

Package: ggchangept (via r-universe)

October 12, 2026

Type Package

Title Combines Changept Analysis with 'ggplot2'

Version 0.6.0

Description A unified, tidy, 'ggplot2'-native interface to changepoint detection in R. A 'cpt_detect()' dispatcher covers fifty methods (penalised and optimal partitioning, multiscale and search methods, nonparametric and kernel methods, Bayesian methods, high-dimensional, functional, covariance and network methods, regression breaks through a formula interface, seasonal-trend decomposition and the classical single-change tests), with distribution families for counts, binary outcomes and waiting times, missing-value handling, grouped data, and constraints on locations and segment lengths. Every result is the same 'ggcpt' object, in one verified location convention, with 'broom'-style tidy/glance/augment methods, 'autoplot()' and composable 'ggproto' layers. Around detection: effect sizes, a test for a change at a date fixed in advance, tests for a change anywhere, confidence intervals, post-detection tests, selection of the number of changes, assumption checks and goodness of fit, robustness to the noise model, stability, consensus and influence diagnostics, supervised detection, sequential monitoring, power and study design, benchmarks, reports and an export under a documented 'JSON' schema. What each engine does (its runtime, whether its answer survives a change of units or a reversal of time, its false positives by noise regime and data type, its false alarms on pure noise) and the calibration of the package's own intervals and tests are measured and shipped as data, and 'cpt_recommend()' chooses from those measurements. 'cpt_register_method()' brings external and non-CRAN detectors into the same grammar.

Depends R (>= 4.0.0)

License GPL (>= 3)

Encoding UTF-8

LazyData true

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, vdiff, 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, KWCCChangepoint, 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>,
<https://pursuitofdatascience.github.io/ggchangepoint/repo>

Config/testthat/edition 3

VignetteBuilder knitr

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

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

Config/Needs/website rmarkdown

Repository <https://pursuitofdatascience.r-universe.dev>

Date/Publication 2026-10-12 16:36:28 UTC

RemoteUrl <https://github.com/pursuitofdatascience/ggchangepoint>

RemoteRef HEAD

RemoteSha a0614017653268154f127d4a16f9547e5f1a45ea

Contents

alarms	5
annotate_segments	6
as_cpt_labels	7
as_cpt_series	8
as_ggcpt	9
as_json	10
augment.ggcpt	12
autoplot.ggcpt	13
bcp_wrapper	15
beast_wrapper	17
bfast_wrapper	18
binsegrcpp_wrapper	19
bocpd_wrapper	21

cpm_wrapper	22
cpop_wrapper	23
cpt_annotate_events	24
cpt_annotations	26
cpt_assumptions	27
cpt_attribute_event	28
cpt_batch	29
cpt_benchmark	30
cpt_calibration	33
cpt_cite	34
cpt_confint	35
cpt_consensus	37
cpt_crops	39
cpt_data_types	41
cpt_datasets	42
cpt_delay	43
cpt_detect	44
cpt_effect	49
cpt_export	50
cpt_families	51
cpt_gof	53
cpt_gt	54
cpt_influence	54
cpt_install_engines	56
cpt_invariances	58
cpt_label_error	59
cpt_label_error_curve	60
cpt_labels	61
cpt_learn_penalty	62
cpt_leverage	64
cpt_load_tcpd	65
cpt_methods	66
cpt_metrics	68
cpt_metrics_annotated	70
cpt_min_detectable	71
cpt_monitor	73
cpt_noise_benchmark	76
cpt_null_power	77
cpt_null_sizes	78
cpt_penalty	79
cpt_power	80
cpt_recommend	83
cpt_regions	85
cpt_register_method	86
cpt_replay	88
cpt_report	89
cpt_robustness	90
cpt_runtimes	91

cpt_scale_space	92
cpt_scenarios	93
cpt_segment_models	95
cpt_select	96
cpt_sensitivity	99
cpt_simulate	100
cpt_solution_path	103
cpt_stability	104
cpt_statistic	106
cpt_test	106
cpt_test_at	108
cpt_test_null	110
cpt_update	111
cpt_verify	112
cpt_wrapper	113
decafs_wrapper	115
ecp_wrapper	116
envcpt_wrapper	117
esac_wrapper	118
fabisearch_wrapper	120
fastcpd_wrapper	122
fcov_wrapper	124
fmean_wrapper	126
fpop_wrapper	127
geom_changepoint	129
geom_cpt_ci	130
geom_cpt_event	131
geom_cpt_label	132
geom_cpt_region	133
geom_cpt_segment	135
geomcp_wrapper	136
ggchangepoint-conditions	137
ggchangepoint-ggproto	138
ggcpt_compare	139
ggcpt_compare_table	141
ggcpt_eval	142
ggcpt_interactive	143
ggcpt_methods	144
ggcpt_plot_methods	145
ggcpt_posterior	147
ggcpt_runlength	147
ggcptplot	148
ggcplot	149
glance.ggcpt	151
hdcov_wrapper	152
hdreg_wrapper	153
idetect_wrapper	155
inspect_wrapper	156

is_ggcpt	157
kcp_wrapper	158
kwc_wrapper	159
mcp_wrapper	161
mosum_wrapper	163
network_wrapper	164
new_ggcpt	166
not_wrapper	168
npmojo_wrapper	169
nsp_wrapper	170
ocd_wrapper	172
pilliat_wrapper	174
predict.ggcpt	176
print.ggcpt	177
scale_colour_cpt	178
scale_fill_cpt_label	179
segmented_wrapper	180
signal_blocks	182
signal_fms	183
signal_mix	183
signal_stairs	184
signal_teeth	185
smuce_wrapper	186
sn_wrapper	187
stat_changepoint	189
stat_cpt_region	190
strucchange_wrapper	191
summary.ggcpt	193
taylor_wrapper	194
tguh_wrapper	195
theme_ggcpt	196
tidy.ggcpt	197
trend_wrapper	198
var_wrapper	199
wbs_wrapper	200
wbs2_wrapper	201
wbsts_wrapper	203

Index**205**

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	<i>Annotate segments with alternating shading</i>
-------------------	---

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\(\)](#), [as_json\(\)](#), [cpt_annotations\(\)](#), [cpt_export\(\)](#), [cpt_verify\(\)](#), [ggchangept-conditions](#), [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$data, aes(index, value)) +
  annotate_segments(fit$changepts$cp, n = nrow(fit$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 $2 * \text{margin} + 1$, 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

<code>x</code>	A numeric vector or matrix, or a <code>ts/mts</code> , <code>xts</code> , <code>zoo</code> or <code>tsibble</code> object. The <code>xts</code> , <code>zoo</code> and <code>tsibble</code> paths need those packages installed (they are Suggests).
<code>index</code>	Optional explicit index, one value per observation. Overrides any index carried by <code>x</code> , and is the way to attach dates to a plain numeric vector.
<code>check_regular</code>	Warn when the index is not equally spaced? Defaults to <code>TRUE</code> . 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()`, `as_json()`, `cpt_annotations()`, `cpt_export()`, `cpt_verify()`, `ggchangepoint-conditions`, `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 <code>autoplot(show_fit = TRUE)</code> and <code>augment()</code> .
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 <code>cp_convention</code> as <code>cp</code> : with "right" they are converted along with <code>cp</code> , 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\(\)](#), [as_json\(\)](#), [cpt_annotations\(\)](#), [cpt_export\(\)](#), [cpt_verify\(\)](#), [ggchangepoint-conditions](#), [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)
```

as_json

Convert a result to JSON

Description

The whole result as JSON, under a documented and versioned schema, for software that consumes changepoint results rather than a person reading them.

Usage

```
as_json(x, ...)

## S3 method for class 'ggcpt'
as_json(x, pretty = TRUE, data = TRUE, assumptions = TRUE, digits = NA, ...)

## Default S3 method:
as_json(x, ...)
```

Arguments

x	A ggcpt object.
...	Ignored.
pretty	Indent the output? Defaults to TRUE.
data	Include the series itself (index, value and, where the engine supplied one, fitted)? Defaults to TRUE; without it the output is the answer alone, a few hundred bytes.
assumptions	Include <code>cpt_assumptions()</code> ? Defaults to TRUE.
digits	Significant digits for numbers. Defaults to NA, the full precision needed for an exact round trip.

Value

A single string of class "json".

Schema 1.0.0

One object with these fields. Every field is always present; one that does not apply is null, so a reader never has to test for a key.

schema, schema_version "ggchangept.ggcpt" and the version, "1.0.0".

method, engine, engine_version, ggchangept_version, r_version, created what produced the result, and when (ISO 8601).

change_in, family what was detected, and the distribution family if one was asked for.

penalty {"type", "value"}.

cp_convention "left": a changepoint is the last observation of its segment.

n the series length.

index {"class", "label", "tz"} of the time index, or null. Dates are written as ISO 8601 strings and timestamps with their offset; tz is a timestamp index's time zone (null otherwise).

changepoints an array of objects, one per changepoint: cp (position) and cp_value, with cp_index when there is an index, and every engine-specific column (ci_lower, ci_upper, posterior_prob, ...).

segments seg_id, start, end, n, param_estimate.

regions, coefficients, constraints the optional slots, as arrays of objects (or an object), or null.

diagnostics residual_dependence (Ljung-Box p_value, lag, acf1), na_omitted and expected_false_positives where recorded.

assumptions the cpt_assumptions() rows, or null.

call the call that made the result, as text.

data {"index", "value", "fitted", "coordinates"} arrays, or null. coordinates is an object with one array per coordinate of a multivariate result (null for a single series), whose first is value.

See Also

`cpt_export()` to write a file, `cpt_import()` to read one back, `cpt_report(format = "json")`.

Other result class: `annotate_segments()`, `as_cpt_series()`, `as_ggcpt()`, `cpt_annotations()`, `cpt_export()`, `cpt_verify()`, `ggchangepoint-conditions`, `is_ggcpt()`, `new_ggcpt()`, `print.ggcpt()`

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 3)), method = "pelt")
cat(as_json(fit, data = FALSE))
```

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. Each coordinate also gets its own `.fitted_<name>` and `.resid_<name>` (its segment means and residuals), 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`, `bocpd`, `beast`, `decafs`, `segmented`, `envcpt`, `mcp` and `bfast`; a formula fit carries its piecewise regression's fitted values the same way.

