else bootstrap. `"native"` the engine's own `ci_lower`/`ci_upper` (SMUCE/HSMUCE simultaneous confidence sets, **strucchange** break-date intervals, **segmented** breakpoint intervals, **mcp** posterior intervals, **bfast** break confidence intervals, and `taylor`'s bootstrap confidence limits). The engines that supply them are the ones `cpt_methods()` marks in its `ci` column; `nsp` is marked there too but is reported under its own provenance below, because its regions are not intervals around an estimate. `"nsp"` Narrowest Significance Pursuit regions computed on the same series and matched to the changepoints. These are *not* intervals around an estimate (see `nsp_wrapper()`), but they are the strongest guarantee available, so the mapping is reported as `source = "nsp_region"` and a changepoint in no region gets `NA`.

"bootstrap" within-segment residual resampling (the cpt_stability() scheme), re-running the detector and taking quantiles of the re-detected location. Model-agnostic and available for every engine; it measures the sampling variability of the *procedure*, conditional on the fitted segmentation, and it is not exact.

"posterior" a credible interval from the engine's posterior changepoint-probability profile (bcp, beast): the narrowest *contiguous* set of positions around the estimate holding level of the posterior changepoint mass in that changepoint's window, where the window is bounded by the neighbouring changepoints so two of them cannot claim the same mass twice.
Expect these to be wide, and read the width as a statement about the profile rather than about the location. Both supplying engines put roughly two-thirds of the window's mass at the estimate itself and spread the remaining third as a thin floor across every other position, so reaching a high level means swallowing that floor. Measured on a 200-point series with one clean change: level = 0.5 gives a width of 0 (the mode alone holds more than half), 0.8 gives 72 (**bcp**) and 91 (**beast**), and 0.95 gives 166 and 187, which is 83% and 94% of the series. The requested level *is* delivered in each case; what a wide interval says is that the posterior did not localise the change, not that the location is uncertain by that much. cpt_confint() warns when an interval covers more than half its window, for exactly that reason.

B — Bootstrap replicates for method = "bootstrap". Defaults to 200.

seed — Optional seed (bootstrap and NSP are both random). The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

... — Passed to cpt_detect() on the bootstrap replicates, or to nsp_wrapper().

Value

A tibble with one row per changepoint: cp, ci_lower, ci_upper, level, source, plus cp_index/ci_lower_index/ci_upper on the original scale when the result carries a time index, and n_replicates for method = "bootstrap" (how many replicates actually contributed a draw for that changepoint). method = "nsp" returns NA bounds for a changepoint that falls in no region, which is a finding rather than a failure.

How well these cover

Measured over 120 replicates on a 200-point series with one changepoint at 100 and a jump of three standard deviations, at a nominal level of 0.95: "bootstrap" on pelt covered 0.992 of the time with a mean width of 2.2; **strucchange**'s native intervals covered 1.000 at width 4.4; **stepR**'s (smuce) covered 0.992 at width 4.6; and "posterior" on **bcp** covered 1.000 at width 157. Every route is *conservative* (none under-covers), and the width is what separates them. The two native routes and the bootstrap are the ones to quote; see the note on "posterior" above for why its interval is so much wider.

See Also

cpt_test(), cpt_stability(), nsp_wrapper().

Examples

```
set.seed(2026)
x <- c(rnorm(80), rnorm(80, 4))
fit <- cpt_detect(x, method = "pelt")
cpt_confint(fit, method = "bootstrap", B = 25, seed = 1)
```

`cpt_consensus` *Consensus changepoints across several detectors*

Description

Runs a set of detectors on one series and reports the locations they agree on. Two detections count as the same changepoint when they fall within `tolerance` of each other, the same tolerance window `cpt_metrics()` matches on (van den Burg and Williams, 2020), so the package has one notion of "close enough" and not two. The *grouping* necessarily differs: `cpt_metrics()` matches two sets one-to-one, while consensus has to cluster K of them, so detections are swept in order and a new cluster opens as soon as one lies more than `tolerance` from the cluster's first member. That cap stops a chain of near-neighbours merging into one arbitrarily wide cluster.

Usage

```
cpt_consensus(
  x,
  methods = c("pelt", "binseg", "amoc"),
  tolerance = 5,
  min_votes = 2,
  change_in = "mean",
  index = NULL,
  seed = NULL,
  ...
)

## S3 method for class 'ggcpt_consensus'
autoplot(object, plot_type = c("series", "agreement"), ...)

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

Arguments

`x`	A `ggcpt_consensus` object (for `print()`).
`methods`	Character vector of method names.
`tolerance`	Matching window, in positions. Defaults to 5.
`min_votes`	Minimum number of methods that must find a location for it to enter the consensus. Defaults to 2.

A value **strictly between 0 and 1** is read as a proportion of the methods that ran; anything else is a count, rounded *up* (`2.5` needs three methods, never two). The boundary is worth knowing, because it falls exactly where a reader thinking in proportions would write "unanimous": with three methods, `min_votes = 0.99` needs all three, while `min_votes = 1` (and `1.0`, which is the same number) is a count of one and so the *least* strict setting there is. For unanimity, pass the number of methods, or a fraction just below 1.

A count larger than the number of methods that ran cannot be reached, so the consensus would be empty by construction; that warns rather than returning a result indistinguishable from "the methods agreed on nothing".

`change_in` Passed to each detector.

`index` Optional time index, carried onto the result.

`seed` Optional seed for reproducibility. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

`...` Passed on to `autoplot.ggcpt()` for `plot_type = "series"`.

`object` A `ggcpt_consensus` object (for `autoplot()`).

`plot_type` `"series"` (the consensus segmentation, votes shown by line width) or `"agreement"` (a method-by-location dot matrix showing exactly who voted for what).

Value

A `ggcpt` object (so it plots and tidies like any other result) whose changepoints tibble carries `votes` and `methods` (a comma-separated list of the methods that found each location), with the per-method detections kept in a `consensus` attribute and drawn by `autoplot(plot_type = "agreement")`.

Consensus is not inference

Agreement among detectors is **not** a p-value, and a location found by six of seven methods is not thereby significant at any level: the methods are run on the same data and are strongly correlated, several of them share an engine, and none of the votes is independent. Read the vote count as a robustness display ("this feature does not depend on which detector I picked"), and use `nsp_wrapper()` or `cpt_confint()` when you need a guarantee.

References

van den Burg GJJ, Williams CKI (2020). "An evaluation of change point detection algorithms." *arXiv preprint arXiv:2003.06222*. doi:10.48550/arXiv.2003.06222.

See Also

`ggcpt_compare()`, `cpt_recommend()`, `cpt_metrics()`.

Examples

```
set.seed(2026)
x <- c(rnorm(80), rnorm(80, 4))
cons <- cpt_consensus(x, methods = c("pelt", "binseg", "amoc"))
```

```
tidy(cons)
ggplot2::autoplot(cons)
```

`cpt_crops` *CROPS: the full penalty path of a penalised changepoint method*

Description

Runs PELT once per *distinct* optimal segmentation as the penalty ranges over `[pen_min, pen_max]`, using the CROPS algorithm of Haynes, Eckley and Fearnhead (2017) as implemented by the **changepoint** package. Instead of committing to one penalty, the analyst sees every segmentation the data admits along the path, together with its cost, and picks the elbow.

Usage

```
cpt_crops(
  x,
  change_in = c("mean", "var", "meanvar"),
  pen_min = NULL,
  pen_max = NULL,
  ...
)

## S3 method for class 'ggcpt_path'
autoplot(
  object,
  type = c("elbow", "path", "segmentations"),
  max_facets = 12,
  ...
)

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

## S3 method for class 'ggcpt_path'
tidy(x, ...)
```

Arguments

`x`	For `cpt_crops()`, a numeric vector; for the `print()` and `tidy()` methods, a `ggcpt_path` object.
`change_in`	What to detect change in: `"mean"`, `"var"`, or `"meanvar"`. Defaults to `"mean"`.
`pen_min, pen_max`	The penalty interval to sweep. Default to `log(n)` and `10 * log(n)`.
`...`	Additional arguments passed to the underlying `changepoint::cpt.mean()`, `cpt.var()`, or `cpt.meanvar()` call.

object	A ggcpt_path object (for autoplot()).
type	Plot type for autoplot(): "elbow" (cost against number of changepoints, the classic CROPS diagnostic), "path" (number of changepoints against penalty), or "segmentations" (the data faceted by solution, with that solution's changepoints drawn).
max_facets	Maximum number of solutions shown by type = "segmentations". Defaults to 12.

Value

A ggcpt_path object: a list with a solutions tibble (one row per distinct segmentation: penalty, n_cpts, cost, and a cpts list-column), the data, and metadata. penalty is the **lower end** of the penalty interval on which that segmentation is optimal: CROPS returns the breakpoints of the penalty axis, so K segmentations come with $K + 1$ boundaries and each row is optimal from its own penalty up to the next row's. Methods: print(), tidy(), and autoplot() (elbow plot by default; type = "path" for penalty vs. number of changepoints; type = "segmentations" for the faceted segmentations).

References

Haynes K, Eckley IA, Fearnhead P (2017). "Computationally efficient changepoint detection for a range of penalties." *Journal of Computational and Graphical Statistics*, **26**(1), 134–143.

Killick R, Eckley I (2014). "changepoint: An R package for changepoint analysis." *Journal of statistical software*, **58**(3), 1–19.

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 3), rnorm(100, -1))
path <- cpt_crops(x)
path
ggplot2::autoplot(path)
ggplot2::autoplot(path, type = "segmentations")
```

---

cpt_datasets *A catalogue of benchmark datasets*

Description

Builds a ready-made collection of labelled series for cpt_benchmark(). By default these are the canonical simulated signals this package already ships, which means the benchmark runs offline, deterministically, and inside R CMD check. Pass source = "tcpd" for the Turing Change Point Dataset instead, which is downloaded and cached.

Usage

```
cpt_datasets(
  source = c("simulated", "tcpd"),
  n = 500,
  seed = 1,
  names = NULL,
  ...
)
```

Arguments

source	"simulated" (default) or "tcpd".
n	Length of each simulated series. Defaults to 500.
seed	Seed for the simulated signals. Defaults to 1. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
names	Optional subset of dataset names.
...	Passed to cpt_load_tcpd() for source = "tcpd".

Value

A named list of list(series, annotations) datasets, ready for cpt_benchmark().

See Also

cpt_load_tcpd(), cpt_benchmark().

Examples

```
names(cpt_datasets(n = 200))
```

---

cpt_delay *Detection delay and false-alarm rate*

Description

Scores an online result the way the sequential literature does: how long after each true change did the first alarm arrive, and how many alarms were raised with no change behind them. cpt_metrics() is the wrong tool for an online detector: it asks whether the *location* was recovered, which a sequential procedure never claims. It will not stop you: cpt_metrics() takes bare integer vectors and never sees which detector produced them, so it cannot know. (An earlier version of this sentence said it warns. It does not, and given that signature it could not.)

Usage

```
cpt_delay(object, truth, max_delay = Inf)

## S3 method for class 'ggcpt_delay'
tidy(x, ...)

## S3 method for class 'ggcpt_delay'
glance(x, ...)

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

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

Arguments

object	A ggcpt_delay object (for autoplot()).
truth	Integer vector of true changepoint positions, on the same clock as the alarms. When object is a monitor built by cpt_replay() the baseline offset is applied automatically.
max_delay	Alarms further than this after a change are treated as false alarms rather than late detections. Defaults to Inf.
x	A ggcpt_delay object.
...	Ignored.

Value

A ggcpt_delay object: a list with per_change (one row per true change: truth, alarm, delay, detected), false_alarms, and the summary statistics mean_delay, median_delay, n_false_alarms and arl: monitored observations per false alarm. The baseline a cpt_replay() monitor trained on is not counted, because no alarm can fire there; n_obs is the length of the stream in series positions, baseline included.

See Also

cpt_monitor(), cpt_replay().

Examples

```
set.seed(2026)
mon <- cpt_replay(c(rnorm(200), rnorm(200, 3)), method = "edetector")
cpt_delay(mon, truth = 200)
```

cpt_detect *Unified changepoint detection dispatcher*

Description

Runs a changepoint detection method on a sequence and returns a tidy ggcpt result object. This is the recommended entry point for most users. See cpt_methods() for the full method table with engines and capabilities.

Usage

```
cpt_detect(
  x,
  method = "pelt",
  change_in = "mean",
  penalty = "MBIC",
  index = NULL,
  y = NULL,
  ...
)
```

Arguments

x	The series. A numeric vector for univariate methods, or a numeric matrix/data frame (rows are time points) for the multivariate methods (run subset(cpt_methods(), multivariate)$method for the list). A ts, xts, zoo or (unkeyed) tsibble is accepted directly and its time index is carried through to tidy() and autoplot(); so is a data frame together with y (and optionally index).
method	Detection method: any method in cpt_methods() whose status is "available" or "registered". Methods whose engines live in Suggests report what to install when missing; cpt_register_method() adds detectors this package does not wrap.
change_in	What to detect change in. One of "mean", "var", "meanvar", "slope", "distribution", "covariance", "network", "regression" or "seasonality". Defaults to "mean". The requested value is validated against the method's capabilities (see cpt_methods()); incompatible combinations error rather than silently running something else. A *compatible* request may still be routed to the method's own native change type, because several engines have no separate estimator for the thing being asked about. That is never silent: the result's change_in records what was actually detected, so compare it with what you asked for. Measured across every method and every value its supports entry lists, six pairs are routed: not's "var" becomes "meanvar" (its variance contrast is piecewise-constant in mean *and* variance), cpm's "mean" and "var" both become "distribution", kcp's become "running mean" and "running var", and wbsts's "mean" becomes "var" (it detects change in the wavelet spectrum). Every other listed combination returns the change type it was asked for.

penalty — Penalty type or value. Either a character string (`"MBIC"`, `"BIC"`, `"SIC"`, `"AIC"`, `"Hannan-Quinn"`, `"None"`) or a numeric penalty value. Defaults to `"MBIC"`. See the penalty-semantics section of `cpt_penalty` for how each engine interprets it; methods that use thresholds, significance levels, or posteriors instead of penalties ignore this argument, and `"segneigh"` falls back to `"SIC"` because **changepoint** does not implement MBIC for Segment Neighbourhood. Note also that the default `"MBIC"` is resolved to a *numeric* value for the numeric-penalty engines (`"fpop"`, `"cpop"`, `"decafs"`), and that value is stronger than those wrappers' own `2 * log(n)` default (19.9 against 11.8 at $n = 360$), so `cpt_detect(x, method = "decafs")` can report fewer changepoints than `decafs_wrapper(x)` on the same series. Pass `penalty` explicitly to make the two entry points agree.

index — Optional time index, one value per observation (dates, say). Detection still runs on observation positions (every wrapped engine assumes an equally spaced sequence), but the index is stored on the result and threaded through `tidy()` (as `cp_index`), `augment()`, `autoplot()` and `cpt_report()`, so the output speaks in the user's own units. An index that is not equally spaced warns. When `x` is a data frame and `y` is given, `index` selects a column of that data frame instead of being a vector.

y — Column selection for the data-frame interface: `cpt_detect(df, y = value, index = date, method = "pelt")`. A bare column name, a string, or a column position. Only meaningful when `x` is a data frame; a data frame passed without `y` keeps its 0.4.0 meaning (one column per coordinate).

... — Additional arguments passed to the specific wrapper (see the wrapper's help page for engine-specific options). Where an argument is also derived from `change_in` (not's `contrast`, cpm's `cpm_type`, kcp's `running_stat`, sn's `parameter`, fastcpd's `family`), a value supplied here takes precedence. Check the spelling against the wrapper's help page: several engines end their own signature in `...` (**wbs**, **not**, **Rbeast**, **strucchange**, **segmented**, **fastcpd**, **fChange**, **bfast**), so for those a misspelt argument name is silently discarded upstream and the engine quietly uses its default rather than reporting the typo. Every other wired method rejects an unknown argument by name.

Value

A `ggcpt` object: a list with `changepoints` (`cp`, `cp_value`), `segments` (`seg_id`, `start`, `end`, `n`, `param_estimate`), `data` (`index`, `value`), the `method`, `change_in`, `penalty`, `cp_convention` and `runtime` that produced it, the matched `call`, and `fit`, the raw upstream object. Optional slots (`data_wide`, `regions`, `diagnostics`, ...) appear only when an engine supplies them; `new_ggcpt()` documents all of them, and `tidy.ggcpt()`, `glance.ggcpt()` and `augment.ggcpt()` are the supported way to read one.

Scale sensitivity of the penalised change-in-mean engines

`"pelt"`, `"binseg"`, `"segneigh"` and `"fpop"` compare a penalty against a *raw* segment cost when `change_in = "mean"`: **changepoint**'s Normal cost assumes a noise standard deviation of 1, and **fpop**'s `lambda` is an absolute penalty on the residual sum of squares. Neither rescales the data, so on a series whose noise is much wider than 1 the penalty is effectively negligible and the segmentation shatters. On 200 observations with one true changepoint in the middle and a jump of five standard

deviations, `"pelt"` returns 1 changepoint at $\sigma = 1$, 39 at $\sigma = 3$ and 141 at $\sigma = 10$. These are means over 20 draws, because a single draw is not stable here: the same three settings gave 21/75 at $n = 100$ and 57/266 at $n = 400$, so the effect grows with the series as well as with the noise. Three ways to avoid it, in order of convenience:

- standardise the series first (`cpt_detect(scale(x)[, 1], method = "pelt")`);
- pass a penalty on the data's own scale, for example `penalty = 2 * log(length(x)) * stats::var(diff(x)) / 2`;
- use `change_in = "meanvar"`, which estimates a variance per segment and is unaffected.

Most other engines are unaffected: SMUCE, WBS, WBS2, NOT, MOSUM, Isolate-Detect, TGUH, CPOP, `"bcp"`, `"beast"` and the nonparametric and multivariate methods estimate or cancel the noise scale internally, and returned the same segmentation at a thousandth, one and a thousand times the units. Three did not, on the same series: `"geomcp"` runs PELT on its mapped distance and angle series and so inherits the sensitivity above; `"decafs"` floors its noise estimate at about 0.03, so it under-segments a series whose noise is smaller than that; and `"bocpd"`'s default prior is on the data's own scale. At a thousandth of the units the last two found nothing. Standardising first avoids all three.

See Also

`cpt_methods()` for what is available and what each method can do. To get the result out: `tidy.ggcpt()`, `glance.ggcpt()`, `augment.ggcpt()`, `summary.ggcpt()` and `print.ggcpt()`. To draw it: `autoplot.ggcpt()`. For the penalty: `cpt_penalty()`.

Examples

```
set.seed(2022)
x <- c(rnorm(100, 0, 1), rnorm(100, 10, 1))
result <- cpt_detect(x, method = "pelt", change_in = "mean")
result
ggplot2::autoplot(result)

# A date index: detection is unchanged, but the report speaks in dates.
dates <- as.Date("2000-01-01") + 0:199
dated <- cpt_detect(x, method = "pelt", index = dates)
tidy(dated)

# The data-frame interface.
df <- data.frame(day = dates, value = x)
cpt_detect(df, y = value, index = day, method = "pelt")
```

---

cpt_gt *A publication-ready changepoint table*

Description

Renders the changepoints of a result as a **gt** table, with the time index, location intervals and segment levels formatted for a paper. Falls back to a plain tibble, with a note, when **gt** is not installed.

Usage

```
cpt_gt(object, title = NULL, subtitle = NULL, digits = 3)
```

Arguments

`object`	A ggcpt object.
`title, subtitle`	Table title and subtitle. Sensible defaults are derived from the result.
`digits`	Digits for the numeric columns. Defaults to 3.

Value

A **gt** table, or a tibble when **gt** is unavailable.

See Also

`cpt_report()`.

Examples

```
set.seed(2026)
cpt_gt(cpt_detect(c(rnorm(60), rnorm(60, 4)), method = "pelt"))
```

---

cpt_influence *Influence diagnostics for a changepoint segmentation*

Description

Perturbs one observation at a time (deleting it, or replacing it with an outlier), re-runs the detector, and reports what changed: the number of changepoints, where they moved to, and how the segment parameters responded. This is the diagnostic family of Wilms, Killick and Matteson (2022), rendered in **ggplot2** so it composes with the rest of the package.

Usage

```
cpt_influence(
  object,
  type = c("delete", "outlier"),
  engine = c("auto", "changepoint.influence", "recompute"),
  subset = NULL,
  outlier_sd = 5,
  seed = NULL,
  ...
)

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

```
## S3 method for class 'ggcpt_influence'
autoplot(
  object,
  plot_type = c("overview", "location", "parameter", "map"),
  ...
)

## S3 method for class 'ggcpt_influence'
tidy(x, ...)
```

Arguments

object	A ggcpt_influence object (for autoplot()).
type	"delete" (drop the observation) or "outlier" (replace it with a large value). Defaults to "delete".
engine	Which implementation to use: "auto" (default) uses **changepoint.influence** for a type = "delete" diagnostic of a change-in-mean fit from a **changepoint** engine when that package is installed (the only models it supports), and the generic recomputation otherwise; "changepoint.influence" insists on the former; "recompute" insists on the latter, which works for every wired and registered method.
subset	Optional integer vector of observation positions to perturb. Influence by deletion costs one detector fit per observation, so on a long series or an expensive engine this is the argument that makes the diagnostic affordable. Defaults to every observation.
outlier_sd	For type = "outlier", how many residual standard deviations the substituted value sits above the fitted level. Defaults to 5. Used by the recomputation engine, which is what "auto" picks for an outlier diagnostic; engine = "changepoint.influence" places its outliers by its own rule and does not read it.
seed	Optional seed, for detectors that randomise. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to cpt_detect() on each perturbed series.
x	A ggcpt_influence object (for print()).
plot_type	Which diagnostic to draw: "overview" (the series with the influential observations highlighted), "location" (perturbed changepoint locations against the perturbed observation), "parameter" (segment-parameter shift per perturbation) or "map" (the full influence map: perturbed observation on x, position on y, parameter shift as fill).

Value

A ggcpt_influence object: a list with

influence a tibble with one row per perturbed observation: index, n_cp, delta_n_cp (against the unperturbed fit), max_shift (largest movement of a surviving changepoint, in positions),

`param_shift` (largest absolute change in a segment parameter) and `cpts` (a list-column of the perturbed changepoint sets). A perturbation whose re-fit failed has `n_cp = NA`, and is warned about;

`param` an $n \times n$ matrix of per-observation segment parameters, one row per perturbation: the input to the influence map;

`original`, `type`, `engine`, `method`

with `print()` and `autoplot()` methods.

References

Wilms I, Killick R, Matteson DS (2022). "Graphical influence diagnostics for changepoint models." *Journal of Computational and Graphical Statistics*, **31**(3), 753–765. doi:10.1080/10618600.2021.2000873.

See Also

`cpt_leverage()`, `cpt_sensitivity()`, `cpt_stability()`.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(40), rnorm(40, 4)), method = "pelt")
inf <- cpt_influence(fit)
inf
head(cpt_leverage(inf))
```

---

cpt_install_engines *Install the engines behind a family of methods*

Description

The package wraps a lot of upstream engines, all of them in `Suggests` so that installing **ggchangepoint** does not drag in dozens of packages nobody asked for. The cost of that choice is a discovery problem: `cpt_detect(x, method = "smuce")` tells you to install **stepR**, but only one package at a time. This installs a whole family in one call.

Usage

```
cpt_install_engines(bundle = "core", dry_run = FALSE, ...)
```

Arguments

`bundle`	Which family to install:
	`"core"` the engines the common methods need: **fpop**, **wbs**, **breakfast**, **not**, **mosum**, **IDetect**, **stepR**.
	`"bayesian"` **bcp**, **ocp**, **Rbeast**.
	`"nonparametric"` **cpm**, **kcpRS**, **CptNonPar**, **SNSeg**.

	`"highdim"` **InspectChangepoint**, **ocd**, **changepoint.geo**, **HDCD**, **changepoints**. `"functional"` **fChange**, **KWCChangepoint**, **fabisearch**. `"regression"` **strucchange**, **segmented**, **EnvCpt**, **DeCAFS**, **cpop**, **fastcpd**. `"inference"` **nsp**, **crossvalidationCP**, **changepoint.influence**, **penaltyLearning**. `"applied"` **trend**, **ChangePointTaylor**, **bfast**, **wbsts**, **binsegRcpp**. `"time"` index and coercion support: **zoo**, **xts**, **tsibble**. `"reporting"` **gt**, **ggrepel**, **plotly**, **ggiraph**, **progressr**, **jsonlite**. `"all"` every engine and extra the package knows about. Several bundles may be given at once.
`dry_run`	Report what would be installed without installing anything. Defaults to `FALSE`.
`...`	Passed to `install.packages()`.

Details

Every package comes from the repositories in `getOption("repos")` except **fpop**, which was archived from CRAN in 2026 at its maintainer's request and is installed from R-Forge (`https://R-Forge.R-project.org`), where it is developed.

Value

Invisibly, a tibble with one row per package (`package`, `bundle`, `installed_before`, `installed_after`).

See Also

`cpt_methods()` for the per-method installation status.

Examples

```
cpt_install_engines("bayesian", dry_run = TRUE)
```

---

cpt_labels *Changepoint labels*

Description

Builds the tidy label set that the supervised functions consume: one row per labelled region, each asserting how many changepoints that stretch of the series contains.

Usage

```
cpt_labels(start, end, change = "change", series = NA_character_)

## S3 method for class 'cpt_labels'
tidy(x, ...)
```

Arguments

start, end	Integer vectors of region boundaries (positions, inclusive).
change	What the region asserts, recycled to length: "change" at least one changepoint lies in the region. "one_change" exactly one does: a stricter label, and the one that makes false positives detectable inside a positive region. "no_change" none does.
series	Optional series identifier, for label sets spanning a panel.
x	A cpt_labels tibble (for tidy(), which returns it as a plain tibble).
...	Ignored.

Value

A cpt_labels tibble with columns label_id, series, start, end, change.

References

Hocking TD, Rigaill G, Vert J, Bach F (2013). "Learning sparse penalties for change-point detection using max margin interval regression." In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 172–180.

See Also

cpt_label_error(), geom_cpt_label(), cpt_learn_penalty(), as_cpt_labels() to convert a plain changepoint set.

Examples

```
cpt_labels(c(40, 70), c(60, 90), c("change", "no_change"))
```

---

cpt_label_error *Score a segmentation against labels*

Description

Counts label errors: a positive region with no changepoint is a false negative, a negative region with one is a false positive, and a "one_change" region with two or more is a false positive as well. This is the accuracy measure supervised changepoint detection is built on, and, unlike an information criterion, it is defined by what the expert asserted rather than by a model assumption.

Usage

```
cpt_label_error(object, labels)

## S3 method for class 'cpt_label_error'
tidy(x, ...)

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

Arguments

object	A ggcpt object, or an integer vector of changepoint positions.
labels	A cpt_labels tibble (or anything with start/end/change columns).
x	A cpt_label_error object (for print()).
...	Ignored.

Value

A tibble with one row per label: label_id, series (the label set's series identifier, NA for a single unnamed series), start, end, change, n_changes (how many detections fell inside), status ("correct", "false_positive" or "false_negative"), carrying the totals in an errors attribute and printing them. An empty label set gives a zero-row tibble with the same columns.

See Also

cpt_labels(), cpt_label_error_curve(), geom_cpt_label().

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 4)), method = "pelt")
labs <- cpt_labels(c(40, 70), c(60, 95), c("one_change", "no_change"))
cpt_label_error(fit, labs)
```

---

cpt_label_error_curve *Label error as a function of the penalty*

Description

Runs one detector across a penalty grid and counts label errors at each setting: the curve penalty learning is fitted to, and the honest way to see whether *any* penalty can satisfy the labels.

Usage

```
cpt_label_error_curve(
  x,
  labels,
  method = "pelt",
  penalties = NULL,
  change_in = "mean",
  ...
)

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

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

Arguments

x	A ggcpt_label_curve object (for print()).
labels	A cpt_labels tibble.
method	Detection method. Defaults to "pelt".
penalties	Numeric vector of penalties to try. When NULL (the default) the grid is chosen *adaptively*: it starts below log(n), where the segmentation shatters, and the top end is found by doubling until the detector reports no changepoints at all. A fixed grid cannot do this: on a series with a large change, a grid that stops at a few hundred never produces a false negative, the error curve never turns back up, and the target interval comes out unbounded above, which is useless to cpt_learn_penalty(). The probe costs at most a dozen extra detector fits; pass penalties explicitly for an expensive engine.
change_in	Passed to the detector.
...	Additional arguments passed to cpt_detect().
object	A ggcpt_label_curve object (for autoplot()).