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`.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(60), rnorm(60, 3)), method = "pelt")
head(augment(fit))
```

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", "diagnostics"),
...
)

```

Arguments

<code>object</code>	A ggcpt object.
<code>show_segments</code>	Logical. Whether to draw the fitted segment means. Defaults to FALSE.
<code>show_ci</code>	Logical. Whether to draw confidence intervals for changepoint locations, when the engine provides them (columns <code>ci_lower</code> / <code>ci_upper</code> on the changepoints tibble, supplied by the engines <code>cpt_methods()</code> marks in its <code>ci</code> column: <code>smuce</code> , <code>hsmuce</code> , <code>strucchange</code> , <code>segmented</code> , <code>mcp</code> , <code>bfast</code> and <code>taylor</code>). <code>nsp</code> is marked there as well but is not drawn by this argument: it reports <code>region_start</code> / <code>region_end</code> rather than an interval around an estimate, which <code>show_regions</code> draws. Drawn as horizontal whiskers near the bottom of the panel. Defaults to FALSE.
<code>show_fit</code>	Logical. Whether to draw the engine's fitted signal (the fitted column of <code>\$data</code> , provided by the engines <code>cpt_methods()</code> marks in its <code>fitted</code> column: <code>smuce</code> , <code>hsmuce</code> , <code>cpop</code> , <code>bcp</code> , <code>bocpd</code> (a causal running mean), <code>beast</code> , <code>decafs</code> , <code>segmented</code> , <code>envcpt</code> , <code>mcp</code> and <code>bfast</code> , plus every formula fit). Defaults to FALSE.
<code>show_regions</code>	Logical. Whether to shade the significance regions an interval-valued method returns (the <code>regions</code> slot; currently <code>nsp_wrapper()</code>). 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.
<code>cptline_alpha</code>	Alpha for changepoint lines. Defaults to 1.
<code>cptline_color</code>	Color for changepoint lines. Defaults to "blue".
<code>cptline_type</code>	Linetype for changepoint lines. Defaults to "solid".
<code>cptline_linewidth</code>	Linewidth for changepoint lines. Defaults to 0.5.
<code>show_points</code>	Logical. Whether to draw data points. Auto-off above 500 obs.
<code>show_line</code>	Logical. Whether to draw the line. Defaults to TRUE.
<code>index</code>	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 <code>index</code> argument of <code>cpt_detect()</code>), and to the observation position when there is none.
<code>labels</code>	Optional <code>cpt_labels()</code> tibble. When supplied, the labelled regions are shaded behind the series and coloured by the outcome <code>cpt_label_error()</code> 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.
<code>type</code>	Which view to draw. "series" (default) is the series with its changepoints; "statistic" and "path" delegate to <code>ggcpt_statistic()</code> and <code>ggcpt_solution_path()</code> , which error with the list of supporting engines when this one does not expose the internals. "scale_space" delegates to <code>ggcpt_scale_space()</code> , which needs nothing from the engine: it sweeps a multiscale detector over the same series,

so it draws for any result. "diagnostics" is the residual check of the fit in four panels: the residuals with the segment boundaries (structure left behind), their autocorrelation per segment (the independence most engines assume), a normal Q-Q plot per segment (the Gaussian cost) and the spread of each segment (a constant variance); `cpt_gof()` gives the same checks as numbers.

... 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.

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(60), rnorm(60, 3), rnorm(60)), method = "pelt")
ggplot2::autoplot(fit)
ggplot2::autoplot(fit, type = "diagnostics")

# a dated series draws its dates
dated <- cpt_detect(Nile, method = "pelt", change_in = "meanvar")
ggplot2::autoplot(dated)
```

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

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

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()`, `idetector_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	<i>BEAST wrapper: Bayesian estimation of abrupt change, seasonality, and trend</i>
---------------	--

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

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

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()`, `idetector_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-NA 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(bfast_wrapper(c(rnorm(60), rnorm(60, 4)), seed = 2026),
           silent = TRUE)
if (!inherits(res, "try-error")) res$changepts
```

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 <code>bfast::bfast()</code> .

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()`, `idetector_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	<i>Fast binary segmentation across loss functions</i>
--------------------	---

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

<code>x</code>	A numeric vector.
<code>change_in</code>	"mean" (Gaussian, the default) or "meanvar", mapped to the engine's <code>mean_norm</code> and <code>meanvar_norm</code> distributions. binsegRcpp has no variance-only cost, so "var" is not offered here; use "meanvar", or <code>cpt_detect</code> (<code>method = "pelt"</code> , <code>change_in = "var"</code>) for a variance-only change.
<code>distribution</code>	Loss function, overriding the mapping from <code>change_in</code> . Run <code>binsegRcpp::get_distribution_info()</code> for the list ("mean_norm", "meanvar_norm", "poisson", "l1", ...).
<code>max_segments</code>	Largest number of segments searched. Defaults to <code>min(20, floor(n / 5))</code> .
<code>n_segments</code>	Number of segments to report. When NULL (the default) it is chosen by BIC over the nested family the engine returns.
<code>min_segment_length</code>	Minimum segment length. Defaults to 2 for the single-parameter costs (<code>mean_norm</code> , <code>poisson</code> , <code>l1</code>), whose engine default allows one-observation segments, and to the engine's own default otherwise.

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()`, `fabiseach_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetector_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	<i>Bayesian online changepoint detection wrapper (BOCPD)</i>
---------------	--

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

<code>x</code>	A numeric vector.
<code>hazard</code>	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).
<code>...</code>	Additional arguments passed to <code>ocp::onlineCPD()</code> .

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()`, `idetector_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 `ar10`.

Usage

```
cpm_wrapper(x, cpm_type = "Mann-Whitney", ar10 = NULL, startup = 20, ...)
```

Arguments

<code>x</code>	A numeric vector.
<code>cpm_type</code>	Test statistic, passed to <code>cpm::processStream()</code> as <code>cpmType</code> . 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, with a smoothing <code>lambda</code> passed through <code>...</code> : 0.1, <code>cpm</code> 's default, or 0.3).
<code>ar10</code>	Target in-control average run length (how many observations, on average, before a false alarm). 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 change-points. The grid is the same for every <code>cpm_type</code> , and 50000 is the ceiling: a long series cannot be given an <code>ar10</code> proportional to its length indefinitely. When NULL (the default) it scales with the series : the smallest grid value of at least $5n$, capped at 50000. A sequential test run over a whole series raises about $n/ar10$ false alarms, so the streaming default of 500 meant 36 changepoints on 10,000 observations with 4 real ones, and 350 on 100,000; $5n$ turned the 36 into exactly the 4. The expected count is recorded as <code>\$diagnostics\$expected_false_positives</code> . Pass <code>ar10 = 500</code> for the 0.5.0 behaviour, and use <code>cpt_monitor()</code> for genuine streaming, whose default is unchanged.
<code>startup</code>	Number of observations after each restart before monitoring begins. Defaults to 20.
<code>...</code>	Additional arguments passed to <code>cpm::processStream()</code> .

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: `bcg_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()`, `idetector_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

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

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()`, `idetector_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_annotators

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_annotators(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\(\)](#), [as_json\(\)](#), [cpt_export\(\)](#), [cpt_verify\(\)](#), [ggchangepoint-conditions](#), [is_ggcpt\(\)](#), [new_ggcpt\(\)](#), [print_ggcpt\(\)](#)

Examples

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

cpt_assumptions	<i>Check the assumptions behind a segmentation</i>
-----------------	--

Description

A short report on whether the result's assumptions look violated for this series, what that does to the answer, and what would model it. It reports and does not act: every automatic correction measured trades true changepoints for false ones, and which side of that trade matters is the analyst's decision.

Usage

```
cpt_assumptions(fit, lag = NULL)

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

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

Arguments

fit	A ggcpt object.
lag	Lag for the Ljung-Box test. Defaults to 10, or $n / 5$ for a short series.
x	A ggcpt_assumptions object.
...	Ignored.

Value

A ggcpt_assumptions tibble with one row per check: component, value, flag (TRUE when the check raises a concern), detail and advice. The components:

residual_dependence Ljung-Box p-value on the within-segment residuals (and their lag-1 autocorrelation in detail). Flagged below 0.05. Autocorrelated noise is the commonest reason for spurious changepoints.

scale_sensitivity the noise standard deviation, flagged when the engine's answer depends on the data's units (measured) and the noise is far from unit scale.

expected_false_positives how many changepoints the engine is expected to report on this much pure noise: $n / ar10$ for cpm, and the measured null-size table otherwise. Flagged above one.

count_plausibility reported changepoints per hundred observations, flagged above one: more than $n / 100$ is more often misconfiguration than a finding.

data_type what the series looks like (continuous, counts, binary, proportions), flagged when a Gaussian cost is fitted to non-Gaussian data.

See Also

`cpt_report()`, which prints this; `cpt_robustness()` for the answer under other noise models.

Other inference: `cpt_attribute_event()`, `cpt_effect()`, `cpt_gof()`, `cpt_null_power()`, `cpt_robustness()`, `cpt_test_at()`, `cpt_test_null()`

Examples

```
set.seed(1)
ar <- as.numeric(arima.sim(list(ar = 0.7), 300))
cpt_assumptions(cpt_detect(ar, method = "pelt"))
```

cpt_attribute_event	<i>Was a detected change the event you have in mind?</i>
---------------------	--

Description

The mirror of `cpt_test_at()`. A detector found a change on 14 March and something is known to have happened on 1 March: is that the same event? The detection's confidence interval answers it: an event inside the interval is consistent with being the change, one outside is not.

Usage

```
cpt_attribute_event(fit, event, level = 0.95, ...)
```

Arguments

fit	A ggcpt object.
event	One or more event locations: positions, values of the fit's index, or a <code>cpt_annotate_events()</code> result.
level	Confidence level of the interval. Defaults to 0.95.
...	Further arguments for <code>cpt_confint()</code> (for example B or seed for a bootstrap interval).

Value

A tibble with one row per event: event, event_position, the nearest changepoint cp (and cp_index), its interval ci_lower/ci_upper, distance (positions from the changepoint to the event), inside and a plain-language verdict.

See Also

Other inference: `cpt_assumptions()`, `cpt_effect()`, `cpt_gof()`, `cpt_null_power()`, `cpt_robustness()`, `cpt_test_at()`, `cpt_test_null()`

Examples

```
set.seed(1)
dates <- as.Date("2026-01-01") + 0:119
fit <- cpt_detect(c(rnorm(60), rnorm(60, 3)), method = "strucchange",
                 index = dates)
cpt_attribute_event(fit, as.Date(c("2026-02-28", "2026-04-10")))
```

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 <code>cpt_batch()</code> , a numeric matrix or data frame (columns are series) or a list of numeric vectors; for the <code>print()</code> and <code>tidy()</code> methods, a <code>ggcpt_batch</code> object.
method	Detection method, passed to <code>cpt_detect()</code> .
change_in	What to detect change in, passed to <code>cpt_detect()</code> .

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::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 ggcp_batch object (for autoplot()).

Value

A ggcp_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 ggcp 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). A method that fails on one dataset (its engine errors, is not installed, or cannot take that series: see [ggchangepoint-conditions](#)) records the message and the grid continues. Invalid data or arguments, which every method would reject, stop the benchmark instead, naming the dataset and method, so a mistake in the call is not reported as a finding about an engine.

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

<code>datasets</code>	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 <code>series</code> and one of <code>truth</code> , <code>changepoints</code> (an integer vector) or <code>annotations</code> (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.
<code>methods</code>	Character vector of method names.
<code>metrics</code>	Which metrics to keep. Defaults to <code>c("covering", "f1")</code> , the pair van den Burg and Williams (2020) established as the benchmark standard.
<code>tolerance</code>	Matching window passed to the metrics. Defaults to 5.
<code>change_in</code>	Passed to every detector.
<code>parallel</code>	Use <code>future::plan()</code> when future.apply is available? Defaults to TRUE.
<code>progress</code>	Show a progressr progress bar when that package is installed and a handler is enabled? Defaults to TRUE.
<code>seed</code>	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> 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 <code>cpt_detect()</code> call.
x	A <code>ggcpt_benchmark</code> object.
object	A <code>ggcpt_benchmark</code> object (for <code>autoplot()</code>).
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 <code>plot_type = "critical_difference"</code> , 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 $k = 2, \dots, 10$). 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_calibration

What the package promises, against what it delivers

Description

The calibration suite: every guarantee the documentation states that a simulation can check, with the realised value and its Monte Carlo standard error. Location intervals are scored for coverage (conditional on a changepoint landing within 20 positions of the truth), tests for their size at 5% on data with no change, and monitors for their in-control run length. The row for `cpt_test()` at a location the data chose is there to show what `selection_adjusted = FALSE` means: its nominal 5% is not what it delivers.

Usage

`cpt_calibration`

Format

A tibble with one row per guarantee and setting:

guarantee what is promised, and by which function.

call the call that was measured.

setting the data and the options it was measured on.

kind "coverage", "size" or "run_length".

nominal the promised value: a confidence level, a test's level or an average run length (for the e-detector, a lower bound).

realised the measured value.

mcse its Monte Carlo standard error.

reps replicates that counted (for coverage, those that detected the change).

Source

data-raw/calibration.R.

See Also

The calibration article on the package website, <https://pursuitofdatascience.github.io/ggchangeplot/articles/calibration.html>.

Other measurement tables: [cpt_data_types](#), [cpt_invariances](#), [cpt_noise_benchmark](#), [cpt_null_sizes](#), [cpt_runtimes](#)

Examples

```
cpt_calibration[, c("call", "nominal", "realised", "mcse")]
```

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 <code>method = "auto"</code> 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 break-point intervals, mcp posterior intervals, bfast break confidence intervals, and taylor's bootstrap confidence limits). The engines that supply them are the ones <code>cpt_methods()</code> 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 <i>not</i> intervals around an estimate (see <code>nsp_wrapper()</code>), but they are the strongest guarantee available, so the mapping is reported as <code>source = "nsp_region"</code> 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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Passed to <code>cpt_detect()</code> on the bootstrap replicates, or to <code>nsp_wrapper()</code> .

Value

A tibble with one row per changepoint: `cp`, `ci_lower`, `ci_upper`, `level`, `source`, plus `cp_index/ci_lower_index/ci_upper_index` 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 = "majority",
  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

<code>x</code>	A <code>ggcpt_consensus</code> object (for <code>print()</code>).
<code>methods</code>	Character vector of method names. A method that fails on this series (its engine errors or is not installed, or it does not offer <code>change_in</code>) is left out of the vote with a warning; invalid data or arguments stop the call.
<code>tolerance</code>	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 "majority": more than half of the methods that ran (two of three, three of four or five), so the rule does not silently change with the size of the panel, as a fixed count of two did (a bare majority of three, a fifth of ten). "all" asks for unanimity. A value strictly between 0 and 1 is read as a proportion of the methods that ran; anything else is a count, rounded <i>up</i> (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 <i>least</i> 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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Passed on to <code>autoplot.ggcpt()</code> for <code>plot_type = "series"</code> .
object	A <code>ggcpt_consensus</code> object (for <code>autoplot()</code>).
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 does not remove shared false positives

Voting removes the idiosyncrasies of individual algorithms. It does not remove a false positive that every algorithm makes for the same reason, and a violated assumption is exactly such a reason: measured under AR(1) noise, every one of `pelt`'s spurious changepoints was reproduced by `wbs` within five observations, and the 2-of-3 consensus of `pelt`, `binseg` and `wbs` had more false positives (1.40) than two of its own members. The useful signal in a panel is its **disagreement**: the disagreement value in the `consensus` attribute is the share of the locations any member found that not every member found, and a high value says the members' shared assumption may be what is wrong (see `cpt_assumptions()` and `cpt_robustness()`).

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

<code>x</code>	For <code>cpt_crops()</code> , a numeric vector, or a <code>ggcpt</code> fit whose series is used (with its change type, when it is one CROPS sweeps); for the <code>print()</code> and <code>tidy()</code> methods, a <code>ggcpt_path</code> object.
<code>change_in</code>	What to detect change in: "mean", "var", or "meanvar". Defaults to "mean".
<code>pen_min, pen_max</code>	The penalty interval to sweep. Default to $\log(n)$ and $10 * \log(n)$.
<code>...</code>	Additional arguments passed to the underlying <code>changepoint::cpt.mean()</code> , <code>cpt.var()</code> , or <code>cpt.meanvar()</code> call.
<code>object</code>	A <code>ggcpt_path</code> object (for <code>autoplot()</code>).
<code>type</code>	Plot type for <code>autoplot()</code> : "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 change-points drawn).
<code>max_facets</code>	Maximum number of solutions shown by <code>type = "segmentations"</code> . 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_data_types	<i>False positives and power by data type</i>
----------------	---

Description

Ten replicates of a 400-point series with one change at 200, at comparable effect sizes across four data types: Gaussian (a shift of 1.4 standard deviations), Bernoulli (probability 0.15 to 0.75), Poisson (rate 2 to 10) and a proportion (binomial of 20, 0.25 to 0.6). Each engine is at its default, as a user who does not set family gets it.

Usage

```
cpt_data_types
```

Format

A tibble with one row per engine and data type:

method the engine.

data_type "gaussian", "bernoulli", "poisson" or "proportion".

reps replicates that ran.

hit_rate share of replicates finding the change within 20 positions.

fp mean number of other reported changepoints.

Source

data-raw/measure_engines.R (part datatype).

See Also

Other measurement tables: [cpt_calibration](#), [cpt_invariances](#), [cpt_noise_benchmark](#), [cpt_null_sizes](#), [cpt_runtimes](#)

Examples

```
b <- subset(cpt_data_types, data_type == "bernoulli")
b[order(-b$fp), ][1:5, ]
```

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

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

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

<code>object</code>	A <code>ggcpt_delay</code> object (for <code>autoplot()</code>).
<code>truth</code>	Integer vector of true changepoint positions, on the same clock as the alarms. When <code>object</code> is a monitor built by <code>cpt_replay()</code> the baseline offset is applied automatically.
<code>max_delay</code>	Alarms further than this after a change are treated as false alarms rather than late detections. Defaults to <code>Inf</code> .
<code>x</code>	A <code>ggcpt_delay</code> object.
<code>...</code>	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 `ar1`: 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,
  ...,
  data = NULL,
  family = NULL,
  group = NULL,
  na_action = c("error", "omit", "engine"),
  fixed = NULL,
  within = NULL,
  min_segment = NULL,
  min_effect = NULL,
  keep_fit = TRUE
)
```

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 `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`).

Also a **formula** with data: $y \sim x_1 + x_2$ fits breakpoints in a regression (methods "strucchange", "segmented" and "fastcpd"; see `cpt_methods()`'s formula column), and $y \sim 1$ is the response alone, which every univariate method takes.

Several series at once: a **dplyr**-grouped data frame, a keyed tsibble, or a long data frame with group naming the column(s) that identify each series. Each group is detected separately and the answer is a `cpt_batch()` result carrying the grouping columns.

Refused rather than coerced: a factor (its level codes are not data), a survival object, and dates or timestamps as values (they convert to an increasing count; pass them as index).

method	Detection method: any method in <code>cpt_methods()</code> whose status is "available" or "registered". Methods whose engines live in Suggests report what to install when missing; <code>cpt_register_method()</code> 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 <code>cpt_methods()</code>); incompatible combinations error rather than silently running something else. A <i>compatible</i> 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 <code>change_in</code> 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 <code>cpt_penalty</code> 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 change-point does not implement MBIC for Segment Neighbourhood. Note also that the default "MBIC" is resolved to a <i>numeric</i> 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 <code>cpt_detect(x, method = "decafs")</code> can report fewer changepoints than <code>decafs_wrapper(x)</code> on the same series. Pass <code>penalty</code> 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 <code>tidy()</code> (as <code>cp_index</code>), <code>augment()</code> , <code>autoplot()</code> and <code>cpt_report()</code> , so the output speaks in the user's own units. An index that is not equally spaced warns. When <code>x</code> is a data frame and <code>y</code> is given, <code>index</code> selects a column of that data frame instead of being a vector.

y	Column selection for the data-frame interface: <code>cpt_detect(df, y = value, index = date, method = "pelt")</code> . 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 <code>change_in</code> (not's contrast, cpm's <code>cpm_type</code> , kcp's <code>running_stat</code> , 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. A name that is a near miss of a real argument of <code>cpt_detect()</code> , the wrapper or its engine (<code>n_interval</code> for <code>n_intervals</code>) is refused with the suggestion, and a modelling-choice value outside the method's vocabulary (<code>cpt_methods()</code> \$choices, e.g. <code>test.stat = "Poison"</code>) is refused with the nearest valid value. <code>seed</code> is honoured by every method: passed to a wrapper that has one, and otherwise used to scope the random stream around the fit (restored afterwards); <code>seed = NULL</code> means unset.
data	A data frame for the formula interface.
family	The distribution a segment is modelled with: "gaussian", "poisson", "binomial", "exponential", "gamma", "laplace" or "l1". NULL (the default) runs each method's own default, which for every parametric method is Gaussian. A single-parameter family changes in its rate or probability, so it is used with <code>change_in = "mean"</code> . Which methods fit which families, and the engine call each combination becomes, is in <code>cpt_families()</code> ; a combination not listed there is refused. The data are checked against the family (counts must be whole and non-negative, binary data 0/1, waiting times positive). fastcpd 's and stepR 's own family values ("ar", "hsmuce", ...) still reach those engines.
group	For a long data frame holding several series, the column(s) identifying each one: a bare name, a string, or <code>c(a, b)</code> .
na_action	What to do with missing values: "error" (the default) refuses them; "omit" drops them, detects on what is observed, and reports every location in the original positions (a changepoint lands on the last observed point of its left segment), keeping the gaps in <code>\$data</code> so plots show them and the translation in <code>\$diagnostics\$na_omitted</code> ; "engine" passes them to an engine that handles them in place, which measured for beast and segmented only (see <code>cpt_methods()</code> \$na_handling), and is refused for the rest. There is no "impute": imputing and then detecting pulls the estimated location towards the imputed stretch, by more the longer the gap.
fixed	Changepoints known in advance, which are kept and the rest estimated around them: the detector runs separately on each stretch between them. Positions, or values of the series' index (a date is placed at the last observation on or before it; for a numeric index such as a ts's years, a number outside 1..n is read as an index value and one inside as a position).
within	Windows the changepoints must fall in ("the policy took effect sometime in Q2"): a pair <code>c(start, end)</code> , a list of pairs, or a two-column table, in positions

or index values; a bound beyond either end of the series is read as that end. The engine searches the whole series and changepoints outside every window are dropped (recorded in `$constraints`); it restricts what is reported rather than re-optimising.

<code>min_segment</code>	Minimum segment length in observations, translated into each engine's own argument (<code>minseglen</code> , <code>min_size</code> , <code>h</code> , <code>delta</code> , ...: <code>cpt_methods()</code> <code>\$min_segment</code> names it). A method without one refuses it. When <code>min_segment</code> is not given, the engines whose own default allows one-observation segments (the change-in-mean changepoint methods other than <code>SegNeigh</code> , <code>np</code> and <code>binsegrcpp</code>) get a floor of two, so a short or monotone series is not split after every point.
<code>min_effect</code>	Drop changepoints whose mean shift is smaller than this many noise standard deviations (estimated from the series' first differences, which a changepoint barely moves), removing the smallest first and re-measuring its neighbours after each removal. For a change in mean only. The size is measured on the same data that located the change, so it is biased upward; see <code>cpt_effect()</code> .
<code>keep_fit</code>	Keep the engine's own object in <code>\$fit</code> ? Defaults to <code>TRUE</code> . A few engines return objects far larger than the answer (a warning says so above 10 MB, once per method per session); <code>FALSE</code> drops it, which only the engine-specific accessors notice. Everything else in the result is plain data, so a saved result reads back in any R session, with no packages installed.

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 $\sigma = 1$, 27 at $\sigma = 3$ and 76 at $\sigma = 10$. These are means over 20 draws, because a single draw is not stable here: the same settings gave 15/38 at $n = 100$ and 47/152 at $n = 400$, so the effect grows with the series as well as with the noise. (0.5.0, before the two-point minimum segment, reported 39 and 141 at $n = 200$: the floor halves the damage and does not remove it.) A measured scale-sensitive engine given noise far from unit scale warns (class `ggchangepoint_scale_sensitive`). 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. Five 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; "bocpd"'s default prior is on the data's own scale; and "envcpt" and "var" changed their answer with the units too (var's gamma_set and lambda_set penalise a squared-error loss on an absolute scale). At a thousandth of the units "decafs", "bocpd" and "var" found nothing. Standardising first avoids all five, and [cpt_invariances](#) holds the measurement.