Value

A ggcpt_label_curve object: a tibble with penalty, n_cp, errors, false_positive, false_negative, plus print() and autoplot(). The target attribute holds the interval of log(penalty) achieving the minimum error, which is what cpt_learn_penalty() regresses on.

See Also

cpt_learn_penalty(), cpt_select().

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 4))
```

```
labs <- as_cpt_labels(60, n = 120)
curve <- cpt_label_error_curve(x, labs, penalties = c(2, 8, 32, 128))
curve
```

`cpt_learn_penalty` *Learn a penalty from labelled series*

Description

Fits the supervised penalty model of Hocking et al. (2013): each series contributes a target *interval* of log-penalties (those achieving the fewest label errors), a feature vector is computed from the series, and a linear model is fitted by minimising the squared hinge loss on those intervals, which is max-margin interval regression. The result has a `predict()` method, and `cpt_detect()` accepts it directly as `penalty`, so a learned penalty is used exactly like a number.

Usage

```
cpt_learn_penalty(
  series,
  labels,
  method = "pelt",
  penalties = NULL,
  engine = c("auto", "penaltyLearning", "native"),
  ...
)

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

## S3 method for class 'ggcpt_penalty_model'
coef(object, ...)

## S3 method for class 'ggcpt_penalty_model'
predict(object, newdata, ...)
```

Arguments

`series`	A named list of numeric vectors (or a matrix/data frame with one column per series).
`labels`	Either a single `cpt_labels` tibble whose `series` column names the series, or a list of label tibbles parallel to `series`.
`method`	Detection method used to build the label-error curves. Defaults to `"pelt"`.
`penalties`	Penalty grid for the curves; passed to `cpt_label_error_curve()`.
`engine`	`"auto"` (default) uses **penaltyLearning**'s `IntervalRegressionCV()` when the package is installed and there are enough series for its cross-validation, and the built-in squared-hinge fit otherwise; `"penaltyLearning"` and `"native"` force the choice.

...	Additional arguments passed to `cpt_detect()` while building the curves.
x	A `ggcpt_penalty_model` (for `print()` and `coef()`).
object	A `ggcpt_penalty_model`.
newdata	A numeric vector (one series), or a list/matrix of series.

Value

A `ggcpt_penalty_model` object with `print()`, `coef()` and `predict()` methods.

The two scales differ and it matters: `coef()` gives an intercept plus one weight per feature **on the log-penalty scale**, which is where the interval regression is fitted, while `predict()` exponentiates and returns a penalty on the natural scale, the scale `cpt_penalty()` and `cpt_detect()` consume. So a coefficient of -0.04 on `log_n` is a multiplicative effect on the penalty, not an additive one.

Reading the coefficients

The signs are often not interpretable, and that is a property of the labels rather than of the fit. A target interval is open above whenever the largest penalty on the grid still achieves the minimum label error, which is the common case, because a large penalty usually keeps the one changepoint the labels ask for. When every series' interval is open above, any sufficiently large prediction is optimal, the problem does not pin the slopes, and the L_2 term settles them near zero with whatever sign the optimiser reached.

Two measurements on four series of very different length and noise, all with one-change labels, differing only in the data drawn. In one, every non-intercept coefficient came out slightly negative, so the predicted penalty *decreased* with n, the opposite of the $\log n$ growth a reader would expect from BIC, and not evidence of anything. In the other, every feature weight went to zero and the model became a **constant**: `predict()` returned the same penalty for all four series. Both fits put every prediction inside its target, which is the property the model is fitted for, and neither outcome says anything about how a penalty should scale.

So do not read a `ggcpt_penalty_model` as having discovered a relationship, and do not be surprised by a constant one.

If the coefficients need to mean something, the labels have to constrain the penalty from both sides: widen `penalties` until the largest one starts to over-segment, so the target intervals close above. `cpt_label_error_curve()` shows whether they do.

References

Hocking TD, Rigaill G, Vert J, Bach F (2013). "Learning sparse penalties for change-point detection using max margin interval regression." In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 172–180.

See Also

`cpt_label_error_curve()`, `cpt_penalty()`.

Examples

```
set.seed(2026)
series <- list(a = c(rnorm(60), rnorm(60, 4)),
               b = c(rnorm(80), rnorm(80, 2)))
labels <- list(a = as_cpt_labels(60, n = 120),
               b = as_cpt_labels(80, n = 160))
model <- cpt_learn_penalty(series, labels,
                           penalties = c(2, 8, 32, 128))
model
stats::predict(model, series$a)
```

`cpt_leverage` *Rank observations by influence*

Description

Orders the observations of a `cpt_influence()` result by how much perturbing them disturbs the segmentation, most influential first. The composite `leverage` score is the sum of three standardised components (the change in the number of changepoints, the largest movement of a changepoint, and the largest change in a segment parameter), so an observation that shifts a location without changing the count is still ranked.

Usage

```
cpt_leverage(object, ...)
```

Arguments

`object`	A `ggcpt_influence` object, or a `ggcpt` object (in which case `cpt_influence()` is run first).
`...`	Passed to `cpt_influence()` when `object` is a `ggcpt`.

Value

A tibble ordered most influential first, with columns `index`, `delta_n_cp`, `max_shift`, `param_shift` and `leverage`.

Rows with `leverage` = `NA` come first, and they are the most influential of all. `max_shift` and `param_shift` are undefined for a perturbation that left the engine with no changepoints at all (there is nothing to match against and no parameters to compare), so the composite score cannot be formed for an observation whose removal destroys the segmentation entirely. The `NA` is kept rather than filled in with a fabricated number; read the `delta_n_cp` column on those rows, which says how many changepoints were lost.

An `NA` here is always that case. If the *original* fit found no changepoints then `max_shift` is missing for every observation, the standardisation returns zeros rather than `NA`s, and every `leverage` is finite. A perturbation whose re-fit *failed* is not one of these: it carries `delta_n_cp` = `NA` as well and is sorted last, since nothing is known about it.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(40), rnorm(40, 4)), method = "pelt")
head(cpt_leverage(fit), 3)
```

`cpt_load_tcpd` *Download and cache the Turing Change Point Dataset*

Description

Fetches the benchmark of van den Burg and Williams (2020), real series from many domains each annotated independently by several human annotators, and caches it under `tools::R_user_dir()`. The multi-annotator structure is the point: scoring against a single "true" set silently discards the disagreement, and `cpt_metrics_annotated()` is built to keep it.

Usage

```
cpt_load_tcpd(name = NULL, cache_dir = NULL, refresh = FALSE, quiet = FALSE)
```

Arguments

`name`	Dataset name, a character vector of names, or `NULL` (the default) for the whole catalogue.
`cache_dir`	Where to cache. Defaults to `tools::R_user_dir("ggchangepoint", "cache")`.
`refresh`	Re-download even when a cached copy exists? Defaults to `FALSE`.
`quiet`	Suppress progress messages. Defaults to `FALSE`.

Value

With `name = NULL`, a tibble catalogue (`name`, `n_annotators`, `cached`), read from the annotations file alone, so listing it downloads no series. Otherwise a named list of `list(series, annotations, index, longname)` datasets, in the shape `cpt_benchmark()` takes.

Network access and licences

This function downloads from GitHub, so it needs a working connection and is skipped in every example and test here. The data belong to their original owners under their own licences; this package neither bundles nor redistributes them. A few TCPD series are not in the repository at all (their sources do not permit redistribution) and are reported as unavailable rather than silently omitted.

References

van den Burg GJJ, Williams CKI (2020). "An evaluation of change point detection algorithms." *arXiv preprint arXiv:2003.06222*. doi:10.48550/arXiv.2003.06222.

Examples

```
# Not run by R CMD check: every call downloads from the Turing Change
# Point Dataset's GitHub repository, and an example must not require
# network access.
## Not run:
cpt_load_tcpd()                       # the catalogue
d <- cpt_load_tcpd("nile")            # one dataset
cpt_benchmark(d, methods = c("pelt", "amoc"))

## End(Not run)
```

`cpt_methods` *Introspect available changepoint detection methods*

Description

Returns a tibble describing every method the package knows about (those that are wired, those a user has registered with `cpt_register_method()`, and those that are planned), along with their capabilities and installation status. Useful for discovering what can be run, what needs to be installed, and which methods expose the extras the diagnostics need (confidence intervals, a fitted signal, a posterior, a detector statistic, a solution path, a bandwidth to sweep).

Usage

```
cpt_methods(capabilities = TRUE)
```

Arguments

`capabilities`	Include the capability flag columns? Defaults to `TRUE`. Set `FALSE` for the compact 0.4.0-shaped table.

Value

A tibble with columns:

method Method name as passed to `cpt_detect()`.

change_in What types of change the method can detect.

engine The upstream R package that implements the method.

status `"available"` (wired in this release), `"registered"` (supplied by the user this session; see `cpt_register_method()`), or `"planned"` (future).

installed `TRUE` if the engine package is installed, `FALSE` if it is a `Suggests` engine that is missing, `NA` for planned and registered methods.

target_release What a planned method is waiting on: a release, or `"when on CRAN"` when the engine package itself is not available from CRAN. `NA` for methods that are already wired. Asking `cpt_detect()` for a planned method reports this rather than claiming the name does not exist.

multivariate, univariate, online, ci, fitted, posterior, statistic, path, scale_space Capability flags (omitted when `capabilities = FALSE`). `ci` means the engine supplies changepoint-location confidence intervals; `fitted` a length-n fitted signal; `posterior` a per-location posterior probability; `statistic` and `path` the internals rendered by `ggcpt_statistic()` and `ggcpt_solution_path()`, which error with the list of supporting engines when a result does not carry them.

`scale_space` is not one of those, despite sitting beside them. Nothing stores a scale space on a result: `cpt_scale_space()` computes one on demand by sweeping a multiscale detector's bandwidth over the series, so it works on *any* series and any result, a `pelt` fit included. What this column marks is the two engines that sweep can be run *with*, i.e. the domain of that function's own `method` argument: `subset(cpt_methods(), scale_space)$method`.

`online` means the *algorithm* is sequential (it consumes observations one at a time), and this table reports it because it governs how the method behaves in batch: an online detector's threshold is a rate per observation, so run over a whole series through `cpt_detect()` it reports roughly $n/\mathrm{arl0}$ changepoints by construction. It does **not** mean the method can be passed to `cpt_monitor()`, which takes its own three: `"edetector"`, `"cpm"` and `"ocd"`. The two sets overlap without coinciding: `bocpd` is an online algorithm this table marks but the monitor does not offer, and `edetector` is native to this package rather than a wrapped engine, so it has no row here at all.

See Also

`cpt_install_engines()` to install a whole family of the engines this table reports on; `cpt_detect()` to run one; `cpt_register_method()` to add your own.

Examples

```
cpt_methods()
# which methods can draw a confidence interval?
subset(cpt_methods(), ci)$method
```

---

cpt_metrics *Changepoint accuracy metrics*

Description

Computes standard accuracy metrics comparing predicted changepoints to ground truth, including precision/recall/F1 with margin, covering metric, Hausdorff distance, adjusted Rand index, annotation error, and MAE/RMSE of matched locations.

Usage

```
cpt_metrics(pred, truth, n, margin = 5)
```

Arguments

`pred`	Predicted changepoint indices (integer vector).
`truth`	Ground truth changepoint indices (integer vector).
`n`	Length of the series.
`margin`	Tolerance margin for matching (default 5).

Details

Precision/recall use a one-to-one matching: each truth may be claimed by at most one prediction (predictions are scanned in order and take the earliest unmatched truth within `margin`, which yields a maximum matching for interval-structured problems). The covering metric follows van den Burg and Williams (2020): the prediction-side partition is always well defined, so an empty `pred` scores the covering of the trivial single-segment partition rather than 0.

Value

A tibble with one row and the columns below. "Higher" and "lower" mark which direction is better, the same directions `cpt_benchmark()` ranks by.

`n`, `n_pred`, `n_truth` the series length and the two changepoint counts, after out-of-range indices are dropped.

`precision`, `recall`, `f1` higher is better. The fraction of predictions that matched a truth, the fraction of truths that were matched, and their harmonic mean, all under the one-to-one matching within `margin`, so they score *whether* a change was found, not how precisely it was located.

`covering` higher is better, in $[0, 1]$. The segmentation covering metric: each true segment's best Jaccard overlap with a predicted segment, averaged weighted by segment length. Unlike F1 it needs no margin and degrades smoothly with location error.

`hausdorff` lower is better, in observations. The largest distance from any changepoint on either side to the nearest one on the other: a worst-case location error, so one badly placed changepoint dominates it.

`rand_index` higher is better. The *adjusted* Rand index between the two segment labellings: 1 for identical partitions, 0 for chance agreement, and negative for worse than chance.

`annotation_error` lower is better. **A count difference only**: `abs(n_pred - n_truth)`. It says nothing about location, so a segmentation with the right *number* of changepoints in entirely the wrong *places* scores a perfect 0. Read it beside `covering` or `hausdorff`, never alone.

`mae_matched`, `rmse_matched` lower is better, in observations. Mean absolute and root-mean-square location error over the matched pairs only, so they describe how well the changepoints that were found are placed and ignore the ones that were missed. `NA` when nothing matched.

Degenerate cases, and which way each metric resolves them

Three of the twelve columns are ratios with a zero denominator when one side is empty, and the row does not resolve them all the same way, so the conventions are worth stating rather than inferring from a benchmark table:

- **Both empty.** The segmentation is exactly right: `precision`, `recall`, `f1`, `covering` and `rand_index` are all 1 and `annotation_error` is 0. (A metric that scored this 0 was a 0.4.0 bug.)
- **One side empty.** `precision` and `recall` are *0* rather than `NA` (finding nothing where there was a change, and finding a change where there was none, both score badly), and so is `f1`. `rand_index` is 0 for the same reason.
- **Nothing matched.** `hausdorff`, `mae_matched` and `rmse_matched` are `NA`, because they are distances and there is no pair to measure. They are also `NA` whenever either side is empty. This is the one place the row mixes conventions: an all-wrong answer gives `f1 = 0` and `mae_matched`

= NA in the same row. cpt_benchmark() treats an NA as the worst rank rather than dropping it.

An index outside 1..(n - 1) is dropped with a warning: locations follow the "left" convention, and an out-of-range one would corrupt the partition metrics rather than merely miss.

See Also

cpt_metrics_annotated() for scoring against multiple annotators, and cpt_benchmark() to run a method-by-dataset grid on these metrics.

Examples

```
cpt_metrics(c(100, 200), c(100, 200), n = 300)
cpt_metrics(c(101, 205), c(100, 200), n = 300, margin = 5)
```

---

cpt_metrics_annotated *Multi-annotator evaluation*

Description

Computes averaged covering and F1 scores against multiple annotation sets, as used in the Turing Change Point Dataset benchmark.

Usage

```
cpt_metrics_annotated(pred, annotations, n, margin = 5)
```

Arguments

pred	Predicted changepoint indices.
annotations	A list of ground-truth annotation vectors.
n	Length of the series.
margin	Tolerance margin (default 5).

Value

A tibble with one row: n, n_annotators, n_pred, and the **four** metrics this averages: precision, recall, f1 and covering. Each is a plain unweighted mean of the per-annotator value from cpt_metrics(), over all n_annotators of them.

This is a narrower table than cpt_metrics() returns, so a call moved from one to the other loses columns: n_truth (there is no single truth), and the location metrics hausdorff, mae_matched, rmse_matched, rand_index and annotation_error.

The three distance metrics are omitted for a reason worth stating, because it is not obvious: they are NA whenever an annotator shares no matched pair with the prediction, so averaging them would quietly divide by fewer annotators than n_annotators reports. Measured on three annotators against one prediction (list(c(100, 200), integer(0), 150) at $n = 300$), mae_matched was available

for **one** of the three and `hausdorff` for two, while `f1` and `covering` were finite for all three. Score those per annotator with `cpt_metrics()` and combine them yourself if you want them, so the divisor is yours to choose.

`covering` and `f1` are the pair the Turing Change Point Dataset benchmark reports, which is why they are the ones averaged here; `precision` and `recall` come along as F1's parts.

See Also

`cpt_metrics()` for the full single-truth table and what each column means.

Examples

```
# two annotators who disagree slightly about where the change is
cpt_metrics_annotated(c(100, 200),
                      annotations = list(c(98, 200), c(100, 203)),
                      n = 300, margin = 5)
```

---

cpt_min_detectable *The smallest detectable change*

Description

Inverts `cpt_power()`: searches for the change size at which the detector reaches a target power. The study-design counterpart of a power curve, and the number that belongs in a pre-registration.

Usage

```
cpt_min_detectable(
  n,
  sigma = 1,
  method = "pelt",
  power = 0.8,
  range = c(0.1, 5),
  n_sim = 100,
  tolerance = 5,
  change_in = "mean",
  location = 0.5,
  noise = "gauss",
  rho = 0,
  df = 3,
  tol = 0.05,
  max_iter = 12,
  seed = NULL,
  ...
)

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

Arguments

n	Series length.
sigma	Noise standard deviation. Defaults to 1.
method	Detection method. Defaults to "pelt".
power	Target detection probability. Defaults to 0.8.
range	Search range for the change size, in standard deviations. Defaults to c(0.1, 5).
n_sim	Replicates per evaluation. Defaults to 100; the answer is only as precise as this makes it, and the returned object records the Monte Carlo interval at the solution.
tolerance	A detection counts as finding the change when it falls within this many positions of it. Defaults to 5.
change_in	What changes. Defaults to "mean". The scenario jump describes, in units of sigma: "mean" shifts the mean by jump; "var" takes the noise standard deviation from sigma to sigma * (1 + jump); "meanvar" does both at once, shifting the mean by jump and the standard deviation to sigma * (1 + jump / 2), the same design cpt_scenarios() simulates; "slope" bends a flat line into one rising by jump * sigma over the series.
location	Changepoint position, as a fraction of n in $(0, 1)$ or an integer position. Defaults to 0.5. A vector runs one scenario per value.
noise	Noise model, passed to cpt_simulate().
rho	AR(1) parameter when noise = "ar1".
df	Degrees of freedom when noise = "t".
tol	Bisection tolerance on the change size. Defaults to 0.05.
max_iter	Maximum bisection steps. Defaults to 12.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Ignored.
x	A ggcpt_min_detectable object.

Value

A list with jump (the smallest change reaching power), achieved_power, mc_se, and the trace of evaluations, with a print() method.

See Also

cpt_power().

Examples

```
# n_sim = 10, max_iter = 3: the search is n_sim detector fits per
# iteration, and n_sim = 30 with max_iter = 4 measured 5.4 s against
# CRAN's 5 s example budget. Raise both for a usable answer.
cpt_min_detectable(n = 200, n_sim = 10, max_iter = 3, seed = 1)
```

`cpt_monitor` *A stateful sequential changepoint monitor*

Description

Creates a detector that consumes observations as they arrive and raises alarms, rather than segmenting a series that is already complete. Feed it with `cpt_update()`, read its alarm log with `alarms()`, and score it with `cpt_delay()`, because for an online method "did you find the location?" is the wrong question and "how long did you take, and how often do you false-alarm?" is the right one.

Usage

```
cpt_monitor(
  method = c("edetector", "cpm", "ocd"),
  baseline = NULL,
  alpha = 0.01,
  arl0 = 500,
  cpm_type = "Mann-Whitney",
  patience = 5000,
  deltas = c(0.5, 1, 2),
  reset = TRUE,
  relearn = 20,
  thresh = "MC",
  mc_reps = 100,
  ...
)

## S3 method for class 'ggcpt_monitor'
tidy(x, ...)

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

## S3 method for class 'ggcpt_monitor'
autoplot(object, plot_type = c("timeline", "statistic", "runlength"), ...)
```

Arguments

`method` Which sequential detector:

- `"edetector"` (default) a mixture Shiryaev-Roberts e-detector; see the section below.
- `"cpm"` **cpm**'s sequential change-point model, tuned by `ARL0`.
- `"ocd"` **ocd**'s high-dimensional online detector. **Multivariate only**: it tracks a projection of the whole vector and needs at least two coordinates, so it refuses a single series rather than falling back to a univariate statistic.

`baseline` A numeric vector (or, for `"ocd"`, a matrix with rows as time points) of pre-change training data used to estimate the in-control mean and scale. Required for `"edetector"` and `"ocd"`; optional for `"cpm"`, which has its own start-up period.

`alpha` Target false-alarm probability for `"edetector"`: the average run length under the null is at least `1 / alpha`, by optional stopping on the martingale $M_t - t$ (see Details). Defaults to `0.01`.

`arl0` Target in-control average run length for `"cpm"`. Defaults to `500`.

`cpm_type` Statistic for `"cpm"`. Defaults to `"Mann-Whitney"`.

`patience` Target patience (average run length) for `"ocd"`. Defaults to `5000`.

`deltas` Shift sizes, in baseline standard deviations, mixed over by `"edetector"`. Defaults to `c(0.5, 1, 2)`, each taken in both directions. The mixture is a uniform average, so adding shifts costs power at the ones already there rather than inflating the false-alarm rate; mix over the range you think the change could fall in, not over everything.

`reset` After an alarm, restart the detector (`TRUE`, the default) or leave it running? Restarting is what makes a monitor report *repeated* changes rather than latching on the first one.

`relearn` How many observations after an alarm are used to re-learn the in-control baseline, during which no further alarm can fire. Defaults to `20`. This matters more than it looks: a real change is *persistent*, so a detector that restarts against the stale pre-change baseline alarms again on the very next observation and keeps alarming for the rest of the series: the monitor reports one change as hundreds, and `cpt_delay()` then counts them all as false alarms. Set `relearn = 0` to switch the behaviour off and see every threshold crossing.

`thresh` Threshold rule for `"ocd"`: `"MC"` (default) calibrates by Monte Carlo against `patience`, or supply a numeric vector of three thresholds. The Monte Carlo calibration is the expensive part of building an `"ocd"` monitor (a minute or more at the default `patience`), so pass thresholds directly when you already have them, or lower `mc_reps` while exploring.

`mc_reps` Monte Carlo repetitions for the `"ocd"` threshold.

`...` Additional arguments passed to the engine's constructor.

`x` A `ggcpt_monitor` object (for `print()`).

`object` A `ggcpt_monitor` object (for `autoplot()`).

`plot_type` `"timeline"` (the monitored series with the alarms marked), `"statistic"` (the detection statistic at each alarm, against its threshold: a monitor records its statistic only when an alarm fires) or `"runlength"` (the gaps between alarms, which estimate the run length). For a multivariate monitor the timeline draws the first coordinate: the alarms are shared, so the rules are right whichever coordinate is shown, but the line is one of several.

Value

A `ggcpt_monitor` object: a list carrying the detector's state between calls, so it is meant to be passed to `cpt_update()` and re-assigned rather than read field by field. The parts worth reading are

`alarms` the tibble `alarms()` and `tidy()` return: one row per alarm, with `time`, `statistic` and `threshold`. Empty (zero rows) until something fires.

`t` how many observations the monitor has consumed through `cpt_update()`, **not** counting the baseline; an alarm's `time` is on this same clock. A monitor built by `cpt_replay()` also carries `offset`, the number of leading observations it trained on, so `time + offset` is a position in the replayed series (`cpt_delay()` applies it for you).

`data` every monitored observation so far, baseline excluded (the first coordinate, for a multivariate monitor), which is what `autoplot()` draws.

`method`, `alpha`, `n_baseline`, `reset`, `relearn` the settings this call fixed, kept so a later `cpt_update()` cannot silently disagree with them.

`state` the detector's internal statistics. Engine internals: their shape differs by `method` and is not part of the interface.

The e-detector, and why it is implemented rather than wrapped

Shin, Ramdas and Rinaldo (2023) give a nonparametric sequential framework with non-asymptotic control of the average run length, and it has no R implementation. The construction used here is the mixture Shiryaev-Roberts e-detector for a sub-Gaussian shift. For each candidate shift δ the increment is the likelihood ratio $e_t^{(\delta)} = \exp(\delta(X_t - \mu_0)/\sigma^2 - \delta^2/(2\sigma^2))$, which has unit mean under the null; the running statistic is $R_t^{(\delta)} = (1 + R_{t-1}^{(\delta)}) e_t^{(\delta)}$; the shifts are combined by *averaging*, $M_t = K^{-1} \sum_\delta R_t^{(\delta)}$; and an alarm is raised the first time $M_t \geq 1/\alpha$.

The averaging is not a detail. Under the null $M_t - t$ is a mean-zero martingale, so optional stopping at the alarm time τ gives $E_\infty[\tau] = E_\infty[M_\tau] \geq 1/\alpha$: a finite-sample lower bound on the in-control average run length, with no asymptotics and no calibration run. A convex combination of e-detectors is an e-detector; a *maximum* of them is not, and taking one silently multiplies the false-alarm rate by roughly the number of shifts mixed over. Every other detector in this package wraps published, separately maintained code; this one does not, and is labelled as such wherever it appears.

References

Shin J, Ramdas A, Rinaldo A (2023). "E-detectors: a nonparametric framework for sequential change detection." *The New England Journal of Statistics in Data Science*, **1**(2), 229–260. doi:10.51387/23NEJSDS49.

See Also

`cpt_update()`, `alarms()`, `cpt_replay()`, `cpt_delay()`.

Examples

```
set.seed(2026)
mon <- cpt_monitor("edetector", baseline = rnorm(100))
mon <- cpt_update(mon, rnorm(50))          # still in control
mon <- cpt_update(mon, rnorm(50, 3))       # a change arrives
alarms(mon)
```

`cpt_penalty` *Construct changepoint penalties*

Description

Helper to construct standard penalty values for use with changepoint detection methods. Returns a numeric penalty value.

Usage

```
cpt_penalty(type, n = NULL, k = 1, value = NULL, alpha = 1.01, series = NULL)
```

Arguments

`type`	Penalty type: `"None"`, `"BIC"` (or `"SIC"`), `"MBIC"`, `"AIC"`, `"Hannan-Quinn"`, `"sSIC"`, or `"Manual"`.
`n`	Series length (at least 3 for the $\log n$-based penalties). Required for BIC, MBIC, AIC, Hannan-Quinn, sSIC.
`k`	Number of parameters per changepoint (typically 2 for mean+variance, 1 for mean-only). Defaults to 1. The `"MBIC"` penalty additionally reads `k` as the number of changepoints being placed, in its $\log \binom{n}{k}$ term.
`value`	Numeric value for `Manual` type.
`alpha`	Exponent of the strengthened SIC (`"sSIC"`) penalty $k(\log n)^{\alpha}$; must exceed 1. Defaults to `1.01` (Fryzlewicz, 2014).
`series`	The series a learned penalty is predicted for. Required only when `type` is a `ggcpt_penalty_model` from `cpt_learn_penalty()`, in which case every other argument is ignored and the model's prediction for this series is returned.

Value

A numeric penalty value.

Penalty semantics across engines

The same penalty name may be interpreted differently by different engines:

- **changepoint-based methods** (PELT, BinSeg, SegNeigh, AMOC): accept character penalties (`"MBIC"`, `"BIC"`, `"AIC"`, `"Hannan-Quinn"`, `"None"`) and pass them to the upstream **changepoint** package. A numeric penalty is translated to that package's `penalty = "Manual"` plus `pen.value`. The one exception is Segment Neighbourhood, for which **changepoint** does not implement MBIC: `cpt_detect(method = "segneigh")` and `cpt_wrapper(cp_method = "SegNeigh")` therefore fall back to `"SIC"` when the default penalty is left in place, so a segneigh result is not directly penalty-comparable with a PELT one. Pass `penalty` explicitly to pin it. For a change in *mean* these engines also read the penalty on the data's own scale rather than a standardised one; see the scale-sensitivity section of `cpt_detect`.

- **Functional-pruning methods** (`fpop`, `cpop`, `decafs`): accept numeric penalties only. When a character penalty is supplied via `cpt_detect()`, it is resolved to a numeric value using `cpt_penalty()` before dispatch.
- **Search-based methods** (WBS, WBS2, NOT, MOSUM, IDetect, TGUH): use internal model-selection criteria (e.g., sSIC, threshold) and generally *ignore* the `penalty` argument. Specify thresholds via the wrapper's own arguments.
- `fastcpd` takes its penalty as `beta`, on its own scale, and defaults to its native `"MBIC"`. `cpt_detect()` forwards a numeric `penalty` as `beta`, and translates the three names the two packages share (`"MBIC"`, `"BIC"`/`"SIC"`, `"MDL"`). Any other character penalty (`"AIC"`, `"Hannan-Quinn"`, `"sSIC"`, `"None"`) has no **fastcpd** equivalent and is left to the engine's default rather than being silently approximated; pass `beta` yourself to pin it. Whatever is used is recorded on the result, so `print()` and `glance()` report the penalty of the fit in hand.
- **Inference/Bayesian methods** (`smuce`, `bcp`, `bocpd`, `beast`, `cpm`, `sn`): are tuned by a significance level, posterior-probability threshold, hazard, or average run length rather than a penalty; see each wrapper.
- `MBIC` in `cpt_penalty()` is a BIC-type penalty that adds a combinatorial term for the number of ways `k` changepoints can be placed in `n` observations, $0.5(k+1)\log n + \log\binom{n}{k}$. It is deliberately stronger than `"BIC"`. It is *not* the modified BIC of Zhang and Siegmund (2007), whose penalty is $1.5k\log n + 0.5\sum_i \log(l_i/n)$ on the **log-likelihood** scale (equivalently $3k\log n + \sum_i \log(l_i/n)$ on the deviance scale, which is how `cpt_select()` states it and the scale its `cost` column uses). It depends on the segment lengths l_i and so cannot be expressed by a function of `n` and `k` alone. Use the character `"MBIC"` with **changepoint**-based methods to get the engine's native MBIC, and `cpt_select(criterion = "mbic")` for the Zhang-Siegmund one.

Examples

```
cpt_penalty("BIC", n = 100)
cpt_penalty("AIC", n = 100)
cpt_penalty("Manual", value = 5)
```

`cpt_power` *Detection power for a changepoint scenario*

Description

Simulates replicate series under a scenario and reports how often the detector finds the change, how accurately it locates it, and how often it reports changes that are not there. This is the calculation that should precede an analysis, and the one applied papers are increasingly asked for.

Usage

```
cpt_power(
  n,
  jump,
  sigma = 1,
```

```
  method = "pelt",
  location = 0.5,
  n_sim = 200,
  tolerance = 5,
  change_in = "mean",
  noise = "gauss",
  rho = 0,
  df = 3,
  seed = NULL,
  parallel = TRUE,
  ...
)

## S3 method for class 'ggcpt_power'
tidy(x, ...)

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

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

Arguments

n	Series length.
jump	Size of the change, in units of sigma. A vector runs one scenario per value.
sigma	Noise standard deviation. Defaults to 1.
method	Detection method. Defaults to "pelt".
location	Changepoint position, as a fraction of n in $(0, 1)$ or an integer position. Defaults to 0.5. A vector runs one scenario per value.
n_sim	Replicates per scenario. Defaults to 200.
tolerance	A detection counts as finding the change when it falls within this many positions of it. Defaults to 5.
change_in	What changes. Defaults to "mean". The scenario jump describes, in units of sigma: "mean" shifts the mean by jump; "var" takes the noise standard deviation from sigma to sigma * (1 + jump); "meanvar" does both at once, shifting the mean by jump and the standard deviation to sigma * (1 + jump / 2), the same design cpt_scenarios() simulates; "slope" bends a flat line into one rising by jump * sigma over the series.
noise	Noise model, passed to cpt_simulate().
rho	AR(1) parameter when noise = "ar1".
df	Degrees of freedom when noise = "t".
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

parallel	Use future::plan() when **future.apply** is available? Defaults to TRUE. It has no effect unless a non-sequential plan is set, but when one is it changes where the replicates' random numbers come from; see the section below, which matters if the power figure is going into a paper.
...	Additional arguments passed to cpt_detect().
x	A ggcpt_power object.
object	A ggcpt_power object (for autoplot()).

Value

A ggcpt_power object: a tibble with one row per scenario, with columns n, jump, sigma, location, power (proportion of replicates detecting the change within tolerance), mc_se (the Monte Carlo standard error of that proportion), mean_abs_error (location error among detections; NaN when no replicate detected a changepoint within tolerance of the true one, because there is no distance to average, and power reads 0 in the same row), false_positives (mean number of *extra* changepoints per replicate) and n_sim; with print() and autoplot().

Reproducibility under a parallel plan

A seeded call is reproducible **for a given** future::plan(), and not across plans. Under a parallel plan the replicates' random numbers come from **future.apply**'s parallel-safe L'Ecuyer streams, derived from seed; run sequentially they come from the calling stream that seed set. Both are deterministic, and they are not the same numbers. Measured on cpt_power(n = c(100, 200), jump = 0.5,n_sim = 8, seed = 11): power = 0.25, 0.125 sequentially and 0, 0.375 on two workers. The scenario is named because the numbers depend on it and on the worker count; what does not depend on either is that the two disagree.

So the guarantee is: same seed and same plan, same answer, every time, whichever plan it is. If a power figure needs to be reproducible by someone else, pin the execution as well as the seed: pass parallel = FALSE, or state the plan alongside the seed. Raising n_sim narrows the gap, because it is Monte Carlo error rather than disagreement: both estimates are of the same quantity, and mc_se says how precisely.

This is specific to cpt_power(), which is the one function here whose parallel tasks consume random numbers. The other six that dispatch on future::plan() (cpt_benchmark(), cpt_batch(), cpt_consensus(), cpt_influence(), cpt_sensitivity() and ggcpt_compare()) farm out work that is deterministic given its input, and were measured to return identical results under a sequential and a two-worker plan, stochastic engines included.

See Also

cpt_min_detectable(), cpt_scenarios(), cpt_simulate().

Examples

```
pw <- cpt_power(n = 200, jump = c(0.5, 1, 2), n_sim = 30, seed = 1)
pw
ggplot2::autoplot(pw)
```

`cpt_recommend` *Recommend a detection method*

Description

Turns the capability matrix into an answer. Given what the analyst knows about their problem (how many dimensions, what kind of change, what the noise looks like, how long the series is, whether they need uncertainty or an online alarm), this returns the shortlist of methods that actually fit, with a reason for each and the reference to cite. It is a decision table, not a model: everything it knows is in `cpt_methods()`, and making that explicit and printable is the point.

Usage

```
cpt_recommend(
  dimension = c("univariate", "multivariate"),
  change_in = "mean",
  noise = c("iid", "heavy", "autocorrelated", "heteroscedastic"),
  n = NULL,
  need_uncertainty = FALSE,
  online = FALSE,
  installed_only = FALSE
)

## S3 method for class 'ggcpt_recommendation'
tidy(x, ...)

## S3 method for class 'ggcpt_recommendation'
print(x, top = 5, ...)
```

Arguments

`dimension`	`"univariate"` (default) or `"multivariate"`.
`change_in`	What kind of change is expected: any value accepted by `cpt_detect()`. Defaults to `"mean"`.
`noise`	Noise structure: `"iid"` (default), `"heavy"` (heavy-tailed), `"autocorrelated"`, or `"heteroscedastic"`.
`n`	Series length, used to flag methods that are impractical at that size. Optional.
`need_uncertainty`	Does the answer have to come with a confidence interval or significance region? Defaults to `FALSE`.
`online`	Is detection sequential (alarms as data arrive) rather than retrospective? Defaults to `FALSE`.
`installed_only`	Restrict to engines that are installed. Defaults to `FALSE`, so the recommendation names the right method even when it needs an install.
`x`	A `ggcpt_recommendation` object (for `print()`).

...	Ignored.
top	How many candidates to print. Defaults to 5.

Value

A tibble of candidate methods ordered by suitability, with columns method, engine, installed, score, why and caveat, and a print() that reads as advice.

See Also

cpt_methods(), cpt_consensus(), cpt_cite().

Examples

```
cpt_recommend(noise = "autocorrelated")
cpt_recommend(dimension = "multivariate", change_in = "covariance")
```

---

cpt_regions *Tidy the significance regions of a ggcpt object*

Description

The interval-valued methods (currently nsp_wrapper()) return regions rather than points: each is guaranteed to contain at least one changepoint at a prescribed *global* significance level. A region is a different object from a confidence interval around an estimate, so it lives in its own slot and has its own accessor rather than being folded into tidy().

Usage

```
cpt_regions(x)
```

Arguments

x	A ggcpt object.

Value

A tibble with columns start, end (positions), length, and, when the result carries a time index, start_index/end_index on the original scale. Any further columns the engine supplied are carried through after those: nsp_wrapper() adds value, the region's statistic. A zero-row tibble when the result carries no regions.

See Also

nsp_wrapper(), geom_cpt_region(), cpt_confint().

Examples

```
set.seed(2026)
fit <- as_ggcpt(50, c(rnorm(50), rnorm(50, 4)),
                regions = data.frame(start = 45, end = 56))
cpt_regions(fit)
```

`cpt_register_method` *Register an external changepoint detector*

Description

Teaches `cpt_detect()` about a detector this package does not (and often cannot) depend on: an engine that is not on CRAN, a Python detector reached through **reticulate**, a deep-learning model, a proprietary in-house method, or a hand-curated set of changepoints. The registered method then works with everything built on the `ggcpt` contract: `autoplot()`, the geoms, `tidy()`/`glance()`/`augment()`, `cpt_metrics()`, `ggcpt_compare()`, `cpt_stability()`, `cpt_consensus()`, `cpt_benchmark()` and `cpt_report()`.

Usage

```
cpt_register_method(
  name,
  fn,
  change_in = "mean",
  engine = "user",
  citation = NULL,
  capabilities = list(),
  cp_convention = c("left", "right"),
  overwrite = FALSE
)

cpt_unregister_method(name)

cpt_registered_methods()
```

Arguments

`name`	Method name, as it will be passed to `cpt_detect()`. Must not clash with a built-in method.
`fn`	A function called as `fn(x, ...)`. It may return either a `ggcpt` object (built with `as_ggcpt()`, say) or a bare vector of changepoint indices, which is coerced with `as_ggcpt()`. A returned `ggcpt` may carry a `$diagnostics` list, which is what makes `cpt_statistic()` and `cpt_solution_path()` work for the method. Two conventions apply to it. `diagnostics$solution_path` needs a `cp` column and may add `contrast`, `start` and `end`; it goes through the same filtering and step-numbering as a

	built-in path, so out-of-range candidates are dropped and `selected` is computed rather than trusted. `diagnostics$statistic` (a numeric vector, or a list with `statistic`, `label` and `threshold`) is padded with `NA` to the length of the series when it is shorter, and the pad is **centred**: a moving-window statistic is trimmed at both ends, so a left-aligned pad would shift every value by the bandwidth. Supply a full-length vector if that is not the alignment you want.
`change_in`	Character vector of `change_in` values the detector supports. Defaults to `"mean"`.
`engine`	Name of the package or system supplying the detector, for display in `cpt_methods()`. Defaults to `"user"`.
`citation`	Optional citation string returned by `cpt_cite()`. When `NULL`, `cpt_cite()` says plainly that the registration supplied none rather than inventing one.
`capabilities`	Optional named list of capability flags overriding the defaults (all `FALSE`): `multivariate`, `online`, `ci`, `fitted`, `posterior`, `statistic`, `path`, `scale_space`.
`cp_convention`	`"left"` (the changepoint is the last index of the left segment, this package's convention) or `"right"`. Used when `fn` returns bare indices.
`overwrite`	Replace an existing registration of the same name? Defaults to `FALSE`.

Value

Invisibly, the method name.

`cpt_registered_methods()` returns a tibble with one row per registered method (columns `method`, `change_in`, `engine`, `has_citation`), or a zero-row tibble when none are registered.

Registered methods are labelled, not endorsed

A registered method is visibly user-supplied: `cpt_methods()` gives it `status = "registered"`, `print()` on its results marks it, and `cpt_cite()` reports the citation you supplied or states that none was given. The package validates the *shape* of what your function returns (through `as_ggcpt()`, which runs the same contract checks as every built-in wrapper); it does not and cannot validate the method.

See Also

`as_ggcpt()` to turn changepoints into a result object without registering a method; `cpt_unregister_method()`; `cpt_registered_methods()`.

Examples

```
# A deliberately trivial detector: split at the largest jump.
cpt_register_method(
  "biggest_jump",
  fn = function(x, ...) which.max(abs(diff(x))),
  change_in = "mean",
  engine = "example",
  citation = "No citation supplied (illustration only)."
)
set.seed(1)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 5)), method = "biggest_jump")
```

```
fit
cpt_unregister_method("biggest_jump")
```

`cpt_replay` *Replay a series through a sequential detector*

Description

Runs a whole series through a monitor in one call, so the alarm timeline of an online method can be studied retrospectively without hand-rolling the update loop. This is the honest version of what 0.4.0's cpm, ocd and bocpd wrappers do implicitly: it reports *alarms*, which arrive after the change, rather than pretending they are estimated changepoint locations.

Usage

```
cpt_replay(x, method = c("edetector", "cpm", "ocd"), baseline = NULL, ...)
```

Arguments

`x`	A numeric vector, or a matrix with rows as time points.
`method`	Sequential detector; see `cpt_monitor()`.
`baseline`	Number of leading observations used as pre-change training data (an integer), or an explicit baseline vector. Defaults to `min(100, floor(n / 4))`.
`...`	Additional arguments passed to `cpt_monitor()`.

Value

A `ggcpt_monitor` that has already consumed the series: the same object `cpt_monitor()` returns, described there, with `alarms` filled in for whatever fired.

See Also

`cpt_delay()`, `cpt_monitor()`.

Examples

```
set.seed(2026)
mon <- cpt_replay(c(rnorm(200), rnorm(200, 3)), method = "edetector")
alarms(mon)
```

`cpt_report` *A reproducible report of a changepoint analysis*

Description

Assembles everything a reader needs to judge a changepoint result into one artifact: the method and its citation, the penalty, the number and locations of the changepoints with intervals where available, the segment table, stability and diagnostics if they were computed, and the session information. Reproducibility and correct attribution in one call.

Usage

```
cpt_report(
  object,
  format = c("md", "text", "gt"),
  file = NULL,
  stability = NULL,
  events = NULL,
  confint = TRUE,
  session = TRUE
)
```

Arguments

`object`	A `ggcpt` object.
`format`	`"md"` (default, GitHub-flavoured markdown as a character vector), `"text"` (plain text) or `"gt"` (a **gt** table of the changepoints, for a publication).
`file`	Optional path to write to. The report is returned invisibly when a file is written. Ignored for `format = "gt"`, which returns a table object rather than lines of text.
`stability`	Optional `cpt_stability()` result to include.
`events`	Optional `cpt_annotate_events()` result to include.
`confint`	Include a `cpt_confint()` table? Defaults to `TRUE`, which uses the engine's own intervals when it has them and skips the section otherwise (rather than silently bootstrapping, which would be slow and unexpected inside a report).
`session`	Include `sessionInfo()`? Defaults to `TRUE`.

Value

A character vector of report lines (or a **gt** table when `format = "gt"`).

See Also

`cpt_gt()`, `cpt_cite()`.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(60), rnorm(60, 4)), method = "pelt")
cat(cpt_report(fit, session = FALSE), sep = "\n")
```

`cpt_scale_space` *Scale space: the statistic across bandwidths*

Description

Sweeps a multiscale detector's bandwidth and returns the statistic at every (location, bandwidth) pair. The resulting heatmap answers the question a single-bandwidth fit cannot: *at which resolutions does this feature exist?* A change that is significant only at a wide bandwidth is a slow shift; one that appears only at a narrow bandwidth is a spike.

Usage

```
cpt_scale_space(x, bandwidths = NULL, method = c("mosum", "npmojo"), ...)

ggcpt_scale_space(x, bandwidths = NULL, method = c("mosum", "npmojo"), ...)
```

Arguments

`x`	A numeric vector (or, for `method = "npmojo"`, a matrix with rows as time points), or any `ggcpt` object. The sweep is computed here rather than read off the result, so a fit from any engine is accepted and the scale space is the one belonging to its series: from a `mosum` or `npmojo` result the engine is adopted as well (see `method`), and from anything else the default `"mosum"` sweep is drawn over the same series.
`bandwidths`	Integer vector of bandwidths to sweep. Defaults to a geometric grid between `max(5, n/50)` and `n/4`.
`method`	Which engine to sweep: `"mosum"` (default) or `"npmojo"`. Taken from `x` when it is a `ggcpt` produced by one of them.
`...`	Additional arguments passed to the engine at each bandwidth.

Value

`cpt_scale_space()` returns a tibble with `index`, `bandwidth`, `statistic`, `threshold`, `significant` (the statistic at that location and bandwidth exceeds the threshold) and `detected` (the engine reported a changepoint there at that bandwidth). The two differ: a location can clear the threshold without surviving the engine's own pruning. `ggcpt_scale_space()` draws the heatmap with the accepted changepoints overlaid.

See Also

`cpt_statistic()`.

Examples

```
set.seed(2026)
x <- c(rnorm(300), rnorm(300, 2))
ss <- cpt_scale_space(x, bandwidths = c(20, 40, 80))
head(ss)
```

`cpt_scenarios` *A grid of simulation scenarios*

Description

Builds the scenario grid a simulation study varies over, as data rather than as nested loops, so it can be inspected, filtered, subsetted and passed straight to `cpt_benchmark()`.

Usage

```
cpt_scenarios(
  n = 500,
  jump = c(0.5, 1, 2),
  location = 0.5,
  noise = "gauss",
  rho = 0.5,
  change_in = "mean",
  n_rep = 1,
  seed = 1,
  as_datasets = TRUE
)
```

Arguments

`n`	Series lengths.
`jump`	Change sizes, in standard deviations.
`location`	Change positions, as fractions of `n`.
`noise`	Noise models; any value `cpt_simulate()` accepts.
`rho`	AR(1) parameters (used by `noise = "ar1"`). Crossed with the other arguments for the `"ar1"` rows, like every other argument here; ignored, and recorded as `0`, for the others.
`change_in`	Change types.
`n_rep`	Replicates per scenario. Defaults to `1`.
`seed`	Base seed; replicate `r` of scenario `i` uses `seed + (i - 1) * n_rep + r`, so the whole grid is reproducible and every cell is independent. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`as_datasets`	Return simulated datasets in `cpt_benchmark()`'s shape (the default), or just the scenario table?

Value

A named list of datasets, or a tibble of scenarios when `as_datasets = FALSE`.

See Also

`cpt_power()`, `cpt_benchmark()`.

Examples

```
scen <- cpt_scenarios(n = 200, jump = c(1, 3), as_datasets = FALSE)
scen
```

`cpt_select` *Choose the number of changepoints*

Description

Builds a ladder of candidate segmentations with $K = 0, 1, \ldots$ changepoints and scores each one, returning the chosen K, the criterion curve behind the choice, and the fitted result at that K. Five criteria are available, including the consistent sample-splitting cross-validation of Zou, Wang and Li (2020) and the segment-length mBIC of Zhang and Siegmund (2007).

Usage

```
cpt_select(
  x,
  method = "pelt",
  criterion = c("bic", "mbic", "aic", "crops_elbow", "cv", "stability"),
  k_max = 20,
  folds = 5,
  B = 100,
  change_in = "mean",
  index = NULL,
  seed = NULL,
  ...
)

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

## S3 method for class 'ggcpt_selection'
tidy(x, ...)

## S3 method for class 'ggcpt_selection'
autoplot(
  object,
  plot_type = c("criterion", "segmentation", "ladder"),
```

```
  max_facets = 12,
  ...
)
```

Arguments

x	A numeric vector, or a ggcpt object (its series and method are used).
method	Detection method used to build the candidate ladder. Defaults to "pelt". Taken from x when it is a ggcpt. The penalised methods ("pelt", "binseg", "segneigh", "amoc", "fpop") give a full nested ladder; the search-based methods tune themselves by an internal criterion and largely ignore penalty, so their ladder collapses to one or two rungs and the function warns.
criterion	Which criterion selects K:

"bic" Gaussian BIC over the ladder, $n\log(\mathrm{RSS}/n)+(2K+1)\log n$. The first term is the Gaussian profile cost reported in the cost column ($-2\log L$ up to an additive constant, for a common variance); the parameter count is K locations plus $K+1$ segment means. Stated because the value column is otherwise not reproducible: both the cost convention and the parameter count vary between authors.

"mbic" the modified BIC of Zhang and Siegmund (2007), $3K\log n+\sum_i \log(l_i/n)$ on the **deviance** ($-2\log L$) scale, which is the scale the cost column is on; the same criterion is $1.5K\log n+0.5\sum_i \log(l_i/n)$ on the log-likelihood scale, which is how cpt_penalty() states it. It depends on the segment lengths l_i and so cannot be expressed by cpt_penalty()'s function of n and k alone. This is the one place in the package where the real Zhang-Siegmund penalty is computed.

"aic" Gaussian AIC over the ladder, $n\log(\mathrm{RSS}/n)+2(2K+1)$: the same cost and the same parameter count as "bic", with 2 in place of $\log n$. That penalty does not grow with n, so it over-selects changepoints, often taking every rung offered: on a 300-point series with changes at 100 and 200, "bic" and "mbic" both choose $K=2$ and "aic" chooses the largest K available. Included because people ask for it and because seeing the curve is instructive; "bic" or "mbic" is the better default.

"crops_elbow" the knee of the CROPS cost-against-K curve, made an explicit rule (maximum distance from the chord joining the endpoints, the standard Kneedle construction) rather than something eyeballed off a plot.

"cv" order-preserved sample-splitting cross-validation (COPPS) via **crossvalidationCP**. This is the criterion with a consistency guarantee. It chooses K with that package's own least-squares estimator (a change in the *mean*), not by re-running method, which then supplies the segmentation at that K. Note that cpss, the authors' own package, was removed from CRAN; **crossvalidationCP** is the supportable route.

"stability" the K whose changepoints are re-detected most often under within-segment bootstrap resampling. A robustness criterion, not a model-selection one; use it to cross-check the others.

k_max	Largest number of changepoints considered. Defaults to 20, capped at floor(n / 4).

`folds` Folds for `criterion = "cv"`. Defaults to `5`; `2` gives the original COPPS split.

`B` Bootstrap replicates for `criterion = "stability"`. Defaults to `100`.

`change_in` Passed to the detector, and inherited from `x` when `x` is a `ggcpt` (an explicit value still wins). Defaults to `"mean"`.

The three closed-form criteria assume a change in the mean whatever this is set to. `"bic"`, `"aic"` and `"mbic"` all score the ladder with the Gaussian profile cost $n \log(\mathrm{RSS}/n)$ (the deviance for a change in mean with a common variance), so on a `change_in = "var"` ladder the *candidates* come from the variance detector while the *score* does not, and splitting a segment whose mean did not move barely reduces RSS: the criterion will tend to choose $K = 0$ on a real variance change. Use `"stability"` there, which scores by re-detection with the same `change_in` and so carries no such assumption. `"cv"` does not help: its estimator is a least-squares change in the mean as well.

`index` Optional time index (a vector of dates, or a `ts`, `xts`, `zoo` or `tsibble` passed as `x`), carried onto the chosen fit so `tidy()` and `autoplot()` report the changepoint on your scale rather than as a position. Inherited from `x` when `x` is an indexed `ggcpt`.

`seed` Optional seed. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

`...` Additional arguments passed to `cpt_detect()` when the ladder is built by repeated detection.

`object` A `ggcpt_selection` object (for `autoplot()`).

`plot_type` `"criterion"` (the criterion against K, with the choice marked), `"segmentation"` (the series with the chosen segmentation) or `"ladder"` (small multiples showing how the segmentation coarsens as K falls, the display that makes the choice inspectable rather than asserted).

`max_facets` Maximum number of rungs drawn by `plot_type = "ladder"`. Defaults to `12`.

Value

A `ggcpt_selection` object: a list with `criterion_table` (one row per candidate K: `k`, `value`, `cost`, `chosen`, and a `cpts` list-column), `k` (the chosen number), `fit` (the `ggcpt` at that K), `criterion` and `data`. Methods: `print()`, `tidy()` and `autoplot()` with `plot_type = "criterion"`, `"segmentation"` or `"ladder"`.

References

Zou C, Wang G, Li R (2020). "Consistent selection of the number of change-points via sample-splitting." *The Annals of Statistics*, **48**(1), 413–439. doi:10.1214/19AOS1814.

Zhang NR, Siegmund DO (2007). "A modified Bayes information criterion with applications to the analysis of comparative genomic hybridization data." *Biometrics*, **63**(1), 22–32. doi:10.1111/j.15410420.2006.00662.x.

Haynes K, Eckley IA, Fearnhead P (2017). "Computationally efficient changepoint detection for a range of penalties." *Journal of Computational and Graphical Statistics*, **26**(1), 134–143.

See Also

cpt_crops(), cpt_penalty(), cpt_sensitivity().

Examples

```
set.seed(2026)
x <- c(rnorm(80), rnorm(80, 4), rnorm(80, 1))
sel <- cpt_select(x, criterion = "bic", k_max = 6)
sel
ggplot2::autoplot(sel)
ggplot2::autoplot(sel, plot_type = "ladder")
```

---

cpt_sensitivity *Sensitivity of a segmentation to its tuning parameters*

Description

The parameter analogue of cpt_influence(): instead of asking which observation drives the answer, it asks which *setting* does. Runs the detector over a grid of tuning values and reports the detected locations for each, which is the direct answer to the commonest reviewer question about a changepoint analysis: "is this robust to the penalty?".

Usage

```
cpt_sensitivity(x, method = "pelt", over = list(), seed = NULL, ...)

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

## S3 method for class 'ggcpt_sensitivity'
tidy(x, ...)

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

Arguments

x	A numeric vector (the series), or a ggcpt object, in which case its series and method are used.
method	Detection method. Taken from x when it is a ggcpt.
over	A named list of parameter vectors to sweep. Every combination is run, so keep the grid small: list(penalty = c(5, 10, 20), minseglen = c(2, 10)).
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments held fixed across the grid and passed to cpt_detect().
object	A ggcpt_sensitivity object (for autoplot()).

Value

A ggcpt_sensitivity object: a list with a grid tibble (one row per setting: the parameter columns, n_cp, and a cpts list-column), the data, and the swept parameter names. Methods: print(), tidy() (one row per detected changepoint) and autoplot() (a location heatmap over the grid).

See Also

cpt_influence(), cpt_select().