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")

# Counts, with a cost written for them.
counts <- c(rpois(100, 2), rpois(100, 6))
cpt_detect(counts, method = "pelt", family = "poisson")

# Missing values, located in the original positions.
gappy <- x
gappy[c(20, 21, 150)] <- NA
cpt_detect(gappy, method = "pelt", na_action = "omit")$changepoints

# A changepoint known in advance, and a window for another.
cpt_detect(x, method = "pelt", fixed = 50)$changepoints

# Breakpoints in a regression.
if (requireNamespace("strucchange", quietly = TRUE)) {
  d <- data.frame(t = 1:200, z = rnorm(200))
  d$y <- ifelse(d$t <= 120, 1 + d$z, 3 - d$z) + rnorm(200, 0, 0.5)
  fit <- cpt_detect(y ~ z, data = d, method = "strucchange")
  tidy(fit, "coefficients")
}
```

}

cpt_effect

*Effect size at each changepoint***Description**

The size of every change a fit reports, in the data's own units and standardised: the level before and after, their difference with an interval, the difference in noise standard deviations, the percentage change, and for count and waiting-time data the rate or hazard ratio.

Usage

```
cpt_effect(fit, level = 0.95, method = c("naive", "split"), seed = NULL, ...)

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

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

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

Arguments

fit	A ggcpt object.
level	Confidence level for the intervals. Defaults to 0.95.
method	"naive" (the default) measures each effect on the data that located the change. "split" locates the changepoints on the odd-numbered observations and measures their effects on the even-numbered ones, which removes the winner's curse at the price of locating on half the data (see below).
seed	Optional seed for method = "split", scoped to this call.
...	Further arguments for the re-run method = "split" makes (overriding what is recovered from the fit).
x	A ggcpt_effect object.
object	A ggcpt_effect object.

Value

A ggcpt_effect tibble with one row per changepoint (per changepoint and coordinate for a multivariate fit, per changepoint and term for a regression fit): cp (and cp_index), before, after, n_before, n_after, delta, delta_lower, delta_upper (a Welch interval), delta_std (the change in noise standard deviations, the unit `cpt_detect(min_effect = )` filters on), pct_change, sd_before, sd_after, sd_ratio, the family-specific ratio where it applies (rate_ratio for Poisson counts, odds_ratio for binary data, hazard_ratio for exponential waiting times, each with an interval), selection_adjusted and method. With `print()` and `autoplot()`.

The winner's curse

A change measured at a location the same data chose is biased upward: the detector put the boundary where the two sides differ most, so the difference it reports is the largest the noise allowed. The bias is worst for the marginal detections, which are exactly the ones whose size matters. `method = "naive"` rows carry `selection_adjusted = FALSE` for that reason, except at a changepoint fixed in advance with `cpt_detect(fixed = )`, which the data did not locate and which is `TRUE`. `method = "split"` is honest about the size (the measuring half never influenced the locations) and less precise about the location (the locating half has half the data), and it assumes the noise is independent from one observation to the next, since the two halves are interleaved.

See Also

`cpt_test_at()` for a change at a date fixed in advance, `cpt_detect()`'s `min_effect`.

Other inference: `cpt_assumptions()`, `cpt_attribute_event()`, `cpt_gof()`, `cpt_null_power()`, `cpt_robustness()`, `cpt_test_at()`, `cpt_test_null()`

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(100, 10), rnorm(100, 12)), method = "pelt")
cpt_effect(fit)
```

```
counts <- c(rpois(100, 4), rpois(100, 8))
cpt_effect(cpt_detect(counts, method = "pelt", family = "poisson"))
```

cpt_export

Write a result to a file, and read it back

Description

`cpt_export()` writes a result where software in any language can read it; `cpt_import()` rebuilds a working `ggcpt` from the file. A saved `.rds` already reads in any R session without this package (a `ggcpt` is a plain list of plain data), so these are for leaving R, or for a format a person can open.

Usage

```
cpt_export(fit, file, format = NULL, ...)
```

```
cpt_import(file, format = NULL, method = "imported", change_in = "mean")
```

Arguments

<code>fit</code>	A <code>ggcpt</code> object.
<code>file</code>	Path to write (<code>cpt_export()</code>) or read (<code>cpt_import()</code>).

format "json" (the full result, see [as_json\(\)](#)) or "csv" (one row per observation: index, value (for a multivariate result, one value_<name> per coordinate), the time index as index_value when there is one, seg_id, fitted and is_changepoint, which is everything needed to rebuild the segmentation but not its metadata). Defaults to the file's extension.

... For `cpt_export()`, further arguments for [as_json\(\)](#).

method, change_in For a CSV, which carries no metadata, the method and change type to record on the rebuilt result. Defaults to "imported" and "mean".

Value

`cpt_export()` returns file, invisibly. `cpt_import()` returns a `ggcpt` object; the engine's own `$fit` is not stored in either format, so it is `NULL`.

See Also

Other result class: [annotate_segments\(\)](#), [as_cpt_series\(\)](#), [as_ggcpt\(\)](#), [as_json\(\)](#), [cpt_annotations\(\)](#), [cpt_verify\(\)](#), [ggchangepoint-conditions](#), [is_ggcpt\(\)](#), [new_ggcpt\(\)](#), [print.ggcpt\(\)](#)

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(50), rnorm(50, 3)), method = "pelt")
path <- tempfile(fileext = ".json")
cpt_export(fit, path)
back <- cpt_import(path)
back$changepoints
unlink(path)
```

cpt_families

Distribution families, and what each request runs

Description

The family argument of [cpt_detect\(\)](#) names the distribution a segment is modelled with, so count data, waiting times and binary series can be segmented with a cost written for them rather than a Gaussian one. `cpt_families()` is the table behind it: every legal combination of method, change_in and family, and the engine call each becomes.

Usage

```
cpt_families(method = NULL)
```

Arguments

method Optional method name(s) to restrict the table to.

Value

A tibble with one row per legal combination: `method`, `change_in`, `family` (NA for a distribution-free method, which takes no family), `engine_call` (what is run) and `data` (what the family requires of the series: "counts", "binary", "positive" or NA).

The families

"gaussian" The default of every parametric method. On "cpm" it also switches from the nonparametric statistics to the Gaussian ones (Student, Bartlett).

"poisson" Counts: `change_in = "mean"` is a change in the rate. Needs non-negative whole numbers.

"binomial" Binary outcomes: a change in the success probability. Needs 0/1 data.

"exponential" Waiting times: a change in the rate (the hazard), so `change_in = "mean"`. Needs positive data.

"gamma" Positive data with a known shape (pass shape, default 1): a change in the scale.

"laplace" Heavy tails: a change in the median and the scale (`change_in = "meanvar"`).

"l1" The absolute-loss cost: a change in the median, robust to outliers (`change_in = "mean"`).

A single-parameter family has one thing to change, so Poisson, binomial, exponential and gamma requests use `change_in = "mean"`.

Why an argument and not a model grammar

A grammar that marks the changing parameter (`poisson(rate = NA)`, `normal(mean = NA, sd = 1)`) would unify `change_in`, `family` and the formula interface in one expression, and measured against the fifty methods it fits 42 of them cleanly. It is deferred to 1.0, where `change_in` can be replaced rather than joined by a second idiom, and this table is the first half of it: `change_in = "mean"`, `family = "poisson"` already means `poisson(rate = NA)` internally.

See Also

`cpt_detect()`, `cpt_methods()` (its `families` column), `cpt_simulate()` to generate data from each family.

Examples

```
cpt_families("pelt")
# every method that can fit a Poisson rate
unique(subset(cpt_families(), family == "poisson")$method)

set.seed(1)
counts <- c(rpois(100, 3), rpois(100, 9))
cpt_detect(counts, method = "pelt", family = "poisson")
```

cpt_gof	<i>Goodness of fit, segment by segment</i>
---------	--

Description

The checks behind a segmentation, as numbers: for every segment, its length, level and spread, whether its residuals look independent (Ljung-Box) and Gaussian (Shapiro-Wilk), and a flag where the segment is too short for either test to mean anything.

Usage

```
cpt_gof(fit, lag = NULL)

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

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

Arguments

fit	A ggcpt object.
lag	Ljung-Box lag. Defaults to $\min(10, n / 5)$ per segment.
x	A ggcpt_gof object.
...	Ignored.

Value

A ggcpt_gof tibble with one row per segment (per segment and coordinate for a multivariate fit): seg_id, start, end, n, mean, sd, acf1 (lag-1 autocorrelation of the residuals), ljung_box_p, shapiro_p, and too_short (fewer than 12 observations, where the tests are left NA). The whole-series Ljung-Box result is attached as the "overall" attribute, and print() summarises what the columns say.

See Also

autoplot(fit, type = "diagnostics") draws the same checks; [cpt_assumptions\(\)](#) reports them with the alternatives.

Other inference: [cpt_assumptions\(\)](#), [cpt_attribute_event\(\)](#), [cpt_effect\(\)](#), [cpt_null_power\(\)](#), [cpt_robustness\(\)](#), [cpt_test_at\(\)](#), [cpt_test_null\(\)](#)

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(100), rnorm(100, 3, 2)), method = "pelt")
cpt_gof(fit)
```

cpt_gt	<i>A publication-ready changepoint table</i>
--------	--

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	<i>Influence diagnostics for a changepoint segmentation</i>
---------------	---

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 <code>ggcpt_influence</code> object (for <code>autoplot()</code>).
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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to <code>cpt_detect()</code> 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 , HDCCD , 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 <code>install.packages()</code> .

Details

Every package comes from the repositories in `getOption("repos")` except the two engines CRAN has archived. **fpop**, archived in September 2026 at its maintainer's request, is installed from R-Forge (<https://R-Forge.R-project.org>), where it is developed. **ocp**, archived in October 2026, is installed from the package website (<https://pursuitofdatascience.github.io/ggchangepoint/repo/>), a CRAN-like repository that serves the last CRAN release of every engine CRAN no longer does.

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_invariances	<i>Invariances of every engine's answer</i>
-----------------	---

Description

Five replicates of a 200-point series with changes at 70 and 140 (a 3-coordinate version for the multivariate engines), detected as given and transformed. A correct mean-change detector returns the same changepoints on $10 * x$, $0.1 * x$ and $x + 100$, and $n - cp$ on the reversed series.

Usage

```
cpt_invariances
```

Format

A tibble with one row per engine:

method the method.

reps replicates that ran.

k median changepoints on the series as given.

scale_invariant the same answer at x , $10 * x$ and $0.1 * x$ in at least four of five replicates. FALSE for the Gaussian-cost engines that assume unit noise, which is why `cpt_detect()` warns when one is handed noise far from unit scale.

shift_invariant the same answer on $x + 100$.

reversal_rate share of replicates whose reversed series gives the mirrored answer.

sequential the reversal test fails in most replicates: the answer depends on the direction of time.

concat_k median changepoints on $c(x, x)$ (four expected, one for a single-change design).

dup_k median changepoints when every observation is repeated twice, which makes the noise strongly autocorrelated: a measure of how badly the engine fails when its independence assumption does.

Source

data-raw/measure_engines.R (part invariance).

See Also

Other measurement tables: [cpt_calibration](#), [cpt_data_types](#), [cpt_noise_benchmark](#), [cpt_null_sizes](#), [cpt_runtimes](#)

Examples

```
subset(cpt_invariances, !scale_invariant)$method
```

cpt_label_error	<i>Score a segmentation against labels</i>
-----------------	--

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'
tidy(x, ...)

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

Arguments

<code>x</code>	A <code>ggcpt_label_curve</code> object (for <code>print()</code>).
<code>labels</code>	A <code>cpt_labels</code> tibble.
<code>method</code>	Detection method. Defaults to "pelt".
<code>penalties</code>	Numeric vector of penalties to try. When <code>NULL</code> (the default) the grid is chosen <i>adaptively</i> : 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 <code>cpt_learn_penalty()</code> . The probe costs at most a dozen extra detector fits; pass penalties explicitly for an expensive engine.
<code>change_in</code>	Passed to the detector.
<code>...</code>	Additional arguments passed to <code>cpt_detect()</code> .
<code>object</code>	A <code>ggcpt_label_curve</code> object (for <code>autoplot()</code>).