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 3))
s <- cpt_sensitivity(x, method = "pelt",
                     over = list(penalty = c(2, 10, 40)))
s
ggplot2::autoplot(s)
```

---

cpt_simulate *Generate simulated changepoint data*

Description

Creates a synthetic time series with known changepoints for testing and benchmarking.

Usage

```
cpt_simulate(
  n,
  changepoints = integer(),
  change_in = c("mean", "var", "meanvar", "slope"),
  params = NULL,
  noise = c("gauss", "t", "ar1", "rw"),
  sd = 1,
  df = 3,
  rho = 0,
  seasonality = NULL,
  sd_trend = NULL,
  seed = NULL
)

rcpt(...)
```

Arguments

n	Length of the series.
changepoints	Integer vector of changepoint locations (last index of each segment before the change).
change_in	What changes: "mean", "var", "meanvar", or "slope".
params	A list of parameters per segment. For mean changes, a vector of segment means. For var changes, a vector of segment sds. For meanvar, a list of lists with mean and sd per segment. For slope, a list with intercept and slope per segment, and the time origin **resets in every segment**: segment i's signal is $\text{intercept}_i + \text{slope}_i \cdot (1, \ldots, l_i)$ with the clock restarting at 1, so each intercept is the level its segment starts from. For a continuous piecewise-linear signal, start each segment where the previous one ended, $\text{intercept}_{i+1} = \text{intercept}_i + \text{slope}_i\, l_i$: with a change at 100, list(list(intercept = 0, slope = 1), list(intercept = 100, slope = -1)) rises to 100 and falls from there without a jump. A caller reasoning in absolute time, who writes segment 2 as the line $200 - t$ (intercept = 200), gets it starting at 199 instead of 99: the slope change plus an unrequested level jump. (Equal intercepts are continuous only when the earlier segment is flat.) When NULL, every segment gets the same neutral parameters, so the series has no actual change. changepoints sets the number of segments (k changepoints make $k + 1$ of them), and a mismatch in either direction warns rather than passing quietly: too few entries recycles the last one, so the trailing changepoints would otherwise be recorded as ground truth with no change behind them, and too many drops the surplus, so a caller who miscounted the changepoints would otherwise get an ordinary series back with a parameter silently unused.
noise	Noise type: "gauss" (Gaussian), "t" (Student-t), "ar1" (AR(1)), or "rw" (random walk).
sd	Noise standard deviation, non-negative (for Gaussian and t; t-noise is rescaled so its standard deviation is exactly sd). Defaults to 1.
df	Degrees of freedom for t-noise; must exceed 2 so the variance exists. Defaults to 3.
rho	AR(1) autocorrelation parameter, strictly between -1 and 1 for stationarity. Defaults to 0. Used only when noise = "ar1".
seasonality	Optional seasonal component added to the signal, as a list with period and amplitude (and optionally phase, in radians, and shape, either "sine" (the default) or "sawtooth"). A seasonal series is where the difference between a real level shift and a phase artefact starts to matter, and it is what bfast_wrapper() is built for; a detector that has never been shown one is untested against the case its users have.
sd_trend	Optional smoothly varying noise scale: a length-2 numeric giving the multiplier on sd at the first and last observation, interpolated log-linearly in between. Distinct from change_in = "var", which is piecewise constant: this is the *gradual* heteroscedasticity that makes constant-variance detectors shatter, and the condition HSMUCE, NSP-self-normalised and **fastcpd**'s variance families exist to handle.

seed	Optional seed for reproducibility. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Passed to cpt_simulate.

Value

A tibble with columns index, value, and seg_id. The true changepoints are stored in the true_changepoints attribute.

See Also

Other test signals: signal_blocks(), signal_fms(), signal_mix(), signal_stairs(), signal_teeth()

Examples

```
dat <- cpt_simulate(200, changepoints = c(100), change_in = "mean",
                    params = c(0, 10), seed = 2022)
attr(dat, "true_changepoints")

# a seasonal series with a level shift, and one with drifting noise
seasonal <- cpt_simulate(240, changepoints = 120, params = c(0, 3),
                         seasonality = list(period = 12, amplitude = 2),
                         seed = 1)
drifting <- cpt_simulate(240, changepoints = 120, params = c(0, 3),
                         sd_trend = c(0.5, 3), seed = 1)
```

---

cpt_solution_path *The solution path of a search-based detector*

Description

The order in which candidate changepoints entered the model, with the contrast (or split criterion) at each step. Binary segmentation splits recursively, WBS/WBS2/NOT/TGUH rank random intervals; in every case the final answer is a prefix of a path, and seeing the path shows how decisively each changepoint beat the next.

Usage

```
cpt_solution_path(object)

ggcpt_solution_path(object, max_steps = 40)
```

Arguments

object	A ggcpt object from an engine with a solution path (subset(cpt_methods(), path)$method).
max_steps	Longest prefix of the path drawn. Defaults to 40: a randomised search proposes hundreds of candidates and only the head of the ranking is readable.

Value

`cpt_solution_path()` returns a tibble with `step`, `cp`, `contrast` and, for interval-based searches, `start`/`end` of the interval that proposed it, plus a `selected` flag marking the changepoints in the final model. `ggcpt_solution_path()` draws it.

`contrast` is the engine's own ordering criterion, and it is a *different quantity* per engine: a penalty value for `binseg`/`segneigh` (from `changepoint::pen.value.full()`), $|CUSUM|$ for `wbs`, $|$`max.contrast`$|$ for `not`, and **breakfast**'s candidate criterion for `wbs2`/`tguh`. The values order the candidates within one result; they are not comparable across engines, and the plot legend names the quantity rather than calling all of them "Contrast".

For `wbs2` and `tguh` the path is **recomputed** with **breakfast**, because their fit objects do not keep the candidate list. `wbs2`'s search is randomised, so its path is a second search of the same series rather than a record of the first: it can differ between calls, and `selected` can be `FALSE` throughout if the recomputed candidates miss the fit's own changepoints. Every other engine's path is read off the fit.

See Also

`cpt_statistic()`, `cpt_crops()` for the penalty path of an optimal-partitioning method.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(150), rnorm(150, 3)), method = "binseg")
cpt_solution_path(fit)
```

---

cpt_stability *Changepoint stability diagnostics via bootstrap*

Description

Most engines report a point set of changepoints with no measure of how fragile it is. `cpt_stability()` fits the detector once, then resamples residuals *within* the fitted segments (so the estimated regime structure is preserved), re-runs the detector on each replicate, and reports how often each location is re-detected. The resulting detection-frequency profile is a cheap, model-agnostic confidence signal available for every wrapped engine, including the many that ship no confidence intervals.

Usage

```
cpt_stability(x, method = "pelt", B = 100, margin = 5, seed = NULL, ...)

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

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

Arguments

x	For cpt_stability(), a numeric vector; for the print() method, a ggcpt_stability object.
method	Detection method, passed to cpt_detect().
B	Number of bootstrap replicates. Defaults to 100.
margin	Tolerance (in indices) when counting a replicate detection as a re-detection of a location. Defaults to 5.
seed	Optional seed for reproducibility. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to every cpt_detect() call.
object	A ggcpt_stability object (for autoplot()).

Value

A ggcpt_stability object: a list with frequency (a tibble of index and freq, the proportion of replicates detecting a changepoint within margin of that index), original (the point-estimate ggcpt), and B. Methods: print() and autoplot() (frequency profile with the original detections marked).

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 4))
st <- cpt_stability(x, method = "pelt", B = 20)
st
ggplot2::autoplot(st)
```

---

cpt_statistic *The detector's statistic as a function of location*

Description

Returns the criterion the engine evaluated at each position, which is what explains *why* a changepoint landed where it did. Available for the engines that expose it (subset(cpt_methods(), statistic)$method lists them), and an error naming those engines for the ones that do not.

Usage

```
cpt_statistic(object)

ggcpt_statistic(object)
```

Arguments

object	A ggcpt object.

Value

`cpt_statistic()` returns a tibble with `index`, `statistic`, `threshold` (the engine's rejection threshold, or `NA`) and `label` (what the statistic is). `ggcpt_statistic()` returns a two-panel ggplot: the series above, the statistic below, sharing the x axis.

See Also

`cpt_solution_path()`, `cpt_scale_space()`.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(200), rnorm(200, 3)), method = "mosum")
head(cpt_statistic(fit))
ggcpt_statistic(fit)
```

---

cpt_test *Test detected changepoints*

Description

Attaches a test to each detected changepoint (`type = "jump"`) or to each fitted segment (`type = "segment"`), using the engine's own test where it has one and an explicitly *unadjusted* two-sample test where it does not.

Usage

```
cpt_test(object, type = c("jump", "segment"), correction = "none")
```

Arguments

`object`	A ggcpt object.
`type`	`"jump"` (one row per changepoint: is the change at this location real?) or `"segment"` (one row per segment: does this segment differ from the one before it?).
`correction`	Multiple-testing correction applied across the rows, one of the `p.adjust` methods (`"none"`, `"bonferroni"`, `"holm"`, `"BH"`, ...). Defaults to `"none"`; a `p_adjusted` column is added when it is not.

Value

A tibble with columns `cp` (or `seg_id`), `estimate`, `statistic`, `p_value`, `method` and `selection_adjusted`, plus `cp_index` on the original scale when the result carries a time index (as `cpt_confint()` does), and `p_adjusted` when `correction` is not `"none"`. `estimate` is the change in the segment mean (the breakpoint's slope change for **segmented**), and `NA` for a regression-mode **strucchange** fit, whose change is a vector of coefficients; its Chow test is run on the full regression model. A multivariate result is refused: both routes test one series.

Selection bias

read this before quoting a p-value: Testing a changepoint at a location that was *chosen because the data looked like it changed there* is circular, and the resulting p-values are anti-conservative, often severely. The `selection_adjusted` column records, per row, whether the test accounts for that:

- `TRUE` for **segmented**'s Davies test, which is built for a nuisance parameter present only under the alternative. It is one *global* test of "is there a breakpoint", not a test per breakpoint, so on a multi-break fit every row carries the same statistic and p-value, and the method string says so.
- `FALSE` for the generic Welch two-sample fallback, which compares the segments either side of the changepoint as if the location had been fixed in advance. Useful as a descriptive effect size with a scale attached; not a valid significance test for the existence of the change.
- `FALSE` for **strucchange**'s route as well, which is the Chow F evaluated *at* each estimated break date. The Chow statistic's reference distribution assumes the date was fixed in advance, so quoting it at a date the Bai-Perron dynamic program chose is exactly the circularity this column exists to flag. Reporting it is conventional in that literature, which does not make it adjusted. The selection-adjusted objects there are the sup-type statistics and the Bai-Perron critical values.

Two further limits worth knowing. `type = "segment"` is **always** the unadjusted Welch test: the native routes above apply only to `type = "jump"`, so a **segmented** fit tested per-segment does not use Davies' test. And the split above is by *engine and type*, not by engine alone. For a guarantee that survives selection, use `nsp_wrapper()` (regions with exact global coverage) or `cpt_confint()` with `method = "nsp"`. The canonical post-detection tests of Jewell, Fearnhead and Witten (2022) are implemented in `ChangepointInference`, which is not on CRAN; `cpt_register_method()` is the supported way to bring it in.

See Also

`cpt_confint()`, `nsp_wrapper()`.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(60), rnorm(60, 4)), method = "pelt")
cpt_test(fit)
```

`cpt_update` *Feed observations to a monitor*

Description

Pushes new data through a `cpt_monitor()`, updating its internal state and appending to its alarm log. The monitor is returned, so the idiom is `mon <- cpt_update(mon, new_obs)`.

Usage

```
cpt_update(monitor, new_obs)
```

Arguments

monitor	A ggcpt_monitor object.
new_obs	New observations: a numeric vector, or a matrix with rows as time points for a multivariate monitor.

Value

The updated ggcpt_monitor: the same structure as the input (see cpt_monitor()), with t advanced by the number of new observations (rows, for a matrix) and any new rows appended to alarms. Re-assign it: the monitor is a value, not a mutable object, so cpt_update(mon, y) without assignment discards the update.

See Also

cpt_monitor(), alarms().

Examples

```
set.seed(2026)
mon <- cpt_monitor("edetector", baseline = rnorm(100))
mon <- cpt_update(mon, c(rnorm(50), rnorm(50, 3)))
alarms(mon)
```

---

cpt_wrapper *Changepoint wrapper*

Description

This function wraps a number of cpt functions from the changepoint package and the cpt.np() function from the changepoint.np package. It is handy that users can use this function to get the same changepoint results as these functions output individually. Moreover, it returns a tibble that inherits the tidyverse style. Functions from the changepoint package do require data normality assumption by default, yet changepoint.np is a non-parametric way to detect changepoints and let data speak by itself. If user sets change_in as np (or cpt_np), a seed should be set before using the function for the sake of reproducibility. For more details on the changepoint and changepoint.np packages, please refer to their documentation.

Usage

```
cpt_wrapper(data, change_in = "mean_var", cp_method = "PELT", ...)
```

Arguments

data	A numeric vector.
change_in	Choice of mean_var, mean, var, and np (or cpt_np for backward compatibility). Each choice corresponds to cpt.meanvar(), cpt.mean(), cpt.var() and cpt.np() respectively. The default is mean_var.

cp_method	A wide range of choices (i.e., AMOC, PELT, SegNeigh or BinSeg). Please note when change_in is np or cpt_np, PELT is the only option.
...	Extra arguments for each cpt function mentioned in the change_in section.

Value

A tibble including which point(s) is/are the changepoint along with raw changepoint value corresponding to that changepoint. Changepoint locations follow the convention of the changepoint package: the last index of the left segment. The upstream cpt object is attached as the "ggcpt_fit" attribute, which is what cpt_detect() stores in the result's $fit.

Standardise the data for a change in mean

With change_in = "mean" the upstream Normal cost assumes a noise standard deviation of 1 and the penalty is compared against the raw residual sum of squares, so a series with wider noise is under-penalised and over-segmented: about 39 changepoints instead of 1 at $\sigma = 3$ on 200 observations (mean over 20 draws). Standardise the series first, or use change_in = "mean_var", which estimates a variance per segment and is unaffected. See the scale-sensitivity section of cpt_detect.

References

Killick R, Eckley I (2014). "changepoint: An R package for changepoint analysis." *Journal of statistical software*, **58**(3), 1–19.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2022)
cpt_wrapper(c(rnorm(100,0,1),rnorm(100,0,10)))
cpt_wrapper(c(rnorm(100,0,1),rnorm(100,10,1)))
```

decafs_wrapper *DeCAFS wrapper: changes amid drift and autocorrelated noise*

Description

Wraps `DeCAFS::DeCAFS()` (Romano, Rigaill, Runge and Fearnhead, 2022), which detects abrupt mean changes when the underlying signal also drifts (random-walk fluctuations) and the noise is AR(1)-autocorrelated, the two regimes in which plain change-in-mean methods over-detect. Model parameters are estimated automatically unless supplied.

Usage

```
decafs_wrapper(x, penalty = NULL, model_param = NULL, ...)
```

Arguments

`x`	A numeric vector.
`penalty`	Penalty β for adding a changepoint. Defaults to `2 * log(length(x))`. `cpt_detect` resolves its own `"MBIC"` default to a stronger numeric value (on a five-changepoint series that is 3 changepoints through the dispatcher against 5 here), so pass `penalty` explicitly when the two must agree.
`model_param`	Optional list of model parameters (`sdEta`, `sdNu`, `phi`) as accepted by `DeCAFS::DeCAFS()`; when `NULL` they are estimated from the data.
`...`	Additional arguments passed to `DeCAFS::DeCAFS()`.

Value

A `ggcpt` object. The `data` tibble carries the estimated signal in its `fitted` column.

References

Romano G, Rigaill G, Runge V, Fearnhead P (2022). "Detecting abrupt changes in the presence of local fluctuations and autocorrelated noise." *Journal of the American Statistical Association*, **117**(540), 2147–2162.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
res <- decafs_wrapper(c(rnorm(100), rnorm(100, 5)))
res$changepoints
```

`ecp_wrapper` *ecp wrapper*

Description

The ecp package provides a non-parametric way to detect changepoints. Unlike the changepoint package, it does not assume raw data to have any formal distribution. This wrapper function wraps two functions from the ecp package, i.e., `e.divisive()` and `e.agglo()`. Users can use either function by switching the `algorithm` argument. Before using the wrapper function, seed should be set for the sake of reproducibility.

Usage

```
ecp_wrapper(data, algorithm = "divisive", min_size = 2, seed = NULL, ...)
```

Arguments

`data`	A numeric vector (for univariate) or matrix/data.frame (for multivariate).
`algorithm`	Either `divisive` or `agglo`. `divisive` is the default.
`min_size`	Minimum number of observations between change points. By default is 2. This argument is only applied when `algorithm = "divisive"`.
`seed`	Optional. A seed for reproducibility of the stochastic permutation test. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`...`	Extra arguments to pass on either from `e.divisive()` or `e.agglo()`.

Value

A tibble includes which point(s) is/are the changepoint along with raw changepoint value corresponding to that changepoint. Changepoint locations follow the `ecp` package convention: the first index of the right segment. When no changepoint is found, an empty tibble is returned (0 rows). The upstream fit is not retained (`$fit` is `NULL` on a `ggcpt` from `cpt_detect(method = "ecp")`), because `ecp::e.agglo()`'s cluster-progression matrix is quadratic in the series length; call the **ecp** functions directly if you need their full output.

References

James NA, Matteson DS (2014). "ecp: An R package for nonparametric multiple change point analysis of multivariate data." *Journal of Statistical Software*, **62**(7), 1–25. doi:10.18637/jss.v062.i07.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2022)
ecp_wrapper(c(rnorm(100,0,1),rnorm(100,0,10)))
ecp_wrapper(c(rnorm(100,0,1),rnorm(100,10,1)))
```

---

envcpt_wrapper *EnvCpt wrapper: changepoints versus trends versus autocorrelation*

Description

Wraps `EnvCpt::envcpt()` (Beaulieu and Killick, 2018), which fits up to twelve competing models (constant mean or linear trend, each with or without changepoints, and with white-noise, AR(1) or AR(2) errors) and lets an information criterion decide whether the series really contains changepoints or merely trend/autocorrelation ("memory"). The changepoints of the winning model (if any) are returned, and the winning model's name is recorded, guarding against the classic false positive of running a mean-shift detector on autocorrelated data.

Usage

```
envcpt_wrapper(
  x,
  models = c("mean", "meancpt", "meanar1", "meanar2", "meanar1cpt", "meanar2cpt",
    "trend", "trendcpt", "trendar1", "trendar2", "trendar1cpt", "trendar2cpt"),
  criterion = c("AIC", "BIC"),
  minseglen = 5,
  ...
)
```

Arguments

`x`	A numeric vector.
`models`	Character vector of models to fit; see `EnvCpt::envcpt()`. Defaults to all twelve.
`criterion`	Model selection criterion: `"AIC"` (default) or `"BIC"`.
`minseglen`	Minimum segment length. Defaults to 5.
`...`	Additional arguments passed to `EnvCpt::envcpt()`.

Value

A ggcpt object. $fit holds the full envcpt output; the selected model name is stored in the penalty descriptor and printed by glance() via penalty_type. Individual model fits that fail are expected (the criterion ignores them), so the engine's own try() output is not passed on; genuine warnings still are, and a series on which no model fits at all raises an error. A constant series has no changepoints and is not handed to the engine, so $fit is NULL there.

References

Beaulieu C, Killick R (2018). "Distinguishing trends and shifts from memory in climate data." *Journal of Climate*, **31**(23), 9519–9543.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
res <- envcpt_wrapper(c(rnorm(100), rnorm(100, 3)))
res$changepoints
```

---

esac_wrapper *ESAC wrapper: sparsity-adaptive high-dimensional detection*

Description

Wraps HDCD::ESAC() (Moen, Glad and Tveten, 2023): Efficient Sparsity Adaptive Changepoint estimation for a change in the mean vector of a high-dimensional series. Where inspect projects onto a single estimated sparse direction, ESAC adapts across the whole sparsity range at once, which is a different regime rather than a refinement of the same one: it is competitive both when a handful of coordinates change and when all of them do.

Usage

```
esac_wrapper(
  x,
  threshold_d = 1.5,
  threshold_s = 1,
```

```
  empirical = FALSE,
  N = 1000,
  seed = NULL,
  ...
)
```

Arguments

x	A numeric matrix or data frame with rows as time points and columns as coordinates.
threshold_d, threshold_s	Leading constants of the dense and sparse thresholds. Defaults follow the engine (1.5 and 1).
empirical	Calibrate the thresholds by Monte Carlo rather than using the theoretical values? Slower but sharper; defaults to FALSE.
N	Monte Carlo samples when empirical = TRUE.
seed	Optional seed (used by the empirical calibration). The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to HDCD::ESAC().

Value

A ggcpt object. The changepoints tibble carries cusum (the ESAC statistic at each detected location) and depth (its level in the recursion).

References

Moen PAJ, Glad IK, Tveten M (2024). "Efficient sparsity adaptive changepoint estimation." *Electronic Journal of Statistics*, **18**(2), 3975–4038. doi:10.1214/24EJS2294.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
X <- matrix(rnorm(100 * 20), nrow = 100)
X[51:100, 1:5] <- X[51:100, 1:5] + 3
esac_wrapper(X)
```

`fabisearch_wrapper` *Network-structure changepoints via non-negative matrix factorisation*

Description

Wraps `fabisearch::detect.cps()` (Ondrus, Olds and Cribben, 2024): factorised binary search for changes in the *network structure* of a high-dimensional series. Each candidate split is scored by how much better a rank-r non-negative matrix factorisation fits the two halves separately than together, and significance is assessed by permutation.

Usage

```
fabisearch_wrapper(
  x,
  min_dist = 35,
  n_runs = 50,
  n_reps = 100,
  alpha = NULL,
  rank = NULL,
  n_core = 1,
  seed = NULL,
  ...
)
```

Arguments

`x`	A non-negative numeric matrix or data frame with rows as time points and columns as nodes. Non-negativity is a hard requirement of NMF, not a preference; see the section below.
`min_dist`	Minimum distance between changepoints. Defaults to `35` (the engine's default), lowered automatically when the series is too short for it.
`n_runs`	NMF runs per candidate split. Defaults to `50`.
`n_reps`	Replicates on each side of the significance test: the split is refitted `n_reps` times and the rows permuted `n_reps` times. Defaults to `100`; at least `2`, because the test compares two samples of this size.
`alpha`	Significance level applied to the p-value each candidate split receives. Defaults to `0.05`. That p-value is a two-sample test of the refitted losses against the permuted ones (a t-test unless `testtype` is passed through `...`), adjusted by Benjamini-Hochberg across the candidate splits. It is not a permutation p-value, so it has no `1 / n_reps` floor; the exact rank tests (`testtype = "wilcox"` or `"ks"`) do have one, and the wrapper warns when it is above `alpha`.
`rank`	NMF rank, a positive whole number. `NULL` estimates it with `fabisearch::opt.rank()`, which is expensive; supplying a rank is much faster.
`n_core`	Cores for the permutation stage. Defaults to `1`.

seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to fabisearch::detect.cps().

Value

A ggcpt object with change_in = "network". Multivariate input is reduced to **one series per observation by taking the cross-sectional mean** of the columns, and that is the series stored on the result: autoplot() draws it, tidy()'s cp_value reads it, and $segments$param_estimate and augment()'s .fitted/.resid are computed from it. It is not any one column of the input. The full input is kept in $data_wide for autoplot(type = "coordinates").

Non-negativity, cost, and the attached namespace

Three practical notes. (1) NMF is undefined for negative entries, so this wrapper refuses them rather than letting the engine fail deep inside a factorisation; shift or rescale the series first if it has negatives. (2) It is by far the most expensive engine here (n_runs times n_reps factorisations per candidate split), so the defaults are lowered in the examples. The engine's own progress output is suppressed, so a long call is silent until it returns. (3) **fabisearch** calls **NMF**'s multi-run machinery, which resolves helpers through the search path and fails with "none of the packages are loaded" when **NMF** is merely loaded; this wrapper therefore attaches **NMF** for the duration of the call and detaches it again afterwards.

References

Ondrus M, Cribben I (2024). "fabisearch: a package for change point detection in and visualization of the network structure of multivariate high-dimensional time series in R." *Neurocomputing*, **578**, 127321. doi:10.1016/j.neucom.2024.127321.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
# A change in *structure*, not in scale: two latent factors drive
# different halves of the node set before and after the change.
# Deliberately tiny -- this is by far the most expensive engine in the
# package (n_runs x n_reps factorisations per candidate split). Measured
# at 5-6 s across fresh sessions, against 27 s for the 2 x 25 /
# n_reps = 4
```

```
# version this replaced -- and this is the floor: `n_reps = 1` fails
# inside fabisearch with "not enough 'x' observations" (the permutation
# test needs two), and smaller matrices are not reliably cheaper because
# the search then evaluates more splits relative to `min_dist` (2 x 10 at
# min_dist = 8 measured 6.6 s). So this one example stays near CRAN's 5 s
# budget by necessity; `cran-comments.md` says so. Use the defaults on
# real data -- the settings here are for the budget, not for detection.
set.seed(2026)
block <- function(n, cols) {
  f <- abs(stats::rnorm(n)) + 0.5
  Y <- matrix(abs(stats::rnorm(n * 5)) * 0.2 + 0.1, n, 5)
  Y[, cols] <- Y[, cols] + f
  Y
}
Y <- rbind(block(12, 1:2), block(12, 3:5))
fabisearch_wrapper(Y, min_dist = 10, n_runs = 1, n_reps = 2,
                   alpha = 0.25, rank = 2)
```

`fastcpd_wrapper` *fastcpd wrapper: fast changepoint detection via sequential gradient descent*

Description

Wraps the **fastcpd** package (Li and Zhang, 2024), a modern PELT-family engine that pairs pruning with sequential gradient descent so that exact or near-exact segmentations of many model families run in near-linear time. This wrapper exposes the time-series families most useful alongside the other engines: mean, variance, mean-and-variance, and AR/ARMA/GARCH model changepoints.

Usage

```
fastcpd_wrapper(
  x,
  family = c("mean", "variance", "meanvariance", "ar", "arma", "garch"),
  order = NULL,
  ...
)
```

Arguments

`x`	A numeric vector, or (for `family` `"mean"`, `"variance"`, `"meanvariance"`) a matrix with one row per time point for multivariate detection.
`family`	Model family: `"mean"`, `"variance"`, `"meanvariance"`, `"ar"`, `"arma"`, or `"garch"`. Defaults to `"mean"`.
`order`	Model order for `"ar"` (a single integer), `"arma"` (length-2), or `"garch"` (length-2). Defaults to `1` for AR, `c(1, 1)` otherwise.

`...` Additional arguments passed to the corresponding `fastcpd::fastcpd.*()` function (e.g. `beta`, `trim`). `beta` is **fastcpd**'s penalty: a number, or one of its own names (`"MBIC"`, the default, `"BIC"`, `"MDL"`). It is recorded on the result, so `print()` and `glance()` report the penalty the fit actually used. `cpt_detect()` forwards a numeric `penalty` here and translates the three names it shares with **fastcpd**; see the penalty-semantics section of `cpt_penalty()`.

Value

A `ggcpt` object.

References

Li X, Zhang X (2024). "fastcpd: Fast change point detection in R." *arXiv preprint arXiv:2404.05933*.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
res <- fastcpd_wrapper(c(rnorm(100), rnorm(100, 4)))
res$changepoints
```

---

fcov_wrapper *Functional covariance changepoints*

Description

Wraps `fChange::fchange()` for changes in the covariance operator, eigenstructure or trace of a functional time series: the changes that leave the mean curve untouched.

Usage

```
fcov_wrapper(
  x,
  target = c("covariance", "trace", "eigenjoint", "eigensingle"),
  statistic = c("Tn", "Mn"),
  critical = c("simulation", "resample", "welch"),
```

```
  type = c("segmentation", "single"),
  alpha = 0.05,
  ...
)
```

Arguments

x	A numeric matrix or data frame with one row per time point and one column per grid location (the curve's resolution).
target	What to test: "covariance" (default), "trace", "eigenjoint" or "eigensingle". This is also by far the biggest lever on run time (see the timing section below), and note that the four answer different questions, so a cheaper one is a different test rather than a faster route to the same answer.
statistic	Test statistic: "Tn" (integrated, the default) or "Mn" (maximum).
critical	How critical values are obtained: "simulation" (default), "resample" or "welch". The first two draw random numbers and there is no seed argument, so call set.seed() first when the answer has to be reproducible.
type	"segmentation" (default, multiple changes) or "single" (one change).
alpha	Significance level. Defaults to 0.05.
...	Additional arguments passed to fChange::fchange().

Value

A ggcpt object with change_in = "covariance". Multivariate input is reduced to **one series per observation by taking the cross-sectional mean** of the columns, and that is the series stored on the result: autoplot() draws it, tidy()'s cp_value reads it, and $segments$param_estimate and augment()'s .fitted/.resid are computed from it. It is not any one column of the input. The full input is kept in $data_wide for autoplot(type = "coordinates").

For a covariance change this matters when reading the plot: a change in the covariance structure need not move the cross-sectional mean at all, so autoplot() can legitimately show changepoint rules on a series with no visible change in it. That is the detector working, not misfiring; use autoplot(type = "coordinates") to see the columns the change is in.

How long this takes

Minutes, not seconds, on a series of a hundred points: by a wide margin the most expensive engine in the package, and slow enough that a first call looks like a hung session. Timed on one Linux x86-64 machine, against fmean_wrapper() on the identical input so the comparison is the same package and the same data:

input	fmean	fcov
$n = 60$, $p = 5$	4.5 s	**316 s**
$n = 120$, $p = 5$	2.9 s	**598 s**

The cost is roughly linear in the number of time points and it is in the engine's own estimation rather than in this wrapper. It is, however, dominated by target, which the rest of this section used to deny. Measured at $n = 60$, $p = 6$, M = 50 on one Linux x86-64 machine:

target	time	changepoints found
`"covariance"` (default)	**477 s**	none
`"eigenjoint"`	21.7 s	none
`"eigensingle"`	21.7 s	none
`"trace"`	**2.1 s**	16, 30, 38

So the default is some two hundred times the cost of `"trace"`, and the example below uses `"trace"` for that reason. Read that as a choice of test and not as a free speedup: the trace is a scalar summary of the covariance operator, so it is a weaker instrument that happens to be cheap, and the row above is one series rather than a comparison of power. If a covariance change matters and the full operator test is the one you want, budget for it.

Two practical consequences either way: size the call before starting it, and do not put the default in a loop: a twelve-replicate study at $n = 120$ is two hours. Another machine will give different absolute numbers; the ratios are the part to plan around. The "Benchmarks" article compares the engines that do scale.

References

Aue A, Rice G, Sönmez O (2020). "Structural break analysis for spectrum and trace of covariance operators." *Environmetrics*, **31**(1), e2617.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
# See the note in ?fmean_wrapper on why this is 10 curves and M = 50.
X <- matrix(rnorm(60 * 10), nrow = 60)
X[31:60, ] <- X[31:60, ] * 3
fcov_wrapper(X, target = "trace", M = 50)
```

`fmean_wrapper` *Functional mean changepoints*

Description

Wraps `fChange::fchange()` for changes in the *mean function* of a functional time series: each observation is a curve, and the question is when the average curve shape changes. The binary-segmentation (`"segmentation"`) mode finds multiple changes; the `"single"` mode runs the one-change test and reports its p-value.

Usage

```
fmean_wrapper(
  x,
  statistic = c("Tn", "Mn"),
  critical = c("simulation", "resample", "welch"),
  type = c("segmentation", "single"),
  alpha = 0.05,
  robust = FALSE,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame with one row per time point and one column per grid location (the curve's resolution).
`statistic`	Test statistic: `"Tn"` (integrated, the default) or `"Mn"` (maximum).
`critical`	How critical values are obtained: `"simulation"` (default), `"resample"` or `"welch"`. The first two draw random numbers and there is no `seed` argument, so call `set.seed()` first when the answer has to be reproducible.
`type`	`"segmentation"` (default, multiple changes) or `"single"` (one change).
`alpha`	Significance level. Defaults to `0.05`.
`robust`	Use the robust (`"robustmean"`) statistic instead of the classical mean one? Defaults to `FALSE`.
`...`	Additional arguments passed to `fChange::fchange()`.

Value

A `ggcpt` object; the changepoints tibble carries the engine's `p_value` for each location. Multivariate input is reduced to **one series per observation by taking the cross-sectional mean** of the columns, and that is the series stored on the result: `autoplot()` draws it, `tidy()`'s `cp_value` reads it, and `$segments$param_estimate` and `augment()`'s `.fitted`/`.resid` are computed from it. It is not any one column of the input. The full input is kept in `$data_wide` for `autoplot(type = "coordinates")`.

References

Aue A, Rice G, Sönmez O (2018). "Detecting and dating structural breaks in functional data without dimension reduction." *Journal of the Royal Statistical Society: Series B*, **80**(3), 509–529. doi:10.1111/rssb.12257.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
# 10 curves and M = 50, not 20 and 200: the example is a demonstration,
# and the wider version measured 6.7 s against CRAN's 5 s budget.
X <- matrix(rnorm(60 * 10), nrow = 60)
X[31:60, ] <- X[31:60, ] + 2
fmean_wrapper(X, M = 50)
```

`fpop_wrapper` *FPOP wrapper: Functional Pruning Optimal Partitioning*

Description

Wraps the `fpop` package for optimal changepoint detection via functional pruning.

Usage

```
fpop_wrapper(x, penalty = NULL, ...)
```

Arguments

`x`	A numeric vector.
`penalty`	Penalty value. Defaults to `2 * log(length(x))` (BIC). This is an *absolute* penalty on the residual sum of squares, so it is only calibrated for noise of standard deviation 1: on wider data the default under-penalises badly and the segmentation shatters. Standardise the series, or scale the penalty by the noise variance (for example `2 * log(length(x)) * stats::var(diff(x)) / 2`). See

the scale-sensitivity section of cpt_detect. This default differs from the one cpt_detect applies, which resolves its "MBIC" default to a stronger numeric value, so the two entry points need not agree unless penalty is given.

... Additional arguments passed to fpop::Fpop().

Details

fpop was archived from CRAN in 2026 at its maintainer's request and is still built on R-Forge, where it is developed. Install it with install.packages("fpop", repos = "https://R-Forge.R-project.org"), or with cpt_install_engines("core").

Value

A ggcpt object.

References

Maidstone R, Hocking T, Rigaill G, Fearnhead P (2017). "On Optimal Multiple Changepoint Algorithms for Large Data." *Statistics and Computing*, **27**(2), 519–533. doi:10.1007/s1122201696363.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 4))
fpop_wrapper(x)$changepoints
# the default penalty assumes sd 1; scale it for wider noise
y <- 5 * x
fpop_wrapper(y, penalty = log(length(y)) * var(diff(y)))$changepoints
```

---

geomcp_wrapper *Geometrically-inspired multivariate changepoint wrapper (geomcp)*

Description

Wraps changepoint.geo::geomcp() (Grundy, Killick and Mihaylov, 2020): each multivariate observation is mapped to its distance from, and angle to, a reference point, and univariate PELT is run on the two mapped series. Distance changes capture shifts in magnitude, angle changes capture shifts in orientation/correlation structure.

Usage

```
geomcp_wrapper(
  x,
  penalty = "MBIC",
  mapping = c("both", "distance", "angle"),
  ...
)
```

Arguments

x	A numeric matrix or data frame with one row per time point.
penalty	Penalty for the univariate PELT runs (a **changepoint**-style character penalty). Defaults to "MBIC".
mapping	Which mapped series' changepoints to report: "both" (union, default), "distance", or "angle".
...	Additional arguments passed to changepoint.geo::geomcp().

Value

A ggcpt object whose changepoints tibble carries a mapping column ("distance" or "angle"; a location found in both is labelled "both").

References

Grundy T, Killick R, Mihaylov G (2020). "High-dimensional changepoint detection via a geometrically inspired mapping." *Statistics and Computing*, **30**, 1155–1166.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
X <- rbind(matrix(rnorm(100 * 4), 100), matrix(rnorm(100 * 4, 2), 100))
res <- geomcp_wrapper(X)
res$changepoints
```

`geom_changepoint` *Changepoint vertical rules geom*

Description

Draws vertical lines at changepoint locations. Mimics `geom_vline` but designed to work with the tidy changepoint data frames returned by the package. Can be used as a standalone layer: `geom_changepoint(data = cp_df, aes(xintercept = cp))`.

Usage

```
geom_changepoint(
  mapping = NULL,
  data = NULL,
  ...,
  na.rm = FALSE,
  show.legend = NA
)
```

Arguments

`mapping`	Set of aesthetic mappings created by `ggplot2::aes()`. Requires `xintercept`.
`data`	A data frame with changepoint information.
`...`	Other arguments passed to `geom_vline`.
`na.rm`	If `FALSE`, missing values are removed.
`show.legend`	Whether to show legend.

Value

A ggplot layer.

See Also

Other ggplot2 layers: `geom_cpt_ci()`, `geom_cpt_event()`, `geom_cpt_label()`, `geom_cpt_region()`, `geom_cpt_segment()`, `stat_changepoint()`

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
cp <- data.frame(cp = cpt_detect(d$y, method = "pelt")$changepoints$cp)
ggplot(d, aes(t, y)) + geom_line() +
  geom_changepoint(aes(xintercept = cp), data = cp, colour = "blue")
```

geom_cpt_ci *Changepoint confidence interval geom*

Description

Draws horizontal whiskers for changepoint-location confidence intervals (e.g. from MOSUM, stepR, strucchange, segmented).

Usage

```
geom_cpt_ci(mapping = NULL, data = NULL, ..., na.rm = FALSE, show.legend = NA)
```

Arguments

mapping	Aesthetic mappings. Requires y (the height at which to draw the whisker) together with xmin and xmax. An x aesthetic is accepted but not needed: the layer is a horizontal error bar, so the interval is given by xmin/xmax and the changepoint itself is usually marked with a separate point layer, as autoplot(show_ci = TRUE) does.
data	A data frame with CI information.
...	Other arguments passed to geom_errorbar (with orientation = "y"; geom_errorbarh() is deprecated).
na.rm	If FALSE, missing values are removed.
show.legend	Whether to show legend.

Value

A ggplot layer.

See Also

Other ggplot2 layers: geom_changepoint(), geom_cpt_event(), geom_cpt_label(), geom_cpt_region(), geom_cpt_segment(), stat_changepoint()

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
ci <- data.frame(xmin = 45, xmax = 56, y = 0)
ggplot(d, aes(t, y)) + geom_line() +
  geom_cpt_ci(aes(xmin = xmin, xmax = xmax, y = y), data = ci,
              inherit.aes = FALSE, width = 0.4, colour = "blue")
```

`geom_cpt_event` *Event annotation geom*

Description

Marks known real-world events on a changepoint plot: a vertical rule plus a text label. "Changepoint at index 147" is not a finding; "changepoint at 2020-03-11, matching the WHO pandemic declaration" is. `cpt_annotate_events()` produces the frame this layer expects and, crucially, also reports the changepoints no event explains and the events no changepoint found.

Usage

```
geom_cpt_event(
  mapping = NULL,
  data = NULL,
  ...,
  colour = "grey30",
  linetype = "dotted",
  angle = 90,
  size = 3,
  vjust = -0.4,
  hjust = 0,
  repel = NULL,
  inherit.aes = FALSE,
  na.rm = FALSE
)
```

Arguments

`mapping`	Aesthetic mappings. Requires `xintercept` for the rule and `label` for the text.
`data`	A data frame of events (see `cpt_annotate_events()`).
`...`	Other arguments passed to the text layer.
`colour`	Rule and text colour. Defaults to `"grey30"`.
`linetype`	Rule linetype. Defaults to `"dotted"`.
`angle`	Text angle in degrees. Defaults to `90`, because event labels are usually longer than the space between events.
`size`	Text size. Defaults to `3`.
`vjust, hjust`	Text justification.
`repel`	Use **ggrepel** to keep labels from overlapping? Defaults to `TRUE` when the package is installed. Requires `y` to be mapped as well (ggrepel places text, it cannot infer a height).
`inherit.aes`	Whether the label layer inherits the plot's aesthetics. Defaults to `FALSE`: an event table has its own columns, and inheriting the series' `x`/`y` mapping would look for columns that are not there.
`na.rm`	If `FALSE`, missing values are removed with a warning.

Value

A list of two ggplot layers (a rule and a label), which can be added to a plot exactly like a single layer.

See Also

cpt_annotate_events().

Other ggplot2 layers: geom_changepoint(), geom_cpt_ci(), geom_cpt_label(), geom_cpt_region(), geom_cpt_segment(), stat_changepoint()

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
ev <- data.frame(x = 50, label = "policy change")
ggplot(d, aes(t, y)) + geom_line() +
  geom_cpt_event(aes(xintercept = x, label = label), data = ev,
                 repel = FALSE)
```

---

geom_cpt_label *Changepoint label geom*

Description

Draws labelled regions behind a series: the central object of supervised changepoint detection (Hocking et al.), where an expert marks intervals as containing a change or not and the penalty is learned from those labels. Fill defaults to the label's change status, so a plot of labels reads as a picture of what the expert asserted; map `fill` to the `status` column of cpt_label_error() instead to get the correct / false-positive / false-negative display.

Usage

```
geom_cpt_label(
  mapping = NULL,
  data = NULL,
  ...,
  alpha = 0.25,
  colour = NA,
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)
```

Arguments

mapping	Aesthetic mappings. Requires xmin and xmax; ymin/ymax default to the panel extent. fill is commonly mapped to change or status.
data	A data frame of labels, e.g. cpt_labels() output.
...	Other arguments passed to ggplot2::geom_rect().
alpha	Fill transparency. Defaults to 0.25.
colour	Border colour. Defaults to NA (no border).
na.rm	If FALSE, missing values are removed with a warning.
show.legend	Whether to show a legend.
inherit.aes	Whether to inherit the plot's aesthetics. Defaults to FALSE.

Value

A ggplot layer.

See Also

cpt_labels(), cpt_label_error(), scale_fill_cpt_label().

Other ggplot2 layers: geom_changepoint(), geom_cpt_ci(), geom_cpt_event(), geom_cpt_region(), geom_cpt_segment(), stat_changepoint()

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
labs <- cpt_labels(c(40, 70), c(60, 90), c("change", "no_change"))
ggplot(d, aes(t, y)) +
  geom_cpt_label(aes(xmin = start, xmax = end, fill = change),
                 data = labs) +
  geom_line() +
  scale_fill_cpt_label()
```

---

geom_cpt_region *Significance region geom*

Description

Draws a changepoint that is an *interval*: a vertical band spanning xmin to xmax and the full height of the panel. This is the display Narrowest Significance Pursuit needs (nsp_wrapper()): NSP returns intervals each of which contains at least one changepoint at a prescribed *global* significance level, which is neither a point estimate nor a confidence interval around one. autoplot() adds this layer automatically for a result that carries regions.

Usage

```
geom_cpt_region(
  mapping = NULL,
  data = NULL,
  ...,
  alpha = 0.2,
  fill = "steelblue",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE
)
```

Arguments

mapping	Aesthetic mappings. Requires xmin and xmax; ymin/ymax default to the panel extent, so they need not be supplied.
data	A data frame of regions, e.g. cpt_regions() output.
...	Other arguments passed to ggplot2::geom_rect().
alpha	Fill transparency. Defaults to 0.2: light enough that the series stays readable through overlapping bands.
fill	Band fill colour. Defaults to "steelblue".
na.rm	If FALSE, missing values are removed with a warning.
show.legend	Whether to show a legend.
inherit.aes	Whether to inherit the plot's aesthetics. Defaults to FALSE: a region frame has its own columns and nothing to do with the series' x/y.

Value

A ggplot layer.

See Also

nsp_wrapper(), cpt_regions().

Other ggplot2 layers: geom_changepoint(), geom_cpt_ci(), geom_cpt_event(), geom_cpt_label(), geom_cpt_segment(), stat_changepoint()

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
regions <- data.frame(xmin = 44, xmax = 57)
ggplot(d, aes(t, y)) +
  geom_cpt_region(aes(xmin = xmin, xmax = xmax), data = regions) +
  geom_line()
```

`geom_cpt_segment` *Changepoint segment level geom*

Description

Draws horizontal segments representing the estimated level of each segment between changepoints. Typically used with data from `augment()`.

Usage

```
geom_cpt_segment(
  mapping = NULL,
  data = NULL,
  ...,
  na.rm = FALSE,
  show.legend = NA
)
```

Arguments

`mapping`	Aesthetic mappings. Requires `x`, `xend`, `y`, `yend`.
`data`	A data frame with segment information.
`...`	Other arguments passed to `geom_segment`.
`na.rm`	If `FALSE`, missing values are removed.
`show.legend`	Whether to show legend.

Value

A ggplot layer.

See Also

Other ggplot2 layers: `geom_changepoint()`, `geom_cpt_ci()`, `geom_cpt_event()`, `geom_cpt_label()`, `geom_cpt_region()`, `stat_changepoint()`

Examples

```
library(ggplot2)
set.seed(2026)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 4)), method = "pelt")
ggplot(fit$data, aes(index, value)) + geom_line(colour = "grey70") +
  geom_cpt_segment(aes(x = start, xend = end, y = param_estimate,
                       yend = param_estimate),
                   data = fit$segments, colour = "blue", linewidth = 1)
```

ggcptplot *Plot for the changepoint package*

Description

The plot for changepoints detected by the changepoint package is a line plot for the raw data and the vertical lines representing each changepoint. The x-axis is the row number of the raw data in the original data vector. The plot inherits ggplot2, meaning users can add ggplot2 functions on top the changepoint plot for customization.

Usage

```
ggcptplot(
  data,
  change_in = "mean_var",
  cp_method = "PELT",
  ...,
  cptline_alpha = 1,
  cptline_color = "blue",
  cptline_type = "solid",
  cptline_linewidth = 0.5,
  cptline_size = lifecycle::deprecated(),
  index = NULL,
  show_points = NULL,
  show_line = TRUE
)
```

Arguments

data	A numeric vector.
change_in	Choice of mean_var, mean, var, and np (or cpt_np for backward compatibility). Each choice corresponds to cpt.meanvar(), cpt.mean(), cpt.var() and cpt.np() respectively. The default is mean_var.
cp_method	A wide range of choices (i.e., AMOC, PELT, SegNeigh or BinSeg). Please note when change_in is np or cpt_np, PELT is the only option.
...	Extra arguments for each cpt function mentioned in the change_in section.
cptline_alpha	The value of alpha for the vertical changepoint line(s), default is 1, meaning no transparency.
cptline_color	The color for the vertical changepoint line(s), default is blue.
cptline_type	The linetype for the vertical changepoint line(s), default is solid.
cptline_linewidth	The linewidth for the vertical changepoint line(s), default is 0.5.
cptline_size	Deprecated. Use cptline_linewidth instead.
index	Optional. A vector of x-axis labels (e.g. dates) of the same length as data.

show_points	Logical. Whether to draw data points. Defaults to TRUE when length(data) <= 500, FALSE otherwise.
show_line	Logical. Whether to draw the line. Defaults to TRUE.

Value

A line plot with data points along with the vertical lines representing changepoints.

See Also

Other plotting: ggcpt_compare_table(), ggcpt_eval(), ggcpt_interactive(), ggecpplot()

Examples

```
ggcptplot(c(rnorm(100,0,1),rnorm(100,0,10)))
ggcptplot(c(rnorm(100,0,1),rnorm(100,10,1)))
```

---

ggcpt_compare *Compare multiple changepoint detection methods*

Description

Runs several detectors on the same data and returns a faceted or overlaid ggplot comparison. Respects future::plan() for parallel execution if the future.apply package is available.

Usage

```
ggcpt_compare(
  x,
  methods = c("pelt", "binseg", "amoc"),
  layout = c("facet", "overlay"),
  change_in = "mean",
  seed = NULL,
  ...
)
```

Arguments

x	A numeric vector (the data series). A one-column matrix or data frame is accepted; wider input is refused, because these detectors are univariate and flattening the columns would invent a changepoint at every seam. Use cpt_batch() for a panel of series.
methods	Character vector of method names (passed to cpt_detect).

layout	Layout type. "facet" (default) shows one panel per method; "overlay" draws all changepoints in one panel, colour-coded. The overlay **dodges** the rules horizontally so that two methods agreeing on an index are both visible, which moves each rule by up to half an observation, so read positions off "facet", or off tidy(), and treat the overlay as a picture of agreement rather than of location.
change_in	What to detect change in. Passed to each detector.
seed	Optional seed for reproducible parallelism. Passed to future.apply::future_lapply() as future.seed, and to set.seed() when running sequentially. Left NULL under a parallel plan, future.seed = TRUE is used, so the workers get parallel-safe streams but the run is not reproducible. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to each detector.

Value

A ggplot object.

Positions, not a time index

Unlike cpt_detect(), cpt_batch() and autoplot.ggcpt(), these two take no index: the input is reduced to a bare numeric vector, so a ts, xts, zoo or tsibble is plotted (and tabulated) in observation positions with an "Index" axis. To compare detectors on dated data, run cpt_detect(x, method = m, index = dates) per method and read tidy()'s cp_index, or plot the results with autoplot().

Examples

```
set.seed(2022)
x <- c(rnorm(100, 0, 1), rnorm(100, 10, 1))
ggcpt_compare(x, methods = c("pelt", "binseg"))
```

---

ggcpt_compare_table *Comparison table*

Description

Returns a tidy tibble combining the results of multiple detectors on the same series.

Usage

```
ggcpt_compare_table(
  x,
  methods = c("pelt", "binseg", "amoc"),
  change_in = "mean",
  ...
)
```

Arguments

x	A numeric vector (the data series). A one-column matrix or data frame is accepted; wider input is refused, because these detectors are univariate and flattening the columns would invent a changepoint at every seam. Use cpt_batch() for a panel of series.
methods	Character vector of method names.
change_in	What to detect change in.
...	Additional arguments passed to each detector.

Value

A tibble with columns method, cp, cp_value.

See Also

Other plotting: ggcpt_eval(), ggcpt_interactive(), ggcptplot(), ggecpplot()

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 5))
ggcpt_compare_table(x, methods = c("pelt", "binseg"))
```

---

ggcpt_eval *Evaluation visualization*

Description

Overlays predictions and ground truth on the series with tolerance windows, colouring true positives, false positives, and misses. Uses the same one-to-one matching as cpt_metrics(), so the plot and the metrics agree.

Usage

```
ggcpt_eval(pred, truth, data_vec, margin = 5)
```

Arguments

pred	Predicted changepoint indices.
truth	Ground truth changepoint indices.
data_vec	The original data vector (for context).
margin	Tolerance margin (default 5).

Value

A ggplot object.

See Also

Other plotting: ggcpt_compare_table(), ggcpt_interactive(), ggcptplot(), ggecpplot()

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 5))
fit <- cpt_detect(x, method = "pelt")
ggcpt_eval(fit$changepoints$cp, truth = 100, data_vec = x)
```

---

ggcpt_interactive *Interactive changepoint plot*

Description

Renders a ggcpt result (or any ggplot built from one) as an interactive HTML widget, with values on hover. A thin convenience wrapper: the static autoplot() path is untouched.

Usage

```
ggcpt_interactive(
  x,
  engine = c("plotly", "ggiraph"),
  width_svg = 8,
  height_svg = 5,
  ...
)
```

Arguments

x	A ggcpt object or a ggplot object.
engine	Which renderer: "plotly" (the default) rebuilds the plot in plotly's own model, which is richer but loses layers plotly does not know; "ggiraph" renders the ggplot itself to interactive SVG, so faceting and every layer survive and the result composes with other htmlwidgets. Neither is a dependency; whichever you ask for must be installed.
width_svg, height_svg	Figure size in inches for engine = "ggiraph".
...	Additional arguments passed to autoplot() when x is a ggcpt object.

Value

A **plotly** or **ggiraph** htmlwidget.

See Also

Other plotting: ggcpt_compare_table(), ggcpt_eval(), ggcptplot(), ggecpplot()

Examples

```
res <- cpt_detect(c(rnorm(50), rnorm(50, 5)), method = "pelt")
ggcpt_interactive(res)
```

`ggcpt_methods` *Coerce, format, and plot ggcpt objects*

Description

Convenience S3 methods for working with `ggcpt` objects: coerce the changepoints to a tibble or data frame, render a one-line summary string, or produce the default plot (a base-graphics fallback that delegates to `autoplot.ggcpt`).

Usage

```
## S3 method for class 'ggcpt'
as_tibble(x, ..., .name_repair = NULL)

## S3 method for class 'ggcpt'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S3 method for class 'ggcpt'
format(x, ...)

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

Arguments

`x`	A `ggcpt` object.
`...`	Additional arguments passed to methods.
`.name_repair`	Ignored (the changepoints tibble already has valid, unique names); present for signature compatibility with the generic.
`row.names, optional`	Passed to `as.data.frame`.

Value

`as_tibble()` and `as.data.frame()` return the changepoints table; `format()` returns a length-one character string; `plot()` draws the default plot and returns the `ggplot` object invisibly.

See Also

`ggcpt_plot_methods` for `plot()` on the other result classes, and `tidy.ggcpt()` for the tidy accessors.

Examples

```
set.seed(2022)
res <- cpt_detect(c(rnorm(50), rnorm(50, 5)), method = "pelt")
as_tibble(res)
as.data.frame(res)
format(res)
```

`ggcpt_plot_methods` *Base plot() methods for ggchangepoint result objects*

Description

Every result class in the package has an `autoplot()` method, but `plot()` is the reflex most users reach for first. Without a method, `plot()` on a list-shaped result falls through to `plot.default` and fails with `'x' is a list, but does not have components 'x' and 'y'`, a message that names neither this package nor `autoplot()`. These methods delegate to the corresponding `autoplot()` method so that `plot(result)` draws the intended figure.

Usage

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

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

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

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

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

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

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

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

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

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

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

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

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

Arguments

x	A result object created by one of the package's cpt_*() functions.
...	Passed to the corresponding autoplot() method.

Details

The plot is drawn as a side effect and the ggplot object is returned invisibly, so plot() works inside a loop or a function while p <- plot(result) still gives you the object to add layers to.

Value

The ggplot object, invisibly. Called for the side effect of drawing the plot.

See Also

autoplot.ggcpt; ggcpt_methods for the ggcpt class itself.

Examples

```
set.seed(2024)
x <- c(stats::rnorm(60), stats::rnorm(60, 4))

plot(cpt_select(x, method = "pelt", criterion = "bic", k_max = 4))
plot(cpt_crops(x, pen_min = 1, pen_max = 30))

# the object is still available to build on
p <- plot(cpt_stability(x, method = "pelt", B = 20, seed = 1))
p + ggplot2::labs(title = "Bootstrap stability")
```

ggcpt_posterior *Posterior probability plot for Bayesian results*

Description

Draws the classic Bayesian changepoint display: the series with its posterior mean (top panel) and the per-location posterior probability of a changepoint (bottom panel). Works with results from `bcp_wrapper()` and `beast_wrapper()`.

Usage

```
ggcpt_posterior(x, prob_threshold = NULL)
```

Arguments

`x`	A `ggcpt` object produced by a Bayesian wrapper.
`prob_threshold`	Probability cutoff drawn as a horizontal reference line in the probability panel; defaults to the threshold recorded on the object (or 0.5).

Value

A ggplot object (two facets sharing the x axis).

Examples

```
res <- bcp_wrapper(c(rnorm(60), rnorm(60, 4)), seed = 2026)
ggcpt_posterior(res)
```

ggcpt_runlength *Run-length posterior heatmap for Bayesian online results*

Description

Draws the signature BOCPD graphic: the posterior distribution of the run length (time since the last changepoint) at every observation, as a heatmap, with the series overlaid on top. Works with results from `bocpd_wrapper()`.

Usage

```
ggcpt_runlength(x, prob_floor = 0.001)
```

Arguments

`x`	A `ggcpt` object produced by `bocpd_wrapper()`.
`prob_floor`	Posterior probabilities below this value are not drawn (keeps the heatmap legible). Defaults to `1e-3`.

Value

A ggplot object.

Examples

```
res <- bocpd_wrapper(c(rnorm(60), rnorm(60, 4)))
ggcpt_runlength(res)
```

---

ggecpplot *Plot for the ecp package*

Description

The plot for changepoints detected by the ecp package is a line plot for the raw data and the vertical lines representing each changepoint. The x-axis is the row number of the raw data in the original data vector. The plot inherits ggplot2, meaning users can add ggplot2 functions on top the changepoint plot for customization.