Value

A `ggcpt_label_curve` object: a tibble with `penalty`, `n_cp`, `errors`, `false_positive`, `false_negative`, plus `print()`, `tidy()` 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_labels	<i>Changepoint labels</i>
------------	---------------------------

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, Rigai 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_learn_penalty	<i>Learn a penalty from labelled series</i>
-------------------	---

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'
tidy(x, ...)
```

```
## 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 <code>cpt_labels</code> 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 <code>cpt_label_error_curve()</code> .
engine	"auto" (default) uses penaltyLearning 's <code>IntervalRegressionCV()</code> 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 <code>cpt_detect()</code> while building the curves.
x	A <code>ggcpt_penalty_model</code> (for <code>print()</code> and <code>coef()</code>).
object	A <code>ggcpt_penalty_model</code> .
newdata	A numeric vector (one series), or a list/matrix of series.

Value

A `ggcpt_penalty_model` object with `print()`, `coef()`, `tidy()` (the coefficients as term and estimate) 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, Rigai 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	<i>Rank observations by influence</i>
--------------	---------------------------------------

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 NAs, 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

<code>name</code>	Dataset name, a character vector of names, or NULL (the default) for the whole catalogue.
<code>cache_dir</code>	Where to cache. Defaults to <code>tools::R_user_dir("ggchangepoint", "cache")</code> .
<code>refresh</code>	Re-download even when a cached copy exists? Defaults to FALSE.
<code>quiet</code>	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 $n/\text{ar}l0$ 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.

families, choices, formula, min_segment, na_handling, cp_convention_upstream The modelling vocabulary (omitted when `capabilities = FALSE`): the distribution families `cpt_detect(family = )` accepts (see `cpt_families()`; NA for a distribution-free method); the engine's modelling-choice arguments and their legal values, which `cpt_detect()` validates; whether it takes a formula with covariates; the engine argument `min_segment` is translated into; what the engine does with a missing value, *measured* rather than read from its documentation ("native", "compacts", "silent_loss" or "reject"; see `cpt_detect(na_action = )`); and the engine's own changepoint convention, which the wrapper translates to "left" ("right": it reports the first observation after the change; "continuous": a location it estimates on a continuous scale; "design": rows of a lagged design).

scale_invariant, sequential, max_cp, tier, noise_model_arg, rate_arg, cost, max_n Measured and recorded properties (omitted when `capabilities = FALSE`): whether the answer is the same at x , $10 * x$ and $0.1 * x$; whether it depends on the direction of time; how many changepoints the method can return where that is fixed; "general" for the general-purpose workhorses; the argument that selects the noise model and the one that sets the false-alarm rate; the runtime

class at $n = 10,000$ ("fast" under a second, "moderate" under ten, "slow"); and the longest series it finished within the measurement's time cap (0 when it did not finish 1,000 observations). The measurements behind them are the `cpt_runtimes` and `cpt_invariances` data sets.

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	<i>Changepoint accuracy metrics</i>
-------------	-------------------------------------

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 = NULL, margin = 5)
```

Arguments

pred	Predicted changepoint indices (integer vector), or a <code>ggcpt</code> fit, whose change-points and length are used. A <code>cpt_monitor()</code> object is accepted with a warning: its alarm times are <i>detection</i> times, which lag each change by the detection delay, so the location metrics score the delay as an error; <code>cpt_delay()</code> is the evaluation built for a monitor.
truth	Ground truth changepoint indices (integer vector).
n	Length of the series. Taken from <code>pred</code> when it is a fit or a monitor.
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. It has a **floor**: an empty prediction scores about $1/(K+1)$ for K true changepoints (0.5 for one change in the middle), so covering is not comparable across problems with different K and should not be averaged over them. `cpt_benchmark()` ranks within each dataset instead for exactly that reason.

`covering_floor`, `covering_scaled` the covering an empty prediction scores on this truth (the sum of the squared true segment shares, $1/(K+1)$ when the segments are equal), and covering rescaled so that the floor is 0 and a perfect segmentation 1, $(covering - floor)/(1 - floor)$: comparable across problems with different K . NA when $K = 0$, where the floor is already 1.

`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**: $\text{abs}(n_pred - n_truth)$. 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.

`precision`, `recall` and `f1` are **thresholded** at margin: a changepoint off by margin counts as found and one off by margin + 1 as missed, so they can move from 1 to 0 on a one-observation change. covering, hausdorff, rand_index and the matched errors are continuous in the location.

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 $1 \dots (n - 1)$ 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 = NULL, margin = 5)
```

Arguments

<code>pred</code>	Predicted changepoint indices, or a <code>ggcpt</code> fit (see <code>cpt_metrics()</code>).
<code>annotations</code>	A list of ground-truth annotation vectors.
<code>n</code>	Length of the series; taken from <code>pred</code> when it is a fit.
<code>margin</code>	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)
```

<code>cpt_min_detectable</code>	<i>The smallest detectable change</i>
---------------------------------	---------------------------------------

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'
tidy(x, ...)

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

```

Arguments

<code>n</code>	Series length.
<code>sigma</code>	Noise standard deviation. Defaults to 1.
<code>method</code>	Detection method. Defaults to "pelt".
<code>power</code>	Target detection probability. Defaults to 0.8.
<code>range</code>	Search range for the change size, in standard deviations. Defaults to <code>c(0.1, 5)</code> . With a non-Gaussian family passed through ... to <code>cpt_power()</code> , the change is in the family's own parameter and the default follows it: <code>c(0.1, 5)</code> times the square root of the baseline rate for "poisson", up to a probability of 0.99 for "binomial", and <code>c(0.1, 5)</code> times the baseline mean for "exponential".
<code>n_sim</code>	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.
<code>tolerance</code>	A detection counts as finding the change when it falls within this many positions of it. Defaults to 5.
<code>change_in</code>	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 $\text{sigma} * (1 + \text{jump})$; "meanvar" does both at once, shifting the mean by jump and the standard deviation to $\text{sigma} * (1 + \text{jump} / 2)$, the same design <code>cpt_scenarios()</code> simulates; "slope" bends a flat line into one rising by $\text{jump} * \text{sigma}$ over the series.
<code>location</code>	Changepoint position, as a fraction of <code>n</code> in $(0, 1)$ or an integer position. Defaults to 0.5. A vector runs one scenario per value.
<code>noise</code>	Noise model, passed to <code>cpt_simulate()</code> .
<code>rho</code>	AR(1) parameter when noise = "ar1".
<code>df</code>	Degrees of freedom when noise = "t".
<code>tol</code>	Bisection tolerance on the change size. Defaults to 0.05.
<code>max_iter</code>	Maximum bisection steps. Defaults to 12.
<code>seed</code>	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	Ignored.
<code>x</code>	A <code>ggcpt_min_detectable</code> object.

Value

A list with `jump` (the smallest change reaching power), `achieved_power`, `mc_se`, and the trace of evaluations, with `print()` and `tidy()` (one row) methods.

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 <i>repeated</i> 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 <i>persistent</i> , 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 <code>cpt_delay()</code> then counts them all as false alarms. Set <code>relearn = 0</code> 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 <code>mc_reps</code> 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_noise_benchmark *False positives and power under four noise regimes*

Description

Ten replicates of a 300-point series with mean changes of two marginal standard deviations at 100 and 200, under independent Gaussian noise, heavy-tailed noise (t_3 , scaled to unit variance), AR(1) noise with $\rho = 0.7$ and heteroscedastic noise (standard deviations 0.5, 1 and 2.5 by segment), for every univariate engine at its default and at each setting of its noise-model argument. A recommendation is a call, not a name: `smuce` and `smuce` with `family = "hsmuce"` are different detectors.

Usage

`cpt_noise_benchmark`

Format

A tibble with one row per engine, setting and regime:

method, setting the engine and the argument setting ("default" or the argument as written).

call the `cpt_detect()` call the row measures.

regime "iid", "heavy", "ar1" or "hetero".

reps replicates that ran.

hits mean number of the two real changes found within ten positions.

fp mean number of reported changepoints more than ten positions from either.

Source

data-raw/measure_engines.R (part noise).

See Also

Other measurement tables: [cpt_calibration](#), [cpt_data_types](#), [cpt_invariances](#), [cpt_null_sizes](#), [cpt_runtimes](#)

Examples

```
ar <- subset(cpt_noise_benchmark, regime == "ar1")
ar[order(ar$fp), c("call", "hits", "fp")]
```

cpt_null_power	<i>What an empty answer could have detected</i>
----------------	---

Description

When a detection comes back empty, the useful question is whether a change of the size you care about would have been found. This runs [cpt_min_detectable\(\)](#) at the fit's own length, method and penalty, with the noise level estimated from the series, and reports the smallest shift detected with the requested power: changes smaller than that could well be there and missed.

Usage

```
cpt_null_power(fit, power = 0.8, n_sim = 50, seed = NULL, ...)

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

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

Arguments

fit	A ggcpt object (usually one with no changepoints; it works for any).
power	Target power. Defaults to 0.8.
n_sim	Simulations per power evaluation. Defaults to 50.
seed	Optional seed, scoped to this call.
...	Further arguments for cpt_min_detectable() .
x	A ggcpt_null_power object.

Value

A ggcpt_null_power list with n, sigma (the noise level used), jump (the smallest detectable change, in the data's units), jump_sd (the same in noise standard deviations), change_in (the fit's change type, which sets what jump measures: a shift in the mean, a rise in the standard deviation, both, or a trend, as [cpt_power\(\)](#) defines them), power, method and constant (TRUE when the series has no variation, in which case nothing is simulated). With print() and tidy() methods. A formula fit is refused: the simulation has no covariates to draw.

See Also

Other inference: [cpt_assumptions\(\)](#), [cpt_attribute_event\(\)](#), [cpt_effect\(\)](#), [cpt_gof\(\)](#), [cpt_robustness\(\)](#), [cpt_test_at\(\)](#), [cpt_test_null\(\)](#)

Examples

```
set.seed(1)
fit <- cpt_detect(rnorm(100), method = "pelt")
cpt_null_power(fit, n_sim = 20, seed = 1)
```

 cpt_null_sizes

False alarms on pure noise at scale

Description

Independent Gaussian noise with no changepoint anywhere, at 1,000 (50 replicates), 10,000 (10) and 100,000 (6) observations, for the engines the runtime table shows finishing that length quickly. Every changepoint reported is a false alarm.

Usage

```
cpt_null_sizes
```

Format

A tibble with one row per engine and length:

method the engine.

n the series length.

reps replicates that ran.

size share of replicates reporting at least one changepoint.

mean_fp mean number of changepoints reported.

Source

data-raw/measure_engines.R (part null).

See Also

Other measurement tables: [cpt_calibration](#), [cpt_data_types](#), [cpt_invariances](#), [cpt_noise_benchmark](#), [cpt_runtimes](#)

Examples

```
subset(cpt_null_sizes, n == max(n))
```

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 **change**point-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,
    family = "gaussian",
    baseline = NULL,
    ...
)

## 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. For a non-Gaussian family it is the change in the family's own parameter, added to baseline: a rate for "poisson", a probability for "binomial" and a mean waiting time for "exponential".
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 $\sigma * (1 + \text{jump})$; "meanvar" does both at once, shifting the mean by jump and the standard deviation to $\sigma * (1 + \text{jump} / 2)$, the same design cpt_scenarios() simulates; "slope" bends a flat line into one rising by $\text{jump} * \sigma$ 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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
parallel	Use <code>future::plan()</code> 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.
family	"gaussian" (the default), or "poisson", "binomial" or "exponential": the replicates are drawn from that family (see <code>cpt_simulate()</code>) and detected with that family's cost (<code>cpt_detect(family =)</code>), so the method must fit it (<code>cpt_families()</code>). <code>sigma</code> , <code>noise</code> , <code>rho</code> and <code>df</code> do not apply.
baseline	The family's parameter before the change, for a non-Gaussian family: a rate (default 5), a probability (0.3) or a mean waiting time (1).
...	Additional arguments passed to <code>cpt_detect()</code> .
x	A <code>ggcpt_power</code> object.
object	A <code>ggcpt_power</code> object (for <code>autoplot()</code>).

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* change-points 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 and the package's measurements into an answer. Given what the analyst knows about their problem (how many dimensions, what kind of change, what the noise and the data look like, how long the series is, how many changes they expect, whether they need uncertainty or an online alarm), this returns the methods that fit, as calls, with the power and false-alarm rate measured for each in that situation, a reason and the caveats. It is a decision table, not a model: everything it knows is in [cpt_methods\(\)](#) and the measurement tables ([cpt_noise_benchmark](#), [cpt_data_types](#), [cpt_runtimes](#)), 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,
  n_expected = NULL,
  data_type = c("continuous", "counts", "binary", "proportion"),
  family = NULL,
  fit = NULL,
  jump = NULL
)
```