Usage

```
ggecpplot(
  data,
  algorithm = "divisive",
  min_size = 2,
  ...,
  cptline_alpha = 1,
  cptline_color = "blue",
  cptline_type = "solid",
  cptline_linewidth = 0.5,
  cptline_size = lifecycle::deprecated(),
  index = NULL,
  show_points = NULL,
  show_line = TRUE
)
```

Arguments

`data`	A numeric vector (for univariate) or matrix/data.frame (for multivariate).
`algorithm`	Either `divisive` or `agglo`. `divisive` is the default.
`min_size`	Minimum number of observations between change points. By default is 2. This argument is only applied when `algorithm = "divisive"`.
`...`	Extra arguments to pass on either from `e.divisive()` or `e.agglo()`.
`cptline_alpha`	The value of alpha for the vertical changepoint line(s), default is 1, meaning no transparency.
`cptline_color`	The color for the vertical changepoint line(s), default is `blue`.

cptline_type	The linetype for the vertical changepoint line(s), default is solid.
cptline_linewidth	The linewidth for the vertical changepoint line(s), default is 0.5.
cptline_size	Deprecated. Use cptline_linewidth instead.
index	Optional. A vector of x-axis labels (e.g. dates) of the same length as data.
show_points	Logical. Whether to draw data points. Defaults to TRUE when length(data) <= 500, FALSE otherwise.
show_line	Logical. Whether to draw the line. Defaults to TRUE.

Value

A line plot with data points along with the vertical lines representing changepoints.

See Also

Other plotting: ggcpt_compare_table(), ggcpt_eval(), ggcpt_interactive(), ggcptplot()

Examples

```
ggecpplot(c(rnorm(100,0,1),rnorm(100,0,10)))
ggecpplot(c(rnorm(100,0,1),rnorm(100,10,1)))
```

---

glance.ggcpt *Glance at a ggcpt object*

Description

Returns a one-row summary of a changepoint detection result.

Usage

```
## S3 method for class 'ggcpt'
glance(x, ...)
```

Arguments

x	A ggcpt object.
...	Additional arguments (ignored).

Details

`total_cost` is reported on whatever scale the engine itself uses, so it is meaningful when comparing penalties within one method and not when comparing one method against another. For the **changepoint** engines it is the unpenalised $-2 \log L$ of the chosen segmentation. Four cases there are `NA` rather than filled with a number that would not mean the same thing:

- `"binseg"` and `"segneigh"`, whose `cpt.range` fits report the raw within-segment cost instead: for one and the same segmentation that is 219.7 where a PELT fit reports 659.9;
- `"np"`, because **changepoint.np** defines no `logLik` method;
- a change in *mean* under the default `"MBIC"` penalty. Loading **changepoint.np** (which this package imports, so it is always loaded) replaces **changepoint**'s `logLik` method for `cpt` objects with one that errors on exactly that combination. Any other penalty (`"BIC"`, `"AIC"`, a numeric value) reports normally, as do `change_in = "var"` and `"meanvar"`.

Value

A one-row tibble with columns: `n`, `n_changepoints`, `method`, `change_in`, `penalty_type`, `penalty_value`, `cp_convention`, `total_cost` (`NA` when the engine does not expose a cost), `runtime` (elapsed seconds when measured by `cpt_detect()`, otherwise `NA`).

---

hdcov_wrapper *High-dimensional covariance changepoints*

Description

Wraps `changepoints::BS.cov()` (Wang, Yu and Rinaldo): binary segmentation on the sample covariance operator, which detects a change in the *dependence structure* of a multivariate series even when every marginal mean and variance is unchanged. No other engine in the package can see that.

Usage

```
hdcov_wrapper(
  x,
  threshold = NULL,
  alpha = 0.05,
  n_perm = 20,
  delta = NULL,
  seed = NULL
)
```

Arguments

`x`	A numeric matrix or data frame, rows as time points.

threshold	Detection threshold on the CUSUM statistic. When NULL (the default) it is calibrated by permutation: the time order is shuffled n_perm times, which destroys any changepoint while preserving the marginal distributions, and the threshold is the 1 - alpha quantile of the largest statistic seen.
alpha	Family-wise level for the permutation threshold. Defaults to 0.05.
n_perm	Permutations used to calibrate the threshold. Defaults to 20; raise it for a sharper threshold at proportional cost.
delta	Minimum spacing between changepoints. Defaults to max(10, floor(n / 20)).
seed	Optional seed (the permutation calibration is random). The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A ggcpt object with change_in = "covariance"; the changepoints tibble carries the CUSUM statistic in cusum.

References

Wang D, Yu Y, Rinaldo A (2021). "Optimal covariance change point localization in high dimensions." *Bernoulli*, **27**(1), 554–575. doi:10.3150/20BEJ1249.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
p <- 5
A <- matrix(rnorm(100 * p), ncol = p)
B <- matrix(rnorm(100 * p), ncol = p)
B[, 2] <- B[, 1] + 0.2 * B[, 2]        # correlation appears
hdcov_wrapper(rbind(A, B), n_perm = 20, alpha = 0.05, seed = 1)
```

`hdreg_wrapper` *High-dimensional regression changepoints*

Description

Wraps `changepoints::CV.search.DP.regression()` (Rinaldo, Wang, Wen, Willett and Yu): dynamic programming with an ℓ_0 penalty for changes in the coefficient vector of a high-dimensional sparse regression, with the two tuning parameters chosen by cross-validation. Where `strucchange_wrapper()` dates breaks in a low-dimensional regression by dynamic programming on the residual sum of squares, this handles the case where there are more covariates than the segments have observations to fit them with.

Usage

```
hdreg_wrapper(
  x,
  response = NULL,
  gamma_set = NULL,
  lambda_set = NULL,
  delta = NULL,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame of covariates, rows as time points.
`response`	A numeric vector of responses, one per row of `x`. Required. Reachable through `cpt_detect()` as `cpt_detect(X, method = "hdreg", response = y)`.
`gamma_set`	Candidate values of the ℓ_0 tuning parameter. Defaults to a small grid scaled by the series length.
`lambda_set`	Candidate lasso penalties. Defaults to `c(0.01, 0.1, 1)`.
`delta`	Minimum spacing. Defaults to `max(5, floor(n / 20))`.
`...`	Additional arguments passed to the engine.

Value

A `ggcpt` object with `change_in = "regression"`. The plotted series is the *response*, which is what a reader of a regression-break plot expects to see.

References

Rinaldo A, Wang D, Wen Q, Willett R, Yu Y (2021). "Localizing changes in high-dimensional regression models." In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, 2089–2097.

See Also

var_wrapper(), strucchange_wrapper().

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
p <- 10
n <- 80
X <- matrix(stats::rnorm(n * p), n, p)
beta1 <- c(rep(2, 3), rep(0, p - 3))
beta2 <- c(rep(0, p - 3), rep(2, 3))
y <- c(X[1:40, ] %*% beta1, X[41:n, ] %*% beta2) + stats::rnorm(n)
hdreg_wrapper(X, response = y, gamma_set = c(1, 10),
              lambda_set = c(0.1, 1))
```

---

idetect_wrapper *Isolate-Detect wrapper*

Description

Wraps the IDetect package. Requires the IDetect package.

Usage

```
idetect_wrapper(x, seed = NULL, ...)
```

Arguments

x	A numeric vector.
seed	Optional seed for reproducibility. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to IDetect::ID().

Value

A ggcpt object. When the engine finds no changepoints (including when it signals "No change-points found"), an empty result is returned rather than an error. A constant series likewise returns the empty result; see the note below.

Constant input

IDetect::ID() does not treat a flat series consistently: its statistics become $0/0$, and what comes back depends on the value and the length. rep(3, 200) yields *126* changepoints, at 1, 3, 4, 6, 7, ...; rep(0, 100) raises "No change-points found"; rep(-2.5, 60) returns the sentinel 0. A constant series plainly has no changepoint, and every other search wrapper here reports none, so this one short-circuits to the empty result. Constancy is decided by exact equality, so a series with tiny but genuine variation still reaches the engine.

References

Anastasiou A, Fryzlewicz P (2022). "Detecting Multiple Generalized Change-Points by Isolating Single Ones." *Metrika*, **85**, 141–174. doi:10.1007/s00184021008216.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
idetect_wrapper(c(rnorm(60), rnorm(60, 4)))$changepoints
# a flat series has no changepoint; see "Constant input" above
nrow(idetect_wrapper(rep(3, 100))$changepoints)
```

---

inspect_wrapper *inspect wrapper: high-dimensional changepoints via sparse projection*

Description

Wraps InspectChangepoint::inspect() (Wang and Samworth, 2018). For a p-variate series whose mean changes in an unknown sparse subset of coordinates, the algorithm computes the CUSUM transformation, finds the optimal sparse projection direction via a convex relaxation, and locates changepoints on the projected univariate series, recursing via wild binary segmentation.

Usage

```
inspect_wrapper(x, lambda = NULL, threshold = NULL, ...)
```

Arguments

x	A numeric matrix or data frame with one row per time point and one column per coordinate.
lambda	Regularisation parameter of the sparse projection; when NULL the engine default $\sqrt{\log(p \log n)/2}$ is used.
threshold	Detection threshold; when NULL it is computed by Monte Carlo (via the engine).
...	Additional arguments passed to InspectChangepoint::inspect().

Value

A ggcpt object. The changepoints tibble carries a strength column (the maximum projected CUSUM statistic). The first coordinate is used for cp_value and the univariate plot line; the full matrix is kept for the faceted multivariate autoplot(). Coordinates that are constant carry no changepoint information and would make the engine's variance rescaling undefined, so they are dropped (with a warning) before detection and an all-constant matrix returns an empty result; the dropped coordinates are still kept for plotting, and reported locations always refer to the original rows.

References

Wang T, Samworth RJ (2018). "High dimensional change point estimation via sparse projection." *Journal of the Royal Statistical Society: Series B*, **80**(1), 57–83.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
X <- cbind(c(rnorm(80), rnorm(80, 3)), c(rnorm(80), rnorm(80, -2)),
           rnorm(160))
res <- inspect_wrapper(X)
res$changepoints
```

is_ggcpt *Test if an object is a ggcpt object*

Description

A class check, useful when a function accepts either a detection result or the raw series. It tests for `ggcpt` in the class vector, so a genuine `ggcpt` subclass (`ggcpt_consensus` is the one) returns `TRUE`. The other `ggcpt_*` classes in the package (`ggcpt_batch`, `ggcpt_benchmark`, `ggcpt_monitor`, `ggcpt_selection` and the rest) are *not* subclasses of `ggcpt` (most are tibble subclasses) and return `FALSE`.

Usage

```
is_ggcpt(x)
```

Arguments

`x`	An object to test.

Value

`TRUE` if `x` inherits from `ggcpt`.

See Also

Other result class: `annotate_segments()`, `as_cpt_series()`, `as_ggcpt()`, `cpt_annotations()`, `new_ggcpt()`, `print.ggcpt()`

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(40), rnorm(40, 4)), method = "pelt")
is_ggcpt(fit)
is_ggcpt(fit$changepoints)
```

kcp_wrapper *Kernel changepoint wrapper (KCP on running statistics)*

Description

Wraps `kcpRS::kcpRS()` (Cabrieto et al., 2018; the KCP framework of Arlot, Celisse and Harchaoui, 2019). The data are mapped to a running statistic (mean, variance, autocorrelation, or correlation) computed on a sliding window, and a Gaussian-kernel change point analysis with a permutation significance test is run on the statistic. Detecting changes in running correlations or variances captures higher-order changes that mean-based methods miss. Multivariate input (matrix or data frame) is supported.

Usage

```
kcp_wrapper(
  x,
  running_stat = c("mean", "var", "autocorr", "corr"),
  wsize = 25,
  nperm = 1000,
  kmax = 10,
  alpha = 0.05,
  seed = NULL,
  ...
)
```

Arguments

x	A numeric vector, matrix, or data frame (columns are variables).
running_stat	Which running statistic to monitor: "mean", "var", "autocorr", or "corr" (correlation requires at least two columns). Defaults to "mean".
wsize	Sliding window size for the running statistic. Defaults to 25.
nperm	Number of permutations for the significance test, at least 2. Defaults to 1000. Fewer than two leaves the engine with no permutation distribution: it reports no changepoints at all for 0, and fails inside its own code for 1.
kmax	Maximum number of changepoints considered. Defaults to 10.
alpha	Significance level of the permutation test. Defaults to 0.05.
seed	Optional seed for reproducibility of the permutation test. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to kcpRS::kcpRS().

Value

A ggcpt object. Reported locations refer to the centre of the sliding window in which the change occurs. The series must be at least wsize long to form one window. Constant coordinates make every running statistic NA, so they are dropped (with a warning) before detection and an all-constant input returns an empty result.

References

Arlot S, Celisse A, Harchaoui Z (2019). "A kernel multiple change-point algorithm via model selection." *Journal of Machine Learning Research*, **20**(162), 1–56.

Cabrieto J, Adolf J, Tuerlinckx F, Kuppens P, Ceulemans E (2018). "Detecting long-lived autodependency changes in a multivariate system via change point detection and regime switching models." *Scientific Reports*, **8**, 15637.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
res <- kcp_wrapper(c(rnorm(60), rnorm(60, 3)), nperm = 100, seed = 2026)
res$changepoints
```

`kwc_wrapper` *Robust depth-based changepoints for functional and multivariate data*

Description

Wraps **KWCChangepoint** (Ramsay and Chenouri): the functional Kruskal-Wallis covariance test, which ranks observations by statistical *depth* and segments on the ranks. Because it never touches the values themselves it is insensitive to heavy tails and outlying curves, which is exactly where the moment-based functional tests degrade.

Usage

```
kwc_wrapper(
  x,
  algorithm = c("fkwc", "dwbs"),
  depth = NULL,
  change_in = c("covariance", "distribution"),
  seed = NULL,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame with one row per observation (time point) and one column per grid location or coordinate.
`algorithm`	`"fkwc"` (default; pruned exact linear time over the depth ranks) or `"dwbs"` (depth-based wild binary segmentation).
`depth`	Depth function. For `"fkwc"` one of `"RPD"` (default), `"FM"`, `"LTR"`, `"FMd"`, `"RPDd"`; for `"dwbs"` one of `"spat"`, `"hs"`, `"mahal"`, `"mahal75"`.
`change_in`	Reported change type: `"covariance"` (default) or `"distribution"`. The test is sensitive to both; this only labels the result.

seed	Optional seed: the random-projection depths and the wild binary segmentation both randomise. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to the engine.

Value

A ggcpt object. Multivariate input is reduced to **one series per observation by taking the cross-sectional mean** of the columns, and that is the series stored on the result: `autoplot()` draws it, `tidy()`'s `cp_value` reads it, and `$segments$param_estimate` and `augment()`'s `.fitted`/`.resid` are computed from it. It is not any one column of the input. The full input is kept in `$data_wide` for `autoplot(type = "coordinates")`.

References

Ramsay K, Chenouri S (2025). "Robust changepoint detection in the variability of multivariate functional data." *Journal of Nonparametric Statistics*. doi:10.1080/10485252.2025.2503891.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
X <- matrix(rnorm(100 * 20), nrow = 100)
X[51:100, ] <- X[51:100, ] * 3
kwc_wrapper(X, seed = 1)
```

---

mcp_wrapper *Bayesian formula-based changepoint regression (mcp)*

Description

Wraps `mcp::mcp()` (Lindeløv): a Bayesian multiple-changepoint regression specified as a *list of formulas*, one per segment. This is the most expressive detector in the package: each segment can have its own intercept, slope, variance and autocorrelation, and the changepoints themselves get full posterior distributions rather than point estimates, summarised here as `ci_lower`/`ci_upper` on the changepoints tibble. (`ggcpt_posterior()` draws a *per-location* probability profile, which only **bcp** and **Rbeast** expose; it does not accept an `mcp` result.)

Usage

```
mcp_wrapper(
  x,
  change_in = c("mean", "slope", "var"),
  n_changepoints = 1,
  model = NULL,
  prior = list(),
  iter = 3000,
  adapt = 1000,
  chains = 3,
  seed = NULL,
  ...
)
```

Arguments

x	A numeric vector.
change_in	Shorthand for the segment model when model is not given: "mean" fits a change in intercept (list(y ~ 1, ~ 1)), "slope" a change in slope (list(y ~ 1 + t, ~ 0 + t)), "var" a change in residual standard deviation (list(y ~ 1, ~ 0 + sigma(1))).
n_changepoints	Number of changepoints in the shorthand model. Defaults to 1.
model	An explicit **mcp** model: a list of formulas. Overrides change_in and n_changepoints, and is the reason to reach for this engine at all.
prior	Optional named list of priors, passed to mcp::mcp().
iter, adapt, chains	Sampler settings, passed through.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to mcp::mcp().

Value

A ggcpt object. The changepoints tibble carries the posterior mean location together with ci_lower/ci_upper from the posterior quantiles, each converted from **mcp**'s continuous cp_1, cp_2, ... to the last observation before the change (ceiling(cp) - 1, because **mcp** starts a segment at x >= cp), and $data$fitted holds the posterior predictive mean, so autoplot(show_ci = TRUE, show_fit = TRUE) shows both.

JAGS is a system dependency

mcp samples through JAGS, a separate program installed outside R. Having the *package* is not the same as being able to *run* it: **mcp** imports **rjags**, and on some platforms **rjags** installs happily and only fails when it looks for the JAGS library at run time, in which case mcp::mcp() returns a fit carrying no posterior samples, with a warning rather than an error. This wrapper checks for that and says so plainly instead of failing several frames later inside summary(). Everything else in the package works without JAGS.

References

Lindeløv JK (2020). "mcp: An R package for regression with multiple change points." *OSF Preprints*. doi:10.31219/osf.io/fzqxv.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
# Not run by R CMD check: whether this works depends on a *system*
# library, and no test of installed R packages predicts that reliably --
# `rjags` can be present and still fail to find JAGS at run time.
## Not run:
set.seed(2026)
fit <- mcp_wrapper(c(rnorm(60), rnorm(60, 4)), iter = 500, adapt = 200)
fit

## End(Not run)
```

`mosum_wrapper` *MOSUM wrapper: Moving Sum*

Description

Wraps the `mosum` package for moving-sum-based changepoint detection, either at a single bandwidth or (with `multiscale = TRUE`) across a bandwidth grid with localised pruning.

Usage

```
mosum_wrapper(x, G = NULL, multiscale = FALSE, seed = NULL, ...)
```

Arguments

`x`	A numeric vector.
`G`	Bandwidth. If `NULL`, automatically selected (`min(n/10, 100)`, but never below 2, for the single-bandwidth procedure; the engine's default bandwidth grid for the multiscale procedure). A bandwidth of 1 leaves the engine's local variance estimate undefined, so the automatic choice is floored at 2.

multiscale	Logical. Use the multiscale MOSUM procedure (mosum::multiscale.localPrune()) instead of a single bandwidth? Defaults to FALSE.
seed	Optional seed for reproducibility. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to mosum::mosum() or mosum::multiscale.localPrune().

Value

A ggcpt object.

References

Eichinger B, Kirch C (2018). "A MOSUM Procedure for the Estimation of Multiple Random Change Points." *Bernoulli*, **24**(1), 526–564. doi:10.3150/16BEJ887.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 3))
mosum_wrapper(x, G = 20)$changepoints
mosum_wrapper(x, multiscale = TRUE)$changepoints
```

---

network_wrapper *Dynamic-network changepoints*

Description

Wraps changepoints::WBS.network() (Yu, Padilla, Wang and Rinaldo): wild binary segmentation on a sequence of networks, detecting the times at which the edge-probability structure changes. The input is one row per time point holding the *vectorised adjacency matrix*, so a series of $p \times p$ networks over n times is an $n \times p^2$ matrix.

Usage

```
network_wrapper(
  x,
  copy2 = NULL,
  n_intervals = 100,
  threshold = NULL,
  alpha = 0.05,
  n_perm = 20,
  delta = NULL,
  seed = NULL
)
```

Arguments

x	The network sequence: an $n \times p^2$ matrix of vectorised adjacency matrices, or an $n \times p \times p$ array.
copy2	An independent second observation of the same network sequence, in the same shape. The method's guarantees rest on sample splitting; see the section below for what happens when there is only one copy.
n_intervals	Number of random intervals. Defaults to 100.
threshold	Detection threshold. When NULL, calibrated by permutation exactly as in hdcov_wrapper().
alpha, n_perm	Level and number of permutations for that calibration.
delta	Minimum spacing. Defaults to max(5, floor(n / 20)).
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A ggcpt object with change_in = "network". The series it carries (and so the one autoplot() draws) is the **mean edge weight** at each time point, rowMeans() of the vectorised adjacency matrices. This is the one multivariate method with no data_wide slot: a $p \times p$ network has p^2 entries per time point, so a facet per coordinate would be unreadable. The changepoints are estimated from the networks themselves, not from the summary.

When you have only one copy of the network

WBS.network() takes two independent observations of the sequence, which is how the theory controls the bias of the squared-Frobenius statistic. Given a single sequence of *binary* networks this wrapper constructs the second copy by splitting each edge indicator at random (each present edge is assigned to one copy with probability one half), which is the usual independent-thinning device and is reported in a message. For weighted networks it splits the weight instead, which is exact for Poisson weights and approximate otherwise. If you have a genuine replicate, pass it as copy2 and none of this applies.

References

Yu Y, Padilla OHM, Wang D, Rinaldo A (2021). "Optimal network online change point localisation." *arXiv preprint arXiv:2101.05477*. doi:10.48550/arXiv.2101.05477.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
p <- 5
mk <- function(n, prob) {
  t(replicate(n, as.numeric(matrix(stats::rbinom(p * p, 1, prob), p))))
}
X <- rbind(mk(40, 0.2), mk(40, 0.6))
network_wrapper(X, n_intervals = 20, n_perm = 20, seed = 1)
```

`new_ggcpt` *Create a ggcpt object*

Description

The low-level constructor for the class every detector in this package returns. It assembles the components into a ggcpt without checking them, which is what makes it useful inside a wrapper and unsuitable as the entry point for hand-built input; use `as_ggcpt()` for that.

Usage

```
new_ggcpt(
  changepoints = tibble::tibble(cp = integer(), cp_value = numeric()),
  segments = tibble::tibble(seg_id = integer(), start = integer(), end = integer(), n =
    integer(), param_estimate = numeric()),
  data = tibble::tibble(index = integer(), value = numeric()),
  method = NA_character_,
  change_in = NA_character_,
  penalty = list(type = NA_character_, value = NA_real_),
  fit = NULL,
  call = NULL,
```

```
  cp_convention = "left",
  runtime = NA_real_
)
```

Arguments

changepoints	A tibble with columns cp and cp_value.
segments	A tibble with segment information: seg_id, start, end, n, param_estimate. param_estimate is the segment **mean** for every method in the package, including the variance, distribution and model-change detectors: it is the segment level, not the parameter that changed. A change_in = "var" result therefore has a param_estimate column that may barely move; read the variance off the data with the segment bounds if that is the quantity you want. Everything derived from this column inherits the convention: augment()'s .fitted/ .resid, cpt_gt()'s level columns, summary(), and the residual construction the bootstrap in cpt_confint() and cpt_stability() uses.
data	A tibble with index and value.
method	Character. The detection method used. A length-one string; defaults to NA_character_. (A zero-length value would make glance() return zero rows instead of its documented single row, because every other column would be recycled against it.)
change_in	Character. What was detected (e.g. "mean", "var", "meanvar"). A length-one string; defaults to NA_character_.
penalty	A list with type and value.
fit	The raw upstream object. Every wrapper stores one except "ecp": ecp::e.agglo() returns a cluster-progression matrix that is quadratic in the series length, so keeping it by default would make the result object explode on a long series. Call ecp::e.divisive() or ecp::e.agglo() directly if you need it. A few of the engines that *are* kept are still large relative to the data: measured on a 2000-point series, strucchange costs about 135 MB (a triangular $O(n^2)$ RSS matrix), bfast about 53 MB and bocpd about 31 MB, while every other engine stays under 4 MB. That is the engine's own object, not overhead this package adds, and it matters mainly when many results are held at once: cpt_batch(keep_fit = FALSE) drops them, or assign res$fit <- NULL yourself.
call	The matched call.
cp_convention	Character. The convention for reporting changepoint locations: "left" (last index of left segment, used by changepoint) or "right" (first index of right segment, used by ecp). Defaults to "left".
runtime	Numeric. Elapsed detection time in seconds, if measured. Defaults to NA.

Value

An object of class ggcpt, holding exactly the components passed in (documented one by one above) plus any of the optional slots listed below. Nothing is validated or derived: as_ggcpt() is the constructor that does both.

Optional slots

Beyond the components in the signature, a ggcpt may carry any of these, each present only when something supplied it and each safe to test for with is.null():

data_wide index plus one column per coordinate, for a multivariate result.

index, index_label a time index (one value per observation) and its axis label; see the index argument of cpt_detect().

regions a tibble of significance regions (start, end, ...) for the interval-valued methods; see nsp_wrapper() and geom_cpt_region().

diagnostics a named list of engine internals rendered by ggcpt_statistic(), ggcpt_solution_path() and ggcpt_scale_space().

registered TRUE when the result came from a user-registered detector rather than a wired engine.

as_ggcpt() is the validating way to build one of these from the outside; this constructor does not check its arguments.

See Also

Other result class: annotate_segments(), as_cpt_series(), as_ggcpt(), cpt_annotations(), is_ggcpt(), print.ggcpt()

Examples

```
set.seed(2026)
new_ggcpt(
  changepoints = tibble::tibble(cp = 50L, cp_value = 0.1),
  data = tibble::tibble(index = 1:100,
                        value = c(rnorm(50), rnorm(50, 4))),
  method = "manual", change_in = "mean")
```

---

not_wrapper *NOT wrapper: Narrowest-Over-Threshold*

Description

Wraps the not package for changepoint detection via the Narrowest-Over-Threshold method. The contrast determines what change is detected: piecewise-constant mean (default), mean and variance, or (continuous or discontinuous) piecewise-linear trend.

Usage

```
not_wrapper(x, contrast = "pcwsConstMean", seed = NULL, ...)
```

Arguments

x	A numeric vector.
contrast	Contrast type. One of "pcwsConstMean", "pcwsLinContMean", "pcwsLinMean", "pcwsConstMeanVar". Defaults to "pcwsConstMean".
seed	Optional seed for reproducibility. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to not::not().

Value

A ggcpt object whose change_in reflects the contrast: "mean", "meanvar", or "slope".

References

Baranowski R, Chen Y, Fryzlewicz P (2019). "Narrowest-Over-Threshold Detection of Multiple Change Points and Change-Point-Like Features." *Journal of the Royal Statistical Society Series B*, **81**(3), 649–672. doi:10.1111/rssb.12322.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
not_wrapper(c(rnorm(60), rnorm(60, 4)))$changepoints
# a change in slope, not in level: pick the contrast to match
trend <- c(1:60, 60 - 1:60) / 10 + rnorm(120)
not_wrapper(trend, contrast = "pcwsLinContMean")$changepoints
```

---

npmojo_wrapper *Nonparametric MOSUM wrapper (NP-MOJO)*

Description

Wraps CptNonPar::np.mojo() (McGonigle and Cho, 2025): nonparametric moving-sum detection of changes in the marginal or joint distribution of a (possibly multivariate) time series, robust to serial dependence.

Usage

```
npmojo_wrapper(x, G = NULL, lag = 0, ...)
```

Arguments

x	A numeric vector or matrix (rows are time points).
G	Moving-window bandwidth. Defaults to `max(20, 0.1 * n)` observations, capped at `n / 2` (the largest bandwidth the engine accepts), so the default also works on series shorter than 40.
lag	Time lag at which changes in the joint distribution are examined; `0` targets the marginal distribution. Defaults to `0`.
...	Additional arguments passed to `CptNonPar::np.mojo()`.

Value

A `ggcpt` object. Constant coordinates leave the kernel statistics undefined, so they are dropped (with a warning) before detection and an all-constant input returns an empty result. The engine calibrates its detection threshold by bootstrap, so the value recorded in the penalty descriptor varies between runs; call `set.seed()` beforehand, or pass `threshold = "manual"` and `threshold.val` through `...`, for a reproducible one.

References

McGonigle ET, Cho H (2025). "Nonparametric data segmentation in multivariate time series via joint characteristic functions." *Biometrika*, **112**(2), asaf024.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
res <- npmojo_wrapper(c(rnorm(100), rnorm(100, 3)))
res$changepoints
```

`nsp_wrapper` *NSP wrapper: Narrowest Significance Pursuit*

Description

Wraps the **nsp** package (Fryzlewicz 2024). NSP inverts the usual framing of post-selection inference: rather than estimating changepoint locations and then asking whether they are real, it returns a set of *intervals*, each of which contains at least one changepoint, with the guarantee holding *globally* across all intervals simultaneously at level `alpha`. The guarantee is exact and finite-sample, and the self-normalised and autoregressive variants keep it under heavy tails, heteroscedasticity and serial dependence.

Usage

```
nsp_wrapper(
  x,
  alpha = 0.1,
  variant = c("poly", "selfnorm", "ar", "tvreg"),
  change_in = c("mean", "slope"),
  deg = NULL,
  M = 1000,
  covariates = NULL,
  ord = 1,
  seed = NULL,
  ...
)
```

Arguments

`x`	A numeric vector.
`alpha`	Global significance level: with probability at least $1-\alpha$, *every* returned interval contains a changepoint. Defaults to `0.1`.
`variant`	Which NSP procedure to run: `"poly"` (default) `nsp::nsp_poly()`: piecewise polynomial signal, Gaussian noise of constant variance. `"selfnorm"` `nsp::nsp_poly_selfnorm()`: self-normalised, for heavy tails and heteroscedasticity. Slower. `"ar"` `nsp::nsp_poly_ar()`: autoregressive noise of order `ord`. `"tvreg"` `nsp::nsp_tvreg()`: a general linear model whose coefficients change; requires `covariates`.
`change_in`	`"mean"` (a piecewise-constant signal, `deg = 0`) or `"slope"` (piecewise linear, `deg = 1`). Ignored when `deg` is given explicitly, and when `variant = "tvreg"` (which takes its model from `covariates`).
`deg`	Degree of the piecewise polynomial. Derived from `change_in` when `NULL`.
`M`	Number of intervals drawn. Defaults to `1000`; the engine's own default.

covariates	A design matrix for variant = "tvreg", with one row per observation.
ord	AR order for variant = "ar". Defaults to 1.
seed	Optional seed. NSP draws random intervals, so a run is reproducible only with one. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to the underlying **nsp** function.

Value

A ggcpt object with a populated regions slot.

What cp means here, and what it does not

NSP produces no point estimates. This wrapper still fills the cp column (with the *midpoint* of each interval) because every downstream consumer in the package (augment(), cpt_metrics(), cpt_consensus(), autoplot()) is built on that column, and a result with an empty cp would silently score as "found nothing". The midpoint is **not** an estimate of the changepoint location and must not be reported as one: the interval is the inferential object. The result therefore

- carries the intervals in a regions slot, read with cpt_regions();
- marks itself, so print() says the cp column is a midpoint and autoplot() shades the bands by default;
- adds a cp_source column reading "region_midpoint" to the changepoints tibble.

References

Fryzlewicz P (2024). "Narrowest significance pursuit: inference for multiple change-points in linear models." *Journal of the American Statistical Association*, **119**(546), 1633–1646. doi:10.1080/01621459.2023.2211733.

See Also

cpt_regions(), geom_cpt_region(), cpt_confint().

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 4))
fit <- nsp_wrapper(x, M = 100, seed = 1)
```

```
cpt_regions(fit)
ggplot2::autoplot(fit)
```

`ocd_wrapper` *ocd wrapper: online high-dimensional changepoint detection*

Description

Wraps the ocd package (Chen, Wang and Samworth, 2022): online multiscale detection of a mean change in a high-dimensional stream, with worst-case detection-delay guarantees and per-observation cost independent of history. The detector assumes standardised data with known pre-change mean; this wrapper estimates the baseline mean and standard deviation from an initial training window, then monitors the remainder of the series, resetting after each declaration so multiple changes can be found.

Usage

```
ocd_wrapper(
  x,
  train = NULL,
  thresh = "MC",
  patience = 5000,
  beta = 1,
  mc_reps = 100,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame with one row per time point and at least two columns. The ocd detector is inherently high-dimensional and cannot be constructed for a single coordinate, so univariate input is rejected; use a univariate engine (see `cpt_methods()`) for one series.
`train`	Number of initial observations used to estimate the baseline mean/sd (not monitored). Defaults to `max(20, floor(0.2 * n))`, capped at `n/2`.
`thresh`	Threshold specification passed to `ocd::ChangepointDetector()`; `"MC"` (default) calibrates by Monte Carlo, which is what makes this the slowest wrapper (see the timing note below). Supplying the three thresholds directly, as a named numeric vector `c(diag =, off_d =, off_s =)`, skips calibration altogether.
`patience`	Target average run length to false alarm. Defaults to `5000`.
`beta`	Assumed lower bound on the squared Euclidean norm of the mean change. Defaults to `1`.
`mc_reps`	Monte Carlo repetitions for threshold calibration. Defaults to `100`. The cost is linear in this and grows with the number of coordinates; see the timing note below.
`...`	Additional arguments passed to `ocd::ChangepointDetector()`.

Value

A ggcpt object. Because the detector is online, reported locations are *declaration times* (the changepoint plus the detection delay). The declared_at column holds the same values as cp, and deliberately: **ocd** declares a change without also estimating where it began, so there is no separate location for the second column to carry. Compare cpm_wrapper(), whose engine supplies both, and whose cp is an estimated location with detection_time strictly later.

How long this takes

Nearly all of the run time is ocd's Monte Carlo threshold calibration, which happens before a single observation is read. It is linear in mc_reps and grows with the number of coordinates. Timed on one Linux x86-64 machine at mc_reps = 5, construction took about 10 s at $p = 3$, 22 s at $p = 10$ and 113 s at $p = 50$; raising mc_reps scales it linearly, so at $p = 3$ it was 38 s at mc_reps = 20 and 189 s at the default mc_reps = 100. The practical reading is that the default costs *minutes* rather than seconds even for a handful of coordinates, and better than half an hour at $p = 50$. Another machine will give different absolute numbers; the linearity in mc_reps is the part to plan around. Monitoring the observations afterwards is cheap by comparison: 0.37 s for a thousand of them at $p = 3$. Lower mc_reps while exploring (the example below uses 2, which measures 3.8 s), or pass thresh directly to skip calibration entirely, which brings the same fit down to a tenth of a second.

References

Chen Y, Wang T, Samworth RJ (2022). "High-dimensional, multiscale online changepoint detection." *Journal of the Royal Statistical Society: Series B*, **84**(1), 234–266.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
X <- rbind(matrix(rnorm(60 * 3), 60), matrix(rnorm(40 * 3, 3), 40))
# `mc_reps = 2`, not the default 100 and not the 5 this example used to
# pass: the calibration is linear in `mc_reps` and is nearly all of the
# cost, so 5 measured 9.7 s here against CRAN's 5 s budget and 2
# measures 3.8 s for the same answer. Neither is a calibration you
# would trust -- see the timing section above.
res <- ocd_wrapper(X, mc_reps = 2)
res$changepoints
```

`pilliat_wrapper` *Pilliat wrapper: high-dimensional detection by three complementary tests*

Description

Wraps `HDCD::Pilliat()` (Pilliat, Carpentier and Verzelen, 2023): a high-dimensional mean-change procedure combining a dense test, a Berk-Jones test and a partial-sum test, so it is powerful across sparsity regimes without estimating the sparsity level. A useful cross-check on `esac_wrapper()`: the two adapt differently and disagreeing answers are informative.

Usage

```
pilliat_wrapper(
  x,
  threshold_d_const = 4,
  threshold_bj_const = 6,
  threshold_partial_const = 4,
  empirical = FALSE,
  N = 100,
  seed = NULL,
  ...
)
```

Arguments

`x`	A numeric matrix or data frame with rows as time points and columns as coordinates.
`threshold_d_const`, `threshold_bj_const`, `threshold_partial_const`	Leading constants of the dense, Berk-Jones and partial-sum thresholds.
`empirical`	Calibrate the thresholds by Monte Carlo rather than using the theoretical values? Slower but sharper; defaults to `FALSE`.
`N`	Monte Carlo samples when `empirical = TRUE`.
`seed`	Optional seed (used by the empirical calibration). The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`...`	Additional arguments passed to `HDCD::Pilliat()`.

Value

A `ggcpt` object.

Dimension precondition

HDCD 1.1's `Pilliat()` builds one fewer partial-sum threshold than it uses whenever the number of coordinates is an exact power of two, so the C routine reads past the end of that vector and the engine reports a changepoint at *every* observation, on pure noise as readily as on a real change. This wrapper refuses those dimensions rather than returning the result, because it is wrong in a way that looks like a finding. `esac_wrapper()` is unaffected at every dimension. Note that constant coordinates are dropped before the count, so 9 coordinates one of which is constant is 8 for this purpose.

The same reasoning makes this the one engine that **errors** on a degenerate segmentation (a changepoint at more than 90% of observations) where every other engine in the package warns and reports what it found. That has a consequence for comparisons: `cpt_batch()` and `cpt_benchmark()` record a failure row for `pilliat` where the same output from `pelt` gives a result plus a warning, so a benchmark table is not scoring the two on equal terms in that case; read the `error` column alongside the metrics.

References

Pilliat E, Carpentier A, Verzelen N (2023). "Optimal multiple change-point detection for high-dimensional data." *Electronic Journal of Statistics*, **17**(1), 1240–1315. doi:10.1214/23EJS2126.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
X <- matrix(rnorm(100 * 20), nrow = 100)
X[51:100, 1:5] <- X[51:100, 1:5] + 3
pilliat_wrapper(X)
```

`print.ggcpt` *Print a ggcpt object*

Description

A compact header (method, what changed, how many changepoints, the convention their locations follow, the penalty and the series length) followed by the first ten changepoints. For the segment table and the fitted parameters use `summary()`; for the changepoints as data use `tidy()`.

Usage

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

Arguments

x	A ggcpt object.
...	Additional arguments (ignored).

Value

x, invisibly. Called for the side effect of printing.

See Also

summary.ggcpt(), tidy.ggcpt().

Other result class: annotate_segments(), as_cpt_series(), as_ggcpt(), cpt_annotations(), is_ggcpt(), new_ggcpt()

Examples

```
set.seed(2026)
print(cpt_detect(c(rnorm(40), rnorm(40, 4)), method = "pelt"))
```

---

scale_colour_cpt *Colour-vision-safe scales for changepoint methods*

Description

The discrete palette used wherever this package colours by method, series or class. It is the Okabe-Ito qualitative palette, which stays distinguishable under deuteranopia, protanopia and tritanopia, extended by recycling with a linetype change so that colour is never the only channel carrying the distinction.

Usage

```
scale_colour_cpt(..., na.value = "grey70")

scale_color_cpt(..., na.value = "grey70")

scale_fill_cpt(..., na.value = "grey70")

scale_linetype_cpt(...)
```

Arguments

...	Passed to ggplot2::discrete_scale().
na.value	Colour for missing values.

Value

A ggplot2 scale.

See Also

theme_ggcpt(), scale_fill_cpt_label().

Other accessibility scales: scale_fill_cpt_label(), theme_ggcpt()

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg, colour = factor(cyl))) +
  geom_point() + scale_colour_cpt() + theme_ggcpt()
```

---

scale_fill_cpt_label *Colour scales for changepoint labels and label errors*

Description

A colour-vision-deficiency-safe fill scale covering both vocabularies the supervised-detection displays use: the label's assertion (change / no_change / one_change) and the outcome of scoring a segmentation against it (correct / false_positive / false_negative). Unknown values fall back to grey rather than erroring, so a partially-scored frame still plots.

Usage

```
scale_fill_cpt_label(..., na.value = "grey70")

scale_colour_cpt_label(..., na.value = "grey70")
```

Arguments

...	Passed to ggplot2::scale_fill_manual() / ggplot2::scale_colour_manual().
na.value	Fill for values outside the vocabulary.

Value

A ggplot2 scale.

See Also

Other accessibility scales: scale_colour_cpt(), theme_ggcpt()

Examples

```
library(ggplot2)
labs <- cpt_labels(c(40, 70), c(60, 90), c("change", "no_change"))
ggplot(labs, aes(xmin = start, xmax = end, ymin = 0, ymax = 1,
                 fill = change)) +
  geom_rect() + scale_fill_cpt_label()
# the same palette covers correct / false_positive / false_negative
class(scale_colour_cpt_label())
```

`segmented_wrapper` *Broken-line regression wrapper (segmented)*

Description

Wraps `segmented::segmented()` (Muggeo, 2003, 2008): maximum likelihood estimation of *continuous* piecewise-linear ("broken-line") regressions, with standard errors and confidence intervals for the breakpoint locations. Where the step-change engines model jumps in the level, `segmented` models kinks in the trend, so `change_in` is `"slope"` and the fitted broken line is stored in the `fitted` column for `autoplot(show_fit = TRUE)`.

Usage

```
segmented_wrapper(x, npsi = 1, conf_level = 0.95, seed = NULL, ...)
```

Arguments

`x`	A numeric vector; a linear model of `x` on time `1:length(x)` is segmented.
`npsi`	Number of breakpoints to estimate. Defaults to `1`.
`conf_level`	Confidence level for breakpoint intervals. Defaults to `0.95`.
`seed`	Optional seed (the estimator uses bootstrap restarting). The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`...`	Additional arguments passed to `segmented::segmented()`.

Value

A `ggcpt` object with `ci_lower`/`ci_upper` columns and the fitted broken line in `$data$fitted`. Breakpoints are rounded to the nearest index; for a continuous fit the reported location is the kink itself. A constant series has no kink and returns an empty result, rather than the arbitrary breakpoint a singular fit would give.

References

Muggeo VM (2003). "Estimating regression models with unknown break-points." *Statistics in Medicine*, **22**(19), 3055–3071.

Muggeo VM (2008). "segmented: An R package to fit regression models with broken-line relationships." *R News*, **8**(1), 20–25.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
y <- cumsum(c(rep(0.5, 100), rep(-0.3, 100))) + rnorm(200)
res <- segmented_wrapper(y, npsi = 1)
res$changepoints
ggplot2::autoplot(res, show_fit = TRUE, show_ci = TRUE)
```

---

signal_blocks *Blocks test signal*

Description

The classic Donoho-Johnstone blocks test signal with known changepoints.

Usage

```
signal_blocks(n = 2048, seed = NULL)
```

Arguments

n	Length of the signal. Defaults to 2048.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A tibble with columns index and value. The true_changepoints attribute contains the known changepoint locations.

References

Donoho, D. L. and Johnstone, I. M. (1994). Ideal spatial adaptation by wavelet shrinkage. *Biometrika*, 81(3), 425-455.

See Also

Other test signals: cpt_simulate(), signal_fms(), signal_mix(), signal_stairs(), signal_teeth()

Examples

```
x <- signal_blocks(seed = 2026)
attr(x, "true_changepoints")
# PELT recovers all eleven Donoho-Johnstone jumps
cpt_detect(x$value, method = "pelt")$changepoints$cp
```

---

signal_fms *FMS (Four-Metric-Segments) test signal*

Description

A piecewise-constant test signal from the WBS/NOT literature.

Usage

```
signal_fms(n = 2000, seed = NULL)
```

Arguments

n	Length of the signal. Defaults to 2000.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A tibble with columns index and value.

See Also

Other test signals: cpt_simulate(), signal_blocks(), signal_mix(), signal_stairs(), signal_teeth()

Examples

```
x <- signal_fms(seed = 2026)
cp <- cpt_detect(x$value, method = "pelt")$changepoints$cp
# the smallest jumps (0.5, against noise sd 0.5) are the ones missed
cpt_metrics(cp, attr(x, "true_changepoints"), n = nrow(x))$covering
```

signal_mix *Mix test signal*

Description

A piecewise-constant/linear signal from the literature.

Usage

```
signal_mix(n = 2000, seed = NULL)
```

Arguments

n	Length of the signal. Defaults to 2000.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A tibble with columns index and value.

See Also

Other test signals: cpt_simulate(), signal_blocks(), signal_fms(), signal_stairs(), signal_teeth()

Examples

```
x <- signal_mix(seed = 2026)
attr(x, "true_changepoints")
# the linear ramps are not level shifts, so a mean-change detector puts
# changepoints inside them rather than at the segment joins
cpt_detect(x$value, method = "pelt")$changepoints$cp
```

signal_stairs *Stairs test signal*

Description

A monotonically stepping signal (staircase).

Usage

```
signal_stairs(n = 2000, seed = NULL)
```

Arguments

n	Length of the signal. Defaults to 2000.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A tibble with columns index and value.

See Also

Other test signals: cpt_simulate(), signal_blocks(), signal_fms(), signal_mix(), signal_teeth()

Examples

```
x <- signal_stairs(n = 500, seed = 2026)
attr(x, "true_changepoints")   # ten steps, so nine changes
cpt_detect(x$value, method = "pelt")$changepoints$cp
```

---

signal_teeth *Teeth test signal*

Description

A piecewise-constant signal with regularly spaced changepoints.

Usage

```
signal_teeth(n = 2000, seed = NULL)
```

Arguments

n	Length of the signal. Defaults to 2000.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A tibble with columns index and value.

See Also

Other test signals: cpt_simulate(), signal_blocks(), signal_fms(), signal_mix(), signal_stairs()

Examples

```
x <- signal_teeth(n = 600, seed = 2026)
attr(x, "true_changepoints")   # a change every 100 observations
cpt_detect(x$value, method = "pelt")$changepoints$cp
```

`smuce_wrapper` *SMUCE / HSMUCE wrapper: multiscale changepoint inference*

Description

Wraps `stepR::stepFit()` for the Simultaneous MUltiscale Changepoint Estimator (SMUCE) of Frick, Munk and Sieling (2014) and its heterogeneous extension HSMUCE (Pein, Sieling and Munk, 2017). SMUCE estimates a step function subject to a simultaneous multiscale test at level `alpha`; the level bounds the probability of over-estimating the number of changepoints, and the fit delivers *confidence intervals for every changepoint location*, which populate the `ci_lower`/`ci_upper` columns of the result and render via `autoplot(show_ci = TRUE)` or `geom_cpt_ci()`.

Usage

```
smuce_wrapper(x, alpha = 0.5, family = c("gauss", "hsmuce"), ...)
```

Arguments

`x`	A numeric vector.
`alpha`	Significance level of the multiscale test in $(0, 1)$; smaller values yield more conservative (fewer-changepoint) fits. Defaults to `0.5`, the upstream recommendation for estimation.
`family`	Noise model: `"gauss"` (SMUCE, homogeneous Gaussian noise) or `"hsmuce"` (HSMUCE, segment-wise variance). Defaults to `"gauss"`. The remaining `stepR` families (`"jsmurf"`, `"mDependentPS"`, ...) all require a filter or covariance specification; call `stepR::stepFit()` directly for those. `"hsmuce"` additionally refuses a series whose point-to-point variation lies more than about seven orders of magnitude below its own scale: a globally flat series, or a step whose segments are numerically constant, as `cpt_simulate(sd = 0)` produces once any rounding is added. **stepR**'s heterogeneous variance estimator aborts the R session on such input rather than raising an error, so it cannot be caught. `"gauss"` handles the whole range.
`...`	Additional arguments passed to `stepR::stepFit()`.

Value

A `ggcpt` object. The `changepoints` tibble carries `ci_lower`/`ci_upper` (confidence interval for each changepoint location) and the `data` tibble carries the SMUCE step fit in its `fitted` column.

References

Frick K, Munk A, Sieling H (2014). "Multiscale change point inference." *Journal of the Royal Statistical Society: Series B*, **76**(3), 495–580.

Pein F, Sieling H, Munk A (2017). "Heterogeneous change point inference." *Journal of the Royal Statistical Society: Series B*, **79**(4), 1207–1227.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
x <- c(rnorm(100), rnorm(100, 3))
res <- smuce_wrapper(x)
res$changepoints
ggplot2::autoplot(res, show_ci = TRUE)
```

`sn_wrapper` *Self-normalisation wrapper (SNSeg)*

Description

Wraps `SNSeg::SNSeg_Uni()` (Zhao, Jiang and Shao, 2022): self-normalised segmentation with nested local windows. Self-normalisation avoids estimating the long-run variance, is robust to temporal dependence, and detects changes in general parameters (mean, variance, quantiles, autocorrelation, or bivariate correlation) within one framework.

Usage

```
sn_wrapper(
  x,
  parameter = c("mean", "variance", "acf", "bivcor"),
  confidence = 0.9,
  grid_size = NULL,
  ...
)
```

Arguments

x	A numeric vector (or a two-column matrix for parameter = "bivcor").
parameter	Which parameter to test for changes: "mean", "variance", "acf", or "bivcor" (bivariate correlation). Defaults to "mean".
confidence	Confidence level of the self-normalised test, one of 0.9, 0.95, 0.99, 0.995 or 0.999. Defaults to 0.9.
grid_size	Grid size controlling the local-window sweep; when NULL the engine's default is used.
...	Additional arguments passed to SNSeg::SNSeg_Uni().

Value

A ggcpt object. About 20 observations are needed for the nested local windows at the default grid_size; a constant series returns an empty result rather than an engine error.

References

Zhao Z, Jiang F, Shao X (2022). "Segmenting time series via self-normalisation." *Journal of the Royal Statistical Society: Series B*, **84**(5), 1699–1725.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
res <- sn_wrapper(c(rnorm(150), rnorm(150, 3)))
res$changepoints
```

`stat_changepoint` *Changepoint detection stat*

Description

Runs changepoint detection inside the ggplot pipeline. Useful for quick exploration: `ggplot(df, aes(t, y)) + geom_line() + stat_changepoint(method = "pelt")`. Draws vertical lines at detected changepoint locations.

Usage

```
stat_changepoint(
  mapping = NULL,
  data = NULL,
  geom = "vline",
  position = "identity",
  ...,
  method = "pelt",
  change_in = "mean",
  na.rm = FALSE,
  show.legend = NA
)
```

Arguments

`mapping`	Aesthetic mappings.
`data`	A data frame.
`geom`	The geometric object to use (default: `"vline"`). The stat computes a single `xintercept` per changepoint and drops `x`/`y`, so `"vline"` is the geom that fits. A geom needing `x`/`y` (`"point"`, and `"rug"`, which consumes `x`/`y` rather than `xintercept`) errors for that reason. `inherit.aes` is fixed at `TRUE` here: the stat re-detects on the plot's own data, so the panel's `x`/`y` mapping is what it reads.
`position`	Position adjustment.
`...`	Other arguments passed to the geom.
`method`	Detection method (passed to `cpt_detect`).
`change_in`	What to detect change in (passed to `cpt_detect`).
`na.rm`	If `FALSE`, missing values are removed.
`show.legend`	Whether to show legend.

Value

A ggplot layer.

See Also

Other ggplot2 layers: geom_changepoint(), geom_cpt_ci(), geom_cpt_event(), geom_cpt_label(), geom_cpt_region(), geom_cpt_segment()

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:100, y = c(rnorm(50), rnorm(50, 4)))
ggplot(d, aes(t, y)) + geom_line() +
  stat_changepoint(method = "pelt", colour = "blue")
```

---

strucchange_wrapper *Bai-Perron structural break wrapper (strucchange)*

Description

Wraps strucchange::breakpoints() (Zeileis et al., 2002), the dynamic-programming implementation of the Bai and Perron (1998, 2003) multiple structural break estimator. Called with a bare numeric vector it dates mean shifts (y ~ 1); called with a formula and data it dates breaks in arbitrary regression coefficients. Break-date confidence intervals from confint() populate ci_lower/ci_upper and render via autoplot(show_ci = TRUE).

Usage

```
strucchange_wrapper(
  x,
  data = NULL,
  breaks = NULL,
  h = 0.15,
  conf_level = 0.95,
  ...
)
```

Arguments

x	A numeric vector (mean-shift mode), or a model formula (regression mode; supply data too).
data	Optional data frame for formula input.
breaks	Maximum number of breaks; when NULL the number is chosen by BIC.
h	Minimal segment size, as a fraction of the sample size (or an integer count). Defaults to 0.15.
conf_level	Confidence level for the break-date intervals. Defaults to 0.95.
...	Additional arguments passed to strucchange::breakpoints().

Value

A ggcpt object with ci_lower/ci_upper columns on the changepoints tibble.

Result size

$fit is the breakpoints object itself, and that object is quadratic in the series length: it keeps RSS.triang, the triangular table of segment residual sums of squares, which is what lets strucchange return the optimal segmentation for *any* number of breaks without refitting. Measured here, the whole result is about 1.7 MB at n = 200, 5.9 MB at n = 400 and 22.6 MB at n = 800, roughly four times larger each time the series doubles, and that one table outweighs everything else in the fit put together, by a margin that widens as the series grows. A single fit is not a problem; a few hundred of them are, so when running this engine over a panel with cpt_batch() keep what you need (res$changepoints) rather than the whole list of results. No other engine here behaves this way: the median result across the others is under ten times the size of the series it was given.

References

Bai J, Perron P (2003). "Computation and analysis of multiple structural change models." *Journal of Applied Econometrics*, **18**(1), 1–22.

Zeileis A, Leisch F, Hornik K, Kleiber C (2002). "strucchange: An R package for testing for structural change in linear regression models." *Journal of Statistical Software*, **7**(2), 1–38.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
res <- strucchange_wrapper(c(rnorm(100), rnorm(100, 3)))
res$changepoints
```

---

summary.ggcpt *Summary of a ggcpt object*

Description

Provides a human-readable digest of a changepoint detection result, including the segment table with levels and lengths (the level is the segment mean whatever 'change_in' says; see new_ggcpt), total cost, penalty, and runtime.

Usage

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

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

Arguments

object	A ggcpt object.
...	Additional arguments (ignored).
x	A summary.ggcpt object (for print()).

Value

A list with class summary.ggcpt containing the summary.

---

taylor_wrapper *Taylor's change point analyzer*

Description

Wraps ChangePointTaylor::change_point_analyzer(): the bootstrap-and-recursion procedure of Wayne Taylor that the quality-control and Six Sigma community uses as its default. Each candidate is scored by the bootstrap probability that a change occurred there, which gives a confidence level per changepoint and a confidence interval for its location, both carried onto the result.

Usage

```
taylor_wrapper(
  x,
  n_bootstraps = 1000,
  min_candidate_conf = 0.5,
  min_conf = 0.9,
  conf_level = 0.95,
  seed = NULL
)
```

Arguments

x	A numeric vector.
n_bootstraps	Bootstrap samples per candidate. Defaults to 1000; the engine accepts 100 to 1,000,000.
min_candidate_conf	Minimum confidence for a candidate to be considered, between 0.3 and 1. Defaults to 0.5.
min_conf	Minimum confidence for a changepoint to be reported, between 0.5 and 1. Defaults to 0.9.
conf_level	Confidence level of the reported location intervals, between 0.9 and 0.999 (the engine's range). Defaults to 0.95.
seed	Optional seed (the procedure is bootstrap-based). The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.

Value

A ggcpt object with ci_lower/ci_upper (so autoplot(show_ci = TRUE) works) and a confidence column.

Series length, and why you cannot interrupt it

This engine is written for the series lengths quality control sees (hundreds to low thousands), and it does not scale. At $n = 10{,}000$ with the default n_bootstraps = 1000 it runs for **minutes**, and more importantly it runs where R cannot look: a setTimeLimit() of 45 seconds was still not honoured after 170, and one of 125 seconds after 200, so the call had to be killed from outside the session. R checks elapsed-time limits and keyboard interrupts at the same points, which means **Ctrl-C will not stop it either**.

So size the call before starting it rather than after. n_bootstraps is the knob (the cost is roughly linear in it), and the "Benchmarks" article lists the methods that do scale to long series. That page is web-only, because the sweep behind it takes over twenty minutes: it is published at https://pursuitofdatascience.github.io/ggchangepoint/articles/benchmarks.html rather than built into the package, so vignette() will not find it.