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

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

```
## S3 method for class 'ggcpt_recommendation'
autoplot(object, top = 15, ...)
```

Arguments

dimension	"univariate" (default) or "multivariate".
change_in	What kind of change is expected: any value accepted by <code>cpt_detect()</code> . Defaults to "mean".
noise	Noise structure: "iid" (default), "heavy" (heavy-tailed), "autocorrelated", or "heteroscedastic".
n	Series length. Methods the runtime table saw fail to finish at that length are left out (and listed), and the slow ones carry a caveat. 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.
n_expected	How many changes you expect, if you know. The single-change designs (amoc, pettitt, buishand, snht) cannot find a second one, so they are ranked down unless this is 1.
data_type	What the values are: "continuous" (default), "counts", "binary" or "proportion". Measured, a Gaussian cost raises about 165 times the false positives on binary data that it raises on Gaussian data, so engines are scored on their measured false positives for this data type, and those that can fit a cost for it (family = "poisson" or "binomial") are preferred and recommended with that argument.
family	Optional distribution family the method must fit (see <code>cpt_families()</code>).
fit	Optional ggcpt fit already made. Its dimension, length and change type fill in the arguments not given; its values set the data type; and when its residuals are autocorrelated (see <code>cpt_assumptions()</code>) the noise is taken as "autocorrelated" unless noise was given.
jump	Optional expected change size, in noise standard deviations. With n, the top three candidates' power to find it is simulated (<code>cpt_power()</code> with 50 replicates), and the printout says so when no method is likely to.
x	A ggcpt_recommendation object (for print()).
...	Ignored.
top	How many candidates to print. Defaults to 5.
object	A ggcpt_recommendation object (for autoplot()).

Value

A ggcpt_recommendation tibble ordered by suitability, with columns method, engine, installed, score, tie (candidates with the same score share a number: the recommender cannot tell them apart on what was supplied), call (the `cpt_detect()` call to run, including the noise-model or family argument the recommendation depends on), hits and false_positives (measured in the stated regime: mean real changes found out of two, and mean spurious ones), power (when jump is given), why and caveat. `print()` reads as advice and `autoplot()` shows the scores and the ties.

How the score is built

Where the method was measured in the stated noise regime, the score starts from $3 - E$, where E is the expected number of errors per series in that benchmark (real changes missed plus false alarms), taking the method's best setting of its noise-model argument (which is then the one in call). A method not measured there starts at 1. Then: minus 3 for a single-change design unless $n_expected = 1$; minus the log of one plus its measured false positives on the stated data type, and plus 1.5 if it fits that data type's family; plus 1 for a location interval and 0.5 for a posterior when uncertainty is needed; at $n \geq 5000$, plus 0.5 for a fast engine and minus 1 for a slow one; and small tie-breakers for the general-purpose methods (0.25) and for an installed engine (0.1). Sort by `false_positives` instead if a false alarm costs you more than a miss.

See Also

[cpt_methods\(\)](#), [cpt_consensus\(\)](#), [cpt_cite\(\)](#).

Examples

```
cpt_recommend(noise = "autocorrelated")
cpt_recommend(data_type = "counts", n_expected = 2)
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	<i>Register an external changepoint detector</i>
---------------------	--

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 <code>fn(x, ...)</code> . 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.

<code>change_in</code>	Character vector of <code>change_in</code> values the detector supports. Defaults to "mean".
<code>engine</code>	Name of the package or system supplying the detector, for display in <code>cpt_methods()</code> . Defaults to "user".
<code>citation</code>	Optional citation string returned by <code>cpt_cite()</code> . When NULL, <code>cpt_cite()</code> says plainly that the registration supplied none rather than inventing one.
<code>capabilities</code>	Optional named list of capability flags overriding the defaults (all FALSE): <code>multivariate</code> , <code>online</code> , <code>ci</code> , <code>fitted</code> , <code>posterior</code> , <code>statistic</code> , <code>path</code> , <code>scale_space</code> .
<code>cp_convention</code>	"left" (the changepoint is the last index of the left segment, this package's convention) or "right". Used when <code>fn</code> returns bare indices.
<code>overwrite</code>	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 <code>min(100, floor(n / 4))</code> .
...	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", "json"),
  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), "gt" (a gt table of the changepoints, for a publication) or "json" (the whole result under as_json() 's versioned schema, for a pipeline rather than a reader).
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_robustness	<i>Does a changepoint survive a change of noise model?</i>
----------------	--

Description

Re-runs a detector under each setting of its noise-model argument (family for **stepR**'s SMUCE, variant for NSP, the cost for the **changepoint** engines, ...) and reports which changepoints appear under every setting and which only under some. A changepoint that survives a change of noise model is evidence; one that survives a bootstrap ([cpt_stability\(\)](#)) or a change of algorithm ([cpt_consensus\(\)](#)) may only be a feature of the noise, which resampling and other algorithms reproduce.

Usage

```
cpt_robustness(
  x,
  method = "pelt",
  over = "noise_model",
  settings = NULL,
  margin = 5,
  seed = NULL,
  ...
)

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

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

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

Arguments

x	A numeric series, or a ggcpt fit (its series, method and penalty are used).
method	The detector, when x is a series.
over	What to vary. Only "noise_model" for now.
settings	Optional named list of settings, each a list of arguments for cpt_detect() , replacing the method's own sweep. The first is the reference.
margin	Positions within which two changepoints count as the same. Defaults to 5.

seed	Optional seed, scoped to this call.
...	Further arguments for every <code>cpt_detect()</code> call.
object	A <code>ggcpt_robustness</code> object (for <code>autoplot()</code>).

Value

A `ggcpt_robustness` object: a list with settings (one row per setting: `setting`, `n_cp`, `powered` and `error`), `changepoints` (one row per changepoint of the reference setting: `cp`, `found_by` (how many other powered settings find it within margin), `of` (how many powered alternatives there are) and `survives`), `union` and `intersection` (distinct locations found by any and by every powered setting) and `fits`. With `print()` and `autoplot()`.

Powerless settings are reported, not counted

A setting that finds nothing where the reference finds several is not a vote against them: measured, NSP's `variant = "ar"` removes the false positives of autocorrelated noise by removing detection altogether (0 of 2 real changes found). Such a setting is marked `powered = FALSE` and left out of `found_by`.

See Also

`cpt_assumptions()`, `cpt_sensitivity()` for sweeping any argument.

Other inference: `cpt_assumptions()`, `cpt_attribute_event()`, `cpt_effect()`, `cpt_gof()`, `cpt_null_power()`, `cpt_test_at()`, `cpt_test_null()`

Examples

```
set.seed(1)
x <- c(rnorm(100), rnorm(100, 2)) + as.numeric(arima.sim(list(ar = 0.6), 200))
cpt_robustness(x, method = "pelt")
```

`cpt_runtimes`

Engine runtimes by series length

Description

One timed `cpt_detect()` call per engine and length, each in its own R process running an installed (byte-compiled) build, after a warm-up call, so the time is the fit and not loading or compiling anything, with a time cap of 120 seconds (300 at a million). A length is attempted only when the previous one finished in under ten seconds. One run per cell: this answers "will it finish?", and is not a basis for claiming one engine is faster than another by a given factor.

Usage

`cpt_runtimes`

Format

A tibble with one row per engine and length attempted:

method the method, as `cpt_detect()` takes it.

n the series length: 1,000, 10,000, 100,000 or 1,000,000, with four mean changes of two noise standard deviations (three coordinates for the multivariate engines).

seconds elapsed time of the fit.

k changepoints reported (four are real; a single-change method reports one).

status "ok", "timeout" or "error".

Source

data-raw/measure_engines.R (part runtime).

See Also

`cpt_methods()`'s cost and max_n columns, `cpt_recommend()`.

Other measurement tables: `cpt_calibration`, `cpt_data_types`, `cpt_invariances`, `cpt_noise_benchmark`, `cpt_null_sizes`

Examples

```
subset(cpt_runtimes, n == 1e5 & status == "ok")
```

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

<code>x</code>	A numeric vector (or, for <code>method = "npmojo"</code> , a matrix with rows as time points), or any <code>ggcpt</code> 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 <code>mosum</code> or <code>npmojo</code> result the engine is adopted as well (see <code>method</code>), and from anything else the default <code>"mosum"</code> sweep is drawn over the same series.
<code>bandwidths</code>	Integer vector of bandwidths to sweep. Defaults to a geometric grid between $\max(5, n/50)$ and $n/4$.
<code>method</code>	Which engine to sweep: <code>"mosum"</code> (default) or <code>"npmojo"</code> . Taken from <code>x</code> when it is a <code>ggcpt</code> produced by one of them.
<code>...</code>	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

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

Description

Detection says where the regimes change; this fits *your* model to each regime. Every segment of fit gets its own copy of model, fitted by engine to that segment's rows, and the result is a table with one fitted model per segment that `tidy()`, `glance()` and `predict()` read.

Usage

```
cpt_segment_models(fit, model = NULL, data = NULL, engine = stats::lm, ...)

## S3 method for class 'ggcpt_segment_models'
tidy(x, conf_level = 0.95, ...)

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

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

Arguments

<code>fit</code>	A <code>ggcpt</code> object.
<code>model</code>	A formula. For a result from a formula fit it defaults to that formula; for a series it defaults to <code>value ~ 1</code> (the segment level), or <code>value ~ time</code> for a change in slope. The series is available as <code>value</code> , its position as <code>time</code> , and the time index (when the result carries one) as <code>index</code> .
<code>data</code>	Optional data frame with one row per observation, supplying covariates the model uses. For a formula fit the covariates are taken from the result.
<code>engine</code>	The fitting function, called as <code>engine(model, data = segment_rows, ...)</code> . Defaults to <code>stats::lm</code> ; <code>stats::glm</code> with a family works the same way.
<code>...</code>	Further arguments for engine.
<code>x</code>	A <code>ggcpt_segment_models</code> object.
<code>conf_level</code>	Confidence level for the coefficient intervals.

Value

A `ggcpt_segment_models` tibble with one row per segment: `segment`, `start`, `end`, `n` and `model` (a list-column of fitted models, `NULL` where the fit failed). `tidy()` gives the coefficient table with a `segment` column; `glance()` one row per segment with `n`, `sigma`, `r_squared`, `AIC`, `BIC` and `logLik`.

See Also

`predict.ggcpt()` to forecast from a segment, `tidy.ggcpt(what = "coefficients")` for the coefficients the detection itself implies.

Other segment models: `predict.ggcpt()`

Examples