References

Taylor WA (2000). *Change-Point Analysis: A Powerful New Tool for Detecting Changes*. Taylor Enterprises, Libertyville, Illinois.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(),

ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
taylor_wrapper(c(rnorm(60), rnorm(60, 3)), n_bootstraps = 200, seed = 1)
```

---

tguh_wrapper *TGUH wrapper*

Description

Wraps the `breakfast` package for Tail-Greedy Unbalanced-Haar detection, with information-criterion model selection.

Usage

```
tguh_wrapper(x, ...)
```

Arguments

`x`	A numeric vector.
`...`	Additional arguments passed to `breakfast::breakfast()`.

Value

A `ggcpt` object.

References

Fryzlewicz P (2018). "Tail-Greedy Bottom-Up Data Decompositions and Fast Multiple Change-Point Detection." *The Annals of Statistics*, **46**(6B), 3390–3421.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
res <- tguh_wrapper(c(rnorm(60), rnorm(60, 4)))
res$changepoints
res$penalty$type
```

`theme_ggcpt` *ggchangepoint theme*

Description

A minimal, publication-ready ggplot2 theme for changepoint plots.

Usage

```
theme_ggcpt(base_size = 11, base_family = "")
```

Arguments

`base_size`	Base font size. Defaults to 11.
`base_family`	Base font family. Defaults to "".

Value

A ggplot2 theme object.

See Also

Other accessibility scales: `scale_colour_cpt()`, `scale_fill_cpt_label()`

Examples

```
library(ggplot2)
ggplot(mtcars, aes(wt, mpg)) + geom_point() + theme_ggcpt()
```

tidy.ggcpt *Tidy a ggcpt object*

Description

Returns the changepoints tibble (one row per changepoint).

Usage

```
## S3 method for class 'ggcpt'
tidy(x, ...)
```

Arguments

x	A ggcpt object.
...	Additional arguments (ignored).

Value

A tibble with columns cp, cp_value, and any method-specific columns.

trend_wrapper *Classical single-changepoint tests (Pettitt, Buishand, SNHT)*

Description

Wraps the three single-change tests that hydrology and climatology use as their standard vocabulary, from the **trend** package. All three test H_0: no change against a single change in the mean, differing in how they measure it:

"pettitt" a rank-based (Mann-Whitney) statistic, distribution-free and robust to outliers.

"buishand" the Buishand range test, based on rescaled adjusted partial sums; assumes normality.

"snht" the standard normal homogeneity test of Alexandersson, the reference method for detecting inhomogeneities in climate records.

Each reports a location *and* a p-value, and, unlike most engines here, that p-value is valid, because the location was not chosen from a larger model search.

Usage

```
trend_wrapper(x, test = c("pettitt", "buishand", "snht"), alpha = 0.05, ...)
```

Arguments

x	A numeric vector.
test	Which test to run. Defaults to "pettitt".
alpha	Significance level below which the changepoint is reported. Defaults to 0.05. A test that does not reject returns an empty result rather than an unsupported location.
...	Additional arguments passed to the **trend** function.

Value

A ggcpt object with the test's p_value and statistic on the changepoints tibble. The per-position test statistic is available through cpt_statistic().

References

Pettitt AN (1979). "A non-parametric approach to the change-point problem." *Journal of the Royal Statistical Society: Series C*, **28**(2), 126–135. doi:10.2307/2346729.

Buishand TA (1982). "Some methods for testing the homogeneity of rainfall records." *Journal of Hydrology*, **58**(1–2), 11–27. doi:10.1016/00221694(82)90066X.

Alexandersson H (1986). "A homogeneity test applied to precipitation data." *Journal of Climatology*, **6**(6), 661–675. doi:10.1002/joc.3370060607.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
x <- c(rnorm(60), rnorm(60, 2))
trend_wrapper(x, test = "pettitt")
trend_wrapper(x, test = "snht")
```

`var_wrapper` *VAR(1) changepoints*

Description

Wraps `changepoints::CV.search.DP.VAR1()` (Wang, Yu, Rinaldo and Willett): dynamic programming with an ℓ_0 penalty for changes in the transition matrix of a vector autoregression, with the two tuning parameters chosen by cross-validation. The change here is in the *dynamics* (how the series predicts itself), not in the level, so it is invisible to every mean-change engine in the package.

Usage

```
var_wrapper(x, gamma_set = NULL, lambda_set = NULL, delta = NULL, ...)
```

Arguments

`x`	A numeric matrix or data frame, rows as time points.
`gamma_set`	Candidate values of the ℓ_0 tuning parameter. Defaults to a small grid scaled by the series length.
`lambda_set`	Candidate lasso penalties. Defaults to `c(0.01, 0.1, 1)`.
`delta`	Minimum spacing. Defaults to `max(5, floor(n / 20))`.
`...`	Additional arguments passed to the engine.

Value

A `ggcpt` object with `change_in = "regression"`. The engine searches every other observation, so a location is resolved to within two; each is reported as the last observation before the change, the package convention, rather than the engine's own index, which sits one or two earlier.

References

Wang D, Yu Y, Rinaldo A, Willett R (2019). "Localizing changes in high-dimensional vector autoregressive processes." *arXiv preprint arXiv:1909.06359*. doi:10.48550/arXiv.1909.06359.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `wbs2_wrapper()`, `wbs_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
p <- 3
step <- function(n, a) {
  Y <- matrix(0, n, p)
  for (i in 2:n) Y[i, ] <- a * Y[i - 1, ] + stats::rnorm(p)
  Y
}
var_wrapper(rbind(step(50, 0.1), step(50, 0.8)),
            gamma_set = c(1, 10), lambda_set = c(0.1, 1))
```

`wbs2_wrapper` *WBS2 wrapper: Wild Binary Segmentation 2*

Description

Wraps the `breakfast` package's WBS2 solution path with steepest-drop-to-low-levels (SDLL) model selection.

Usage

```
wbs2_wrapper(x, ...)
```

Arguments

`x`	A numeric vector.
`...`	Additional arguments passed to `breakfast::breakfast()`.

Value

A `ggcpt` object.

Reproducibility

This engine is **not reproducible call to call within an R session**, and no argument here can make it so. On a 200-point series with one change at 100, repeated identical calls (same input, `set.seed()` re-run beforehand so that `.Random.seed` is byte-identical on entry) return a last changepoint of either 183 or 188, roughly evenly split. The variation is therefore not driven by R's random number stream, which is why this wrapper has no `seed` argument to offer: there is no stream to pin.

It is upstream, not in this package. Calling `breakfast::breakfast(x, solution.path = "wbs2", model.selection = "sdll")` directly (**breakfast** 2.5) reproduces it exactly. A *fresh* R session is deterministic (five separate sessions agreed), so what varies is state the engine carries between calls.

In practice this is rare, and needs a series whose model selection sits near a tie. Measured over eight other series with three identical calls each, `wbs2` was stable on all eight; the flip above reproduces

only on that one configuration. tguh, which uses the same package, was stable throughout, as were the other 35 of the 50 wired methods this experiment covered (the rest need an engine that was not installed when it was run). If you need a segmentation you can reproduce exactly, take the result of the first call in a session, or use an engine with a seed argument (cpt_methods() reports which wrappers have one).

References

Fryzlewicz P (2020). "Detecting Multiple Change-Point Features via Narrowest-Over-Threshold." *Journal of the Royal Statistical Society Series B*, **82**(5), 1377–1418.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs_wrapper(), wbsts_wrapper()

Examples

```
set.seed(2026)
res <- wbs2_wrapper(c(rnorm(60), rnorm(60, 4)))
res$changepoints
```

---

wbsts_wrapper *WBS for nonstationary time series*

Description

Wraps wbsts::wbs.lsw() (Korkas and Fryzlewicz): wild binary segmentation applied to the locally stationary wavelet spectrum, so it detects changes in the *second-order* structure (variance and autocovariance) of a nonstationary series. Where wbs looks for jumps in the level, this looks for jumps in how the series behaves.

Usage

```
wbsts_wrapper(
  x,
  n_intervals = 0,
  cstar = 0.75,
  lambda = 0.75,
  scales = NULL,
```

```
  seed = NULL,
  ...
)
```

Arguments

x	A numeric vector.
n_intervals	Number of random intervals (M). Defaults to 0, which is the engine's "all dyadic intervals" setting.
cstar	The unbalancedness parameter $c_\star$ of the search: a candidate split may leave at most this fraction of its interval on either side. Between 0.5 and 1; defaults to the engine's 0.75.
lambda	How many wavelet scales are used when scales is NULL: floor(3 * lambda * log(log(n))) of them, from the finest down. Defaults to the engine's 0.75.
scales	Wavelet scales to use: at least two different whole numbers from 1 to floor(log2(n)) - 1. NULL lets the engine choose.
seed	Optional seed. The seed is scoped to this call: .Random.seed is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to wbsts::wbs.lsw().

Value

A ggcpt object with change_in = "var".

References

Korkas KK, Fryzlewicz P (2017). "Multiple change-point detection for non-stationary time series using wild binary segmentation." *Statistica Sinica*, **27**(1), 287–311. doi:10.5705/ss.202015.0262.

See Also

Other changepoint engines: bcp_wrapper(), beast_wrapper(), bfast_wrapper(), binsegrcpp_wrapper(), bocpd_wrapper(), cpm_wrapper(), cpop_wrapper(), cpt_wrapper(), decafs_wrapper(), ecp_wrapper(), envcpt_wrapper(), esac_wrapper(), fabisearch_wrapper(), fastcpd_wrapper(), fcov_wrapper(), fmean_wrapper(), fpop_wrapper(), geomcp_wrapper(), hdcov_wrapper(), hdreg_wrapper(), idetect_wrapper(), inspect_wrapper(), kcp_wrapper(), kwc_wrapper(), mcp_wrapper(), mosum_wrapper(), network_wrapper(), not_wrapper(), npmojo_wrapper(), nsp_wrapper(), ocd_wrapper(), pilliat_wrapper(), segmented_wrapper(), smuce_wrapper(), sn_wrapper(), strucchange_wrapper(), taylor_wrapper(), tguh_wrapper(), trend_wrapper(), var_wrapper(), wbs2_wrapper(), wbs_wrapper()

Examples

```
set.seed(2026)
y <- c(as.numeric(stats::arima.sim(list(ar = 0.1), 250)),
       as.numeric(stats::arima.sim(list(ar = 0.9), 250)))
```

```
wbsts_wrapper(y)
```

`wbs_wrapper` *WBS wrapper: Wild Binary Segmentation*

Description

Wraps the `wbs` package for randomised changepoint detection via Wild Binary Segmentation.

Usage

```
wbs_wrapper(x, n_intervals = 5000, threshold = NULL, seed = NULL, ...)
```

Arguments

`x`	A numeric vector.
`n_intervals`	Number of random intervals. Defaults to `5000`.
`threshold`	Manual threshold for detection. If `NULL`, model selection uses the strengthened Schwarz Information Criterion (sSIC).
`seed`	Optional seed for reproducibility. The seed is scoped to this call: `.Random.seed` is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
`...`	Additional arguments passed to `wbs::wbs()`.

Value

A `ggcpt` object.

References

Fryzlewicz P (2014). "Wild Binary Segmentation for Multiple Change-Point Detection." *The Annals of Statistics*, **42**(6), 2243–2281. doi:10.1214/14AOS1245.

See Also

Other changepoint engines: `bcp_wrapper()`, `beast_wrapper()`, `bfast_wrapper()`, `binsegrcpp_wrapper()`, `bocpd_wrapper()`, `cpm_wrapper()`, `cpop_wrapper()`, `cpt_wrapper()`, `decafs_wrapper()`, `ecp_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetect_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmojo_wrapper()`, `nsp_wrapper()`, `ocd_wrapper()`, `pilliat_wrapper()`, `segmented_wrapper()`, `smuce_wrapper()`, `sn_wrapper()`, `strucchange_wrapper()`, `taylor_wrapper()`, `tguh_wrapper()`, `trend_wrapper()`, `var_wrapper()`, `wbs2_wrapper()`, `wbsts_wrapper()`

Examples

```
set.seed(2026)
res <- wbs_wrapper(c(rnorm(60), rnorm(60, 4)), n_intervals = 200)
res$changepoints
res$penalty$type   # sSIC selection, because `threshold` was not supplied
```

Index

∗ **accessibility scales**
scale_colour_cpt, 143
scale_fill_cpt_label, 144
theme_ggcpt, 159
∗ **changepoint engines**
bcp_wrapper, 13
beast_wrapper, 14
bfast_wrapper, 15
binsegrcpp_wrapper, 17
bocpd_wrapper, 18
cpm_wrapper, 19
cpop_wrapper, 20
cpt_wrapper, 83
decafs_wrapper, 85
ecp_wrapper, 86
envcpt_wrapper, 87
esac_wrapper, 88
fabisearch_wrapper, 90
fastcpd_wrapper, 92
fcov_wrapper, 93
fmean_wrapper, 96
fpop_wrapper, 97
geomcp_wrapper, 98
hdcov_wrapper, 118
hdreg_wrapper, 120
idetect_wrapper, 121
inspect_wrapper, 122
kcp_wrapper, 124
kwc_wrapper, 126
mcp_wrapper, 127
mosum_wrapper, 129
network_wrapper, 130
not_wrapper, 134
npmojo_wrapper, 135
nsp_wrapper, 137
ocd_wrapper, 139
pilliat_wrapper, 141
segmented_wrapper, 145
smuce_wrapper, 150
sn_wrapper, 151
strucchange_wrapper, 154
taylor_wrapper, 156
tguh_wrapper, 158
trend_wrapper, 160
var_wrapper, 162
wbs2_wrapper, 163
wbs_wrapper, 166
wbsts_wrapper, 164
∗ **ggplot2 layers**
geom_changepoint, 100
geom_cpt_ci, 101
geom_cpt_event, 102
geom_cpt_label, 103
geom_cpt_region, 104
geom_cpt_segment, 106
stat_changepoint, 153
∗ **plotting**
ggcpt_compare_table, 109
ggcpt_eval, 110
ggcpt_interactive, 111
ggcptplot, 107
ggecpplot, 116
∗ **result class**
annotate_segments, 6
as_cpt_series, 7
as_ggcpt, 8
cpt_annotations, 23
is_ggcpt, 124
new_ggcpt, 132
print.ggcpt, 142
∗ **test signals**
cpt_simulate, 76
signal_blocks, 146
signal_fms, 147
signal_mix, 148
signal_stairs, 148
signal_teeth, 149

alarms, 5, *57*, *59*, *83*

annotate_segments, 6, *8*, *10*, *24*, *124*, *134*, *143*
as.data.frame, *112*
as.data.frame.ggcpt *(*ggcpt_methods*)*, 112
as_cpt_labels, 7, *44*
as_cpt_series, *6*, 7, *10*, *24*, *124*, *134*, *143*
as_ggcpt, *6*, *8*, 8, *24*, *66, 67*, *124*, *132–134*, *143*
as_tibble.ggcpt *(*ggcpt_methods*)*, 112
augment.ggcpt, 10, *38, 39*
autoplot.ggcpt, 11, *32*, *39*, *109*, *112*, *114*
autoplot.ggcpt_batch *(*cpt_batch*)*, 24
autoplot.ggcpt_benchmark *(*cpt_benchmark*)*, 26
autoplot.ggcpt_consensus *(*cpt_consensus*)*, 31
autoplot.ggcpt_delay *(*cpt_delay*)*, 35
autoplot.ggcpt_events *(*cpt_annotate_events*)*, 22
autoplot.ggcpt_influence *(*cpt_influence*)*, 40
autoplot.ggcpt_label_curve *(*cpt_label_error_curve*)*, 45
autoplot.ggcpt_monitor *(*cpt_monitor*)*, 57
autoplot.ggcpt_path *(*cpt_crops*)*, 33
autoplot.ggcpt_power *(*cpt_power*)*, 61
autoplot.ggcpt_selection *(*cpt_select*)*, 72
autoplot.ggcpt_sensitivity *(*cpt_sensitivity*)*, 75
autoplot.ggcpt_stability *(*cpt_stability*)*, 79

bcp_wrapper, 13, *15, 16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *115*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
beast_wrapper, *14*, 14, *16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *115*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
bfast_wrapper, *8*, *14, 15*, 15, *18–21*, *77*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
binsegrcpp_wrapper, *14–16*, 17, *19–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
bocpd_wrapper, *14–16*, *18*, 18, *20, 21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *115*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

coef.ggcpt_penalty_model *(*cpt_learn_penalty*)*, 47
cpm_wrapper, *14–16*, *18, 19*, 19, *21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
cpop_wrapper, *14–16*, *18–20*, 20, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
cpt_annotate_events, 22, *69*, *102, 103*
cpt_annotations, *6*, *8*, *10*, 23, *124*, *134*, *143*
cpt_batch, *16*, 24, *63*, *108–110*, *133*, *142*, *155*
cpt_benchmark, *16*, 26, *34, 35*, *50*, *53, 54*, *63*, *71, 72*, *142*
cpt_cite, 28, *65*, *67*, *69*
cpt_confint, 29, *32*, *65*, *69*, *81, 82*, *133*, *138*
cpt_consensus, *8*, *22*, 31, *63*, *65*, *138*
cpt_crops, 33, *75*, *79*
cpt_datasets, *24*, *26–28*, 34
cpt_delay, *5*, 35, *57–59*, *68*
cpt_detect, *7, 8*, *10*, *12*, *16, 17*, *21*, *25*, *30*, 37, *41*, *46–48*, *52*, *60*, *63, 64*, *66*, *74, 75*, *80*, *84, 85*, *93*, *98*, *109*, *120*, *134*
cpt_gt, 39, *69*, *133*
cpt_influence, 40, *49*, *63*, *75, 76*
cpt_install_engines, 42, *52*
cpt_label_error, *12*, *44*, 44, *103, 104*
cpt_label_error_curve, *45*, 45, *47, 48*
cpt_labels, *12*, 43, *45*, *104*

cpt_learn_penalty, *44*, *46*, 47, *60*
cpt_leverage, *42*, 49
cpt_load_tcpd, *28*, *35*, 50
cpt_methods, *10*, *12*, *29*, *37*, *39*, *43*, 51, *64, 65*, *139*, *164*
cpt_metrics, *7, 8*, *22*, *26*, *28*, *31, 32*, *35*, 52, *54, 55*, *110*, *138*
cpt_metrics_annotated, *24*, *26*, *50*, *54*, 54
cpt_min_detectable, 55, *63*
cpt_monitor, *5*, *36*, *52*, 57, *68*, *82, 83*
cpt_penalty, *38, 39*, *48*, 60, *73*, *75*, *93*
cpt_power, *55, 56*, 61, *72*
cpt_recommend, *32*, 64
cpt_regions, 65, *105*, *138*
cpt_register_method, *10*, *28*, *37*, *51, 52*, 66, *82*
cpt_registered_methods, *67*
cpt_registered_methods *(*cpt_register_method*)*, 66
cpt_replay, *36*, *59*, 68
cpt_report, *8*, *23*, *38*, *40*, 69
cpt_scale_space, *52*, 70, *81*
cpt_scenarios, *56*, *62, 63*, 71
cpt_select, *46*, *61*, 72, *76*
cpt_sensitivity, *42*, *63*, *75*, 75
cpt_simulate, *56*, *62, 63*, *71*, 76, *78*, *147–149*
cpt_solution_path, *17*, *66*, 78, *81*
cpt_stability, *30*, *42*, *69*, 79, *133*
cpt_statistic, *66*, *70*, *79*, 80, *161*
cpt_test, *30*, 81
cpt_unregister_method, *67*
cpt_unregister_method *(*cpt_register_method*)*, 66
cpt_update, *57–59*, 82
cpt_wrapper, *14–16*, *18–21*, 83, *85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

decafs_wrapper, *14–16*, *18–21*, *84*, 85, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

ecp_wrapper, *14–16*, *18–21*, *84, 85*, 86, *88, 89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
envcpt_wrapper, *14–16*, *18–21*, *84, 85*, *87*, 87, *89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
esac_wrapper, *14–16*, *18–21*, *84, 85*, *87, 88*, 88, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140–142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

fabisearch_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, 90, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
fastcpd_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, 92, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
fcov_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, 93, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
fmean_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93–95*, 96, *98, 99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
format.ggcpt *(*ggcpt_methods*)*, 112
fpop_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97*, 97, *99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

geom_changepoint, *13*, 100, *101*, *103–106*, *154*
geom_cpt_ci, *100*, 101, *103–106*, *150*, *154*
geom_cpt_event, *23*, *100, 101*, 102, *104–106*, *154*

geom_cpt_label, *44, 45*, *100, 101*, *103*, 103, *105, 106*, *154*
geom_cpt_region, *65*, *100, 101*, *103, 104*, 104, *106*, *134*, *138*, *154*
geom_cpt_segment, *100, 101*, *103–105*, 106, *154*
geomcp_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97, 98*, 98, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
ggcpt_compare, *16*, *32*, *63*, 108
ggcpt_compare_table, *108*, 109, *111*, *117*
ggcpt_eval, *108*, *110*, 110, *111*, *117*
ggcpt_interactive, *108*, *110, 111*, 111, *117*
ggcpt_methods, 112, *114*
ggcpt_plot_methods, *13*, *112*, 113
ggcpt_posterior, *13, 14*, 115, *127*
ggcpt_runlength, *18*, 115
ggcpt_scale_space, *12*, *134*
ggcpt_scale_space *(*cpt_scale_space*)*, 70
ggcpt_solution_path, *12*, *52*, *134*
ggcpt_solution_path *(*cpt_solution_path*)*, 78
ggcpt_statistic, *12*, *52*, *134*
ggcpt_statistic *(*cpt_statistic*)*, 80
ggcptplot, 107, *110, 111*, *117*
ggecpplot, *108*, *110, 111*, 116
glance.ggcpt, *38, 39*, 117
glance.ggcpt_delay *(*cpt_delay*)*, 35

hdcov_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, 118, *121–123*, *126, 127*, *129–132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
hdreg_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, 120, *122, 123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

idetect_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121*, 121, *123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
inspect_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121, 122*, 122, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
install.packages, *43*
is_ggcpt, *6*, *8*, *10*, *24*, 124, *134*, *143*

kcp_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, 124, *127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
kwc_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126*, 126, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

mcp_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, 127, *130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
mosum_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129*, 129, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

network_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, 130, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
new_ggcpt, *6*, *8*, *10*, *24*, *38*, *124*, 132, *143*, *156*
not_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, 134, *136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*
npmojo_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135*, 135, *138*, *140*, *142*, *146*, *151, 152*, *155*, *157, 158*, *161, 162*, *164–166*

nsp_wrapper, *9*, *12*, *14–16*, *18–21*, *29*, *30*, *32*, *65*, *82*, *84*, *85*, *87–89*, *91*, *93*, *95*, *97–99*, *104*, *105*, *119*, *121–123*, *126*, *127*, *129*, *130*, *132*, *134–136*, 137, *140*, *142*, *146*, *151*, *152*, *155*, *157*, *158*, *161*, *162*, *164–166*

ocd_wrapper, *14–16*, *18–21*, *84*, *85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126*, *127*, *129*, *130*, *132*, *135*, *136*, *138*, 139, *142*, *146*, *151*, *152*, *155*, *158*, *161*, *162*, *164–166*

p.adjust, *81*
pilliat_wrapper, *14–16*, *18–21*, *84*, *85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126*, *127*, *129*, *130*, *132*, *135*, *136*, *138*, *140*, 141, *146*, *151*, *152*, *155*, *158*, *161*, *162*, *164–166*
plot.default, *113*
plot.ggcpt *(*ggcpt_methods*)*, 112
plot.ggcpt_batch *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_benchmark *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_consensus *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_delay *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_events *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_influence *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_label_curve *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_monitor *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_path *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_power *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_selection *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_sensitivity *(*ggcpt_plot_methods*)*, 113
plot.ggcpt_stability *(*ggcpt_plot_methods*)*, 113
predict.ggcpt_penalty_model *(*cpt_learn_penalty*)*, 47
print.cpt_label_error *(*cpt_label_error*)*, 44
print.ggcpt, *6*, *8*, *10*, *24*, *39*, *124*, *134*, 142
print.ggcpt_batch *(*cpt_batch*)*, 24
print.ggcpt_benchmark *(*cpt_benchmark*)*, 26
print.ggcpt_consensus *(*cpt_consensus*)*, 31
print.ggcpt_delay *(*cpt_delay*)*, 35
print.ggcpt_events *(*cpt_annotate_events*)*, 22
print.ggcpt_influence *(*cpt_influence*)*, 40
print.ggcpt_label_curve *(*cpt_label_error_curve*)*, 45
print.ggcpt_min_detectable *(*cpt_min_detectable*)*, 55
print.ggcpt_monitor *(*cpt_monitor*)*, 57
print.ggcpt_path *(*cpt_crops*)*, 33
print.ggcpt_penalty_model *(*cpt_learn_penalty*)*, 47
print.ggcpt_power *(*cpt_power*)*, 61
print.ggcpt_recommendation *(*cpt_recommend*)*, 64
print.ggcpt_selection *(*cpt_select*)*, 72
print.ggcpt_sensitivity *(*cpt_sensitivity*)*, 75
print.ggcpt_stability *(*cpt_stability*)*, 79
print.summary.ggcpt *(*summary.ggcpt*)*, 155

rcpt *(*cpt_simulate*)*, 76

scale_color_cpt *(*scale_colour_cpt*)*, 143
scale_colour_cpt, 143, *144*, *159*
scale_colour_cpt_label *(*scale_fill_cpt_label*)*, 144
scale_fill_cpt *(*scale_colour_cpt*)*, 143
scale_fill_cpt_label, *104*, *144*, 144, *159*
scale_linetype_cpt *(*scale_colour_cpt*)*, 143
segmented_wrapper, *14–16*, *18–21*, *84*, *85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126*, *127*, *129*, *130*, *132*, *135*, *136*, *138*, *140*, *142*, 145, *151*, *152*, *155*, *158*, *161*, *162*, *164–166*
signal_blocks, *78*, 146, *147–149*
signal_fms, *78*, *147*, 147, *148*, *149*
signal_mix, *78*, *147*, 148, *149*

signal_stairs, *78*, *147, 148*, 148, *149*
signal_teeth, *78*, *147–149*, 149
smuce_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, 150, *152*, *155*, *158*, *161, 162*, *164–166*
sn_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151*, 151, *155*, *158*, *161, 162*, *164–166*
stat_changepoint, *100, 101*, *103–106*, 153
strucchange_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, 154, *158*, *161, 162*, *164–166*
summary, *142*
summary.ggcpt, *39*, *143*, 155

taylor_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, 156, *158*, *161, 162*, *164–166*
tguh_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, 158, *161, 162*, *164–166*
theme_ggcpt, *13*, *144*, 159
tidy, *59*, *142*
tidy.cpt_label_error *(*cpt_label_error*)*, 44
tidy.cpt_labels *(*cpt_labels*)*, 43
tidy.ggcpt, *38, 39*, *112*, *143*, 160
tidy.ggcpt_batch *(*cpt_batch*)*, 24
tidy.ggcpt_benchmark *(*cpt_benchmark*)*, 26
tidy.ggcpt_delay *(*cpt_delay*)*, 35
tidy.ggcpt_events *(*cpt_annotate_events*)*, 22
tidy.ggcpt_influence *(*cpt_influence*)*, 40
tidy.ggcpt_monitor *(*cpt_monitor*)*, 57
tidy.ggcpt_path *(*cpt_crops*)*, 33
tidy.ggcpt_power *(*cpt_power*)*, 61
tidy.ggcpt_recommendation *(*cpt_recommend*)*, 64
tidy.ggcpt_selection *(*cpt_select*)*, 72
tidy.ggcpt_sensitivity *(*cpt_sensitivity*)*, 75
trend_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, 160, *162*, *164–166*

var_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, *161*, 162, *164–166*

wbs2_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, *161, 162*, 163, *165, 166*
wbs_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, *161, 162*, *164, 165*, 166
wbsts_wrapper, *14–16*, *18–21*, *84, 85*, *87–89*, *91*, *93*, *95*, *97–99*, *119*, *121–123*, *126, 127*, *129, 130*, *132*, *135, 136*, *138*, *140*, *142*, *146*, *151, 152*, *155*, *158*, *161, 162*, *164*, 164, *166*