```
set.seed(1)
d <- data.frame(x = runif(200))
d$y <- ifelse(seq_len(200) <= 100, 1 + 2 * d$x, 3 - d$x) + rnorm(200, 0, 0.3)
fit <- cpt_detect(y ~ x, data = d, method = "strucchange")
sm <- cpt_segment_models(fit)
tidy(sm)
glance(sm)
```

cpt_select

Choose the number of changepoints

Description

Builds a ladder of candidate segmentations with $K = 0, 1, \dots$ 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, $n \log(\text{RSS}/n) + (2K + 1) \log n$. 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 <code>cpt_penalty()</code> states it. It depends on the segment lengths l_i and so cannot be expressed by <code>cpt_penalty()</code>'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, $n \log(\text{RSS}/n) + 2(2K + 1)$: 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. An elbow needs $K \geq 1$, so this criterion cannot return $K = 0$. "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 <i>mean</i>), 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. It cannot return $K = 0$. "bic", "mbic" and "cv" can return $K = 0$; "aic" in principle can and in practice (measured at $n = 400$) picked <code>k_max</code> or next to it at every true K. <code>cpt_select()</code> warns (class <code>ggchangepoint_selection_unadjusted</code>) when "aic" lands at the top of the ladder, and when "crops_elbow" or "stability" returns its smallest candidate.
<code>k_max</code>	Largest number of changepoints considered. Defaults to 20, capped at $\text{floor}(n / 4)$.
<code>folds</code>	Folds for <code>criterion = "cv"</code> . Defaults to 5; 2 gives the original COPPS split.
<code>B</code>	Bootstrap replicates for <code>criterion = "stability"</code> . Defaults to 100.
<code>change_in</code>	Passed to the detector, and inherited from <code>x</code> when <code>x</code> is a <code>ggcpt</code> (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 $n \log(\text{RSS}/n)$ (the deviance for a change in mean with a common variance), so on a <code>change_in = "var"</code> ladder the <i>candidates</i> come from the variance detector while the <i>score</i> 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 <code>change_in</code> and so carries no such assumption. "cv" does not help: its estimator is a least-squares change in the mean as well.
<code>index</code>	Optional time index (a vector of dates, or a <code>ts</code> , <code>xts</code> , <code>zoo</code> or <code>tsibble</code> passed as <code>x</code>), carried onto the chosen fit so <code>tidy()</code> and <code>autoplot()</code> report the changepoint on your scale rather than as a position. Inherited from <code>x</code> when <code>x</code> is an indexed <code>ggcpt</code> .
<code>seed</code>	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	Additional arguments passed to <code>cpt_detect()</code> when the ladder is built by repeated detection.
<code>object</code>	A <code>ggcpt_selection</code> object (for <code>autoplot()</code>).
<code>plot_type</code>	"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).
<code>max_facets</code>	Maximum number of rungs drawn by <code>plot_type = "ladder"</code> . 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	<i>Sensitivity of a segmentation to its tuning parameters</i>
-----------------	---

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: <code>list(penalty = c(5, 10, 20), minseglen = c(2, 10))</code> .
seed	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> 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 <code>cpt_detect()</code> .
object	A ggcpt_sensitivity object (for <code>autoplot()</code>).

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,
    family = c("gaussian", "poisson", "binomial", "exponential")
)

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 $\text{intercept}_i + \text{slope}_i \cdot (1, \dots, l_i)$ 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, <code>list(list(intercept = 0, slope = 1), list(intercept = 100, slope = -1))</code> 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. <code>changepoints</code> 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 <code>bfast_wrapper()</code> 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 <code>change_in = "var"</code> , which is piecewise constant: this is the <i>gradual</i> 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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
family	The distribution of the observations: "gaussian" (the default, signal plus noise), "poisson" (counts whose rate is the segment's parameter), "binomial" (0/1 outcomes whose probability is the segment's parameter) or "exponential" (waiting times whose mean is the segment's parameter). The last three generate a change in that one parameter, so they take <code>change_in = "mean"</code> , one number per segment in <code>params</code> (defaults: rate 5, probability 0.5, mean 1), and no noise, <code>sd</code> , <code>seasonality</code> or <code>sd_trend</code> . A rate change in counts is a different detection problem from a Gaussian mean shift, which is why the power curves are not interchangeable.
...	Passed to <code>cpt_simulate</code> .

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)

# counts whose rate doubles
```

```
counts <- cpt_simulate(200, changepoints = 100, params = c(4, 8),
                      family = "poisson", seed = 1)
```

cpt_solution_path	<i>The solution path of a search-based detector</i>
-------------------	---

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, $|\max.\text{contrast}|$ 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,
  bootstrap = c("iid", "block"),
  block_length = NULL,
  reversal = TRUE,
  ...
)

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

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

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

Arguments

<code>x</code>	A numeric vector, or a <code>ggcpt</code> fit (its series, method, change type and penalty are used, and it is the original the replicates are compared with); for the <code>print()</code> method, a <code>ggcpt_stability</code> object.
<code>method</code>	Detection method, passed to <code>cpt_detect()</code> . Ignored when <code>x</code> is a fit.
<code>B</code>	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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
bootstrap	"iid" (the default) resamples residuals one at a time within each segment; "block" resamples contiguous blocks of them, which keeps the residuals' short-range dependence, so an artifact of autocorrelated noise can move between replicates instead of being rebuilt in the same place every time.
block_length	Block length for <code>bootstrap = "block"</code> . Defaults to $n^{1/3}$ lengthened by the residuals' lag-1 autocorrelation r by the factor $(1 + r)/(1 - r)$, and capped at a quarter of the series.
reversal	Also run the detector once on the reversed series, and flag each changepoint that the reversed series does not find within margin? Defaults to TRUE. An offline detector looks at the whole series, so a changepoint that does not survive reversal is a borderline one: measured, it is the same instability the bootstrap finds, for one extra fit. Sequential (online) detectors genuinely depend on direction and get NA.
...	Additional arguments passed to every <code>cpt_detect()</code> call.
object	A <code>ggcpt_stability</code> object (for <code>autoplot()</code>).

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`), `B`, `n_failed` (replicates on which the detector failed, which are left out of the proportion rather than counted as "found nothing"), `bootstrap`, `block_length` and `reversal` (one row per original changepoint: `cp` and survives). Methods: `print()`, `tidy()` (one row per original changepoint: `cp`, `stability` and, when reversal ran, `survives_reversal`) and `autoplot()` (frequency profile with the original detections marked).

What a high score means

Reproducible under resampling, which is not the same as *real*, and under dependent noise the two diverge: measured under AR(1) noise, a spurious changepoint reached a stability of 1.00 and the score separated real from spurious changepoints in only 4 of 10 replicates. The artifact is a feature of the realised noise path, and an i.i.d. resample of the residuals rebuilds it every time. `bootstrap = "block"` lets it vary; `cpt_robustness()`, which changes the noise model rather than the sample, is the check that touches the cause; and `cpt_assumptions()` tests the residuals directly.

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	<i>The detector's statistic as a function of location</i>
---------------	---

Description

Returns the criterion the engine evaluated at each position, which is what explains *why* a change-point 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	<i>Test detected changepoints</i>
----------	-----------------------------------

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",
  relevance = NULL
)
```

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.
relevance	Optional smallest change worth acting on, in noise standard deviations. The null hypothesis becomes "the change is no larger than relevance" rather than "there is no change", so a statistically detectable but practically negligible shift is not significant. Applies to the Welch route (the changepoints of a series fit); the p_value column is then the relevance test's, and a relevance column records the threshold in the data's units.

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. The exception is a changepoint that *was* fixed in advance, through `cpt_detect(fixed = )`: its row is TRUE.

- The fallback compares means whatever the fit detected, so on a fit of a change in variance, slope or distribution it tests a different change from the one found, and warns (class `ggchangept_assumption`).
- 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)
```

<code>cpt_test_at</code>	<i>Test for a change at a date fixed in advance</i>
--------------------------	---

Description

The question behind most applied changepoint work: the policy took effect on 1 March, the plant was retooled in week 14, the drug was approved in Q2. *Did anything change then?* Because the date was chosen before looking at the data, this is an ordinary two-sample comparison at a fixed split and needs no adjustment for selection: the p-value means what it says.

Usage

```
cpt_test_at(
  x,
  when,
  window = 0,
  change_in = c("mean", "var", "meanvar", "distribution"),
  family = c("gaussian", "poisson", "binomial", "exponential", "l1"),
  span = NULL,
  index = NULL,
```

```

    data = NULL,
    level = 0.95,
    B = 999,
    seed = NULL
  )

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

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

```

Arguments

x	A numeric series (a vector, ts, zoo and so on), a ggcpt fit (its series and index are used), or a formula with data for a break in a regression (a Chow test).
when	When the change took effect: the first observation of the new regime , as a position or a value of the series' index (a date; for a numeric index such as a ts's years, a number outside 1..n is read as an index value). A detected changepoint cp (the last observation of the old regime) corresponds to when = cp + 1.
window	Allow the change anywhere within window observations either side of when. The statistic is then the largest over the window and its p-value comes from a permutation distribution of that maximum, which pays for the search. Defaults to 0: the date is exact.
change_in	What to test: "mean" (the default), "var", "meanvar" or "distribution".
family	For change_in = "mean": "gaussian" (a Welch t-test, the default), "poisson" (an exact test of two rates), "binomial" (Fisher's exact test of two proportions), "exponential" (an exact F test of two rates) or "l1" (a Wilcoxon rank-sum test, robust to outliers).
span	Optional number of observations each side of the split to compare. Defaults to all of them; a span keeps a change elsewhere in a long series out of the comparison.
index	Optional time index when x is a bare series.
data	A data frame, for formula input.
level	Confidence level for the interval on the change.
B	Permutations for window > 0. Defaults to 999.
seed	Optional seed for the permutations, scoped to this call.
...	Ignored.

Value

A ggcpt_test_at tibble with one row: when, cp (the last observation before it, in the package's convention) and cp_index, window, estimate (the change: a difference in means, a ratio of rates or variances), conf_low, conf_high, statistic, p_value, n_before, n_after, method and selection_adjusted (TRUE: the location was not chosen from the data, and a window's search is paid for by the permutation).

See Also

`cpt_attribute_event()` for the mirror question (was a detected change the event you know about?), `cpt_effect()`.
Other inference: `cpt_assumptions()`, `cpt_attribute_event()`, `cpt_effect()`, `cpt_gof()`, `cpt_null_power()`, `cpt_robustness()`, `cpt_test_null()`

Examples

```
set.seed(1)
dates <- as.Date("2026-01-01") + 0:119
x <- c(rnorm(60), rnorm(60, 0.8))
cpt_test_at(x, when = as.Date("2026-03-02"), index = dates)
# the policy took effect sometime in that fortnight
cpt_test_at(x, when = 61, window = 7, seed = 1)
```

<code>cpt_test_null</code>	<i>Test for a change anywhere in a series</i>
----------------------------	---

Description

A test of the global null hypothesis "the series has no changepoint", the question an empty detection raises. Nothing is located first, so nothing is selected, and the p-value means what it says.

Usage

```
cpt_test_null(x, method = c("cusum", "pettitt", "supF"), index = NULL)
```

Arguments

<code>x</code>	A numeric series, or a <code>ggcpt</code> fit (its series is used).
<code>method</code>	Which test: "cusum" (the default) the CUSUM test for a change in mean: the largest standardised cumulative-sum deviation, referred to the Kolmogorov distribution of the supremum of a Brownian bridge. The noise scale is estimated from first differences, which a change barely affects. "pettitt" Pettitt's rank test, distribution-free, for a shift in location (approximate p-value). "supF" Andrews' sup-F structural change test for a change in mean, with 15% trimming, through strchange .
<code>index</code>	Optional time index for a bare series.

Value

A tibble with one row: `method`, `statistic`, `p_value`, `location` (where the statistic peaks: the most likely changepoint, reported for orientation only), `n` and `selection_adjusted` (TRUE).

Assumptions

All three assume independent observations. Under positive autocorrelation they reject too often, which is the same failure the detectors share; see [cpt_assumptions\(\)](#).

See Also

[cpt_null_power\(\)](#) for what an empty answer could have detected, [cpt_test_at\(\)](#) for a pre-specified date.

Other inference: [cpt_assumptions\(\)](#), [cpt_attribute_event\(\)](#), [cpt_effect\(\)](#), [cpt_gof\(\)](#), [cpt_null_power\(\)](#), [cpt_robustness\(\)](#), [cpt_test_at\(\)](#)

Examples

```
set.seed(1)
cpt_test_null(rnorm(200))
shifted <- c(rnorm(100), rnorm(100, 0.6))
cpt_test_null(shifted, method = "pettitt")
cpt_test_null(shifted, method = "supF")
```

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 <code>ggcpt_monitor</code> 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_verify

Re-run a result against the engines installed now

Description

A result records which version of its engine produced it (see `glance()`'s `engine_version`). `cpt_verify()` answers the question that record exists for: *does the answer still hold?* It re-runs the same method on the same series with the settings the result recorded, using whatever engines are installed today, and reports whether the changepoints moved and which versions changed in between.

Usage

```
cpt_verify(object, ..., tolerance = 0)

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

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

Arguments

<code>object</code>	A <code>ggcpt</code> object from <code>cpt_detect()</code> .
<code>...</code>	Further arguments for the re-run, overriding what was recovered from the result (for instance an <code>engine</code> argument the recorded call held as a variable, which cannot be recovered from the object).
<code>tolerance</code>	Positions a changepoint may move and still count as the same. Defaults to 0: an exact match.
<code>x</code>	A <code>ggcpt_verification</code> object.

Details

Upstream engines change: a default is revised, a bug is fixed, a tie-breaking rule moves. None of that raises an error, so a result produced last year and re-run today can differ without anything saying so. This is the check to run when re-opening an analysis, and before quoting an old result as current.

The re-run uses the series stored on the result, its method, the change type it can be asked for, its penalty and family, the missing-value handling and constraints it recorded (`na_action`, `fixed`,

within, min_segment, min_effect), and every engine argument of the original call that was written as a literal (seed = 42, n_intervals = 500). An argument written as a variable cannot be recovered from the object, and is listed in not_recovered; pass it again through ... if it mattered. A result read back with `cpt_import()` from JSON keeps the call it was written with, so it replays the same way; one made by anything but `cpt_detect()` or a wrapper (`as_ggcpt()`, the `cpt_select()` returns) is refused, since re-running `cpt_detect()` would not reproduce how it was made.

Value

A `ggcpt_verification` object: a list with `verified` (TRUE when every changepoint reproduces within tolerance), `then` and `now` (the two sets of locations), `added` and `removed` (locations in one set with no partner in the other), `versions` (a tibble with component, then, now and changed), `not_recovered` (call arguments that could not be replayed) and `refit`, the new `ggcpt`. With a `print()` method.

See Also

`glance.ggcpt()` for the recorded `engine_version`.

Other result class: `annotate_segments()`, `as_cpt_series()`, `as_ggcpt()`, `as_json()`, `cpt_annotations()`, `cpt_export()`, `ggchangepoint-conditions`, `is_ggcpt()`, `new_ggcpt()`, `print.ggcpt()`

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(80), rnorm(80, 4)), method = "pelt")
cpt_verify(fit)
```

`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

<code>data</code>	A numeric vector.
<code>change_in</code>	Choice of <code>mean_var</code> , <code>mean</code> , <code>var</code> , and <code>np</code> (or <code>cpt_np</code> for backward compatibility). Each choice corresponds to <code>cpt.meanvar()</code> , <code>cpt.mean()</code> , <code>cpt.var()</code> and <code>cpt.np()</code> respectively. The default is <code>mean_var</code> .
<code>cp_method</code>	A wide range of choices (i.e., AMOC, PELT, SegNeigh or BinSeg). Please note when <code>change_in</code> is <code>np</code> or <code>cpt_np</code> , PELT is the only option.
<code>...</code>	Extra arguments for each <code>cpt</code> function mentioned in the <code>change_in</code> section. For a change in mean (and for <code>np</code>) <code>minseglen</code> defaults to 2 rather than the engine's 1, so no segment is a single observation; pass it to choose another value. SegNeigh keeps the engine's default, which is the only one it implements.

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()`, `idetector_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, Rigaiil, 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

<code>x</code>	A numeric vector.
<code>penalty</code>	Penalty β for adding a changepoint. Defaults to $2 * \log(\text{length}(x))$. 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 <code>penalty</code> explicitly when the two must agree.
<code>model_param</code>	Optional list of model parameters (<code>sdEta</code> , <code>sdNu</code> , <code>phi</code>) as accepted by <code>DeCAFS::DeCAFS()</code> ; when <code>NULL</code> they are estimated from the data.
<code>...</code>	Additional arguments passed to <code>DeCAFS::DeCAFS()</code> .

Value

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

References

Romano G, Rigaiil 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\(\)](#), [idetector_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.agгло()`. 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

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

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.agгло()`'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()`, `idetector_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

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

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()`, `idetector_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

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

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](https://doi.org/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()`, `idetector_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	<i>Network-structure changepoints via non-negative matrix factorisation</i>
--------------------	---

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

<code>x</code>	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.
<code>min_dist</code>	Minimum distance between changepoints. Defaults to 35 (the engine's default), lowered automatically when the series is too short for it.
<code>n_runs</code>	NMF runs per candidate split. Defaults to 50.
<code>n_reps</code>	Replicates on each side of the significance test: the split is refitted <code>n_reps</code> times and the rows permuted <code>n_reps</code> times. Defaults to 100; at least 2, because the test compares two samples of this size.
<code>alpha</code>	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 <code>testtype</code> 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 (<code>testtype = "wilcox"</code> or <code>"ks"</code>) do have one, and the wrapper warns when it is above alpha.
<code>rank</code>	NMF rank, a positive whole number. NULL estimates it with <code>fabisearch::opt.rank()</code> , which is expensive; supplying a rank is much faster.
<code>n_core</code>	Cores for the permutation stage. Defaults to 1.

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

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()`, `idetector_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. Every family the engine documents is reachable except its user-supplied custom cost: Gaussian mean, variance and both; the count and binary families ("poisson", "binomial") and waiting times ("exponential"); linear and penalised regression ("lm", "lasso") and generalised linear models ("poisson", "binomial" with covariates); and the time-series models "ar", "arma", "arima", "garch" and, for a multivariate series, "var".

Usage

```
fastcpd_wrapper(
  x,
  family = c("mean", "variance", "meanvariance", "ar", "arma", "arima", "garch", "var",
    "lm", "lasso", "poisson", "binomial", "exponential"),
  order = NULL,
  covariates = NULL,
  ...
)
```

Arguments

x A numeric vector, or (for family "mean", "variance", "meanvariance" and "var") a matrix with one row per time point for multivariate detection.

family	Model family. "mean" (the default), "variance", "meanvariance"; "poisson", "binomial", "exponential" (a change in the rate or probability of a count, binary or waiting-time series, or in a regression on covariates); "lm", "lasso" (regression, which needs covariates); "ar", "arma", "arima", "garch", "var".
order	Model order for "ar" and "var" (a single integer), "arma" and "garch" (length 2) or "arima" (length 3). Defaults to 1 for AR and VAR, c(1, 1) for ARMA and GARCH and c(1, 0, 0) for ARIMA.
covariates	Optional numeric matrix of regressors, one row per observation, for "lm", "lasso", "poisson" and "binomial". Include a column of ones for an intercept. Without it the count and binary families fit an intercept only: a change in the rate or probability. <code>cpt_detect()</code> builds it from a formula.
...	Additional arguments passed to the corresponding <code>fastcpd::fastcpd.*()</code> function (e.g. <code>beta</code> , <code>trim</code>). <code>beta</code> is fastcpd 's penalty: a number, or one of its own names ("MBIC", the default, "BIC", "MDL"). It is recorded on the result, so <code>print()</code> and <code>glance()</code> report the penalty the fit actually used. <code>cpt_detect()</code> forwards a numeric penalty here and translates the three names it shares with fastcpd ; see the penalty-semantics section of <code>cpt_penalty()</code> .

Value

A `ggcpt` object. The engine's per-segment parameter estimates (thetas) are kept as `$coefficients`, one row per segment and parameter, for the families where they are coefficients (the regressions and the intercept-only count, binary and waiting-time models).

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()`, `idetector_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

# A change in a Poisson rate
counts <- fastcpd_wrapper(c(rpois(100, 3), rpois(100, 9)),
                          family = "poisson")
counts$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

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

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()`, `idetector_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

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

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](https://doi.org/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()`, `idetector_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

<code>x</code>	A numeric vector.
<code>penalty</code>	Penalty value. Defaults to $2 * \log(\text{length}(x))$ (BIC). This is an <i>absolute</i> 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 $2 * \log(\text{length}(x)) * \text{stats::var}(\text{diff}(x)) / 2$). 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 <code>penalty</code> is given.
<code>...</code>	Additional arguments passed to <code>fpop::Fpop()</code> .

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, Rigai G, Fearnhead P (2017). "On Optimal Multiple Changepoint Algorithms for Large Data." *Statistics and Computing*, **27**(2), 519–533. doi:[10.1007/s1122201696363](https://doi.org/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\(\)](#), [idetector_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
```

geom_changepoint	<i>Changepoint vertical rules geom</i>
------------------	--

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

<code>mapping</code>	Set of aesthetic mappings created by <code>ggplot2::aes()</code> . Requires <code>xintercept</code> .
<code>data</code>	A data frame with changepoint information.
<code>...</code>	Other arguments passed to <code>geom_vline</code> .
<code>na.rm</code>	If <code>FALSE</code> , missing values are removed.
<code>show.legend</code>	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\(\)](#), [ggchangepoint-ggproto](#), [stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)

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 change-point 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\(\)](#), [ggchangepoint-ggproto](#), [stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)

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	<i>Event annotation geom</i>
----------------	------------------------------

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 <code>xintercept</code> for the rule and <code>label</code> for the text.
data	A data frame of events (see <code>cpt_annotate_events()</code>).
...	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 <code>y</code> to be mapped as well (<code>ggrepel</code> 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' <code>x/y</code> 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\(\)](#), [ggchangepoint-ggproto](#), [stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)

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. <code>cpt_labels()</code> output.
...	Other arguments passed to <code>ggplot2::geom_rect()</code> .
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()`, `ggchangepoint-ggproto`, `stat_changepoint()`, `stat_cpt_region()`

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. <code>cpt_regions()</code> output.
...	Other arguments passed to <code>ggplot2::geom_rect()</code> .
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()`, `ggchangepoint-ggproto`, `stat_changepoint()`, `stat_cpt_region()`

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	<i>Changepoint segment level geom</i>
------------------	---------------------------------------

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

<code>mapping</code>	Aesthetic mappings. Requires <code>x</code> , <code>xend</code> , <code>y</code> , <code>yend</code> .
<code>data</code>	A data frame with segment information.
<code>...</code>	Other arguments passed to <code>geom_segment</code> .
<code>na.rm</code>	If <code>FALSE</code> , missing values are removed.
<code>show.legend</code>	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\(\)](#), [ggchangepoint-ggproto](#), [stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)

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)
```

geomcp_wrapper	<i>Geometrically-inspired multivariate changepoint wrapper (geomcp)</i>
----------------	---

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

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

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\(\)](#), [idetec_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
```

ggchangepoint-conditions

Conditions raised by ggchangepoint

Description

Every error, warning and message this package raises carries a class, so code that calls it can react to the *kind* of failure rather than matching its wording. The text is written for a person and may be improved between releases; the classes and the fields listed here are the interface, and follow the package's deprecation policy.

Errors

Every error inherits from `ggchangepoint_error` (and from `error` and `condition`). The subclasses, most specific first:

`ggchangepoint_input_error` The call itself is not usable: a bad argument or data that violates a precondition. Its subclasses say which precondition: `ggchangepoint_short_series` (too few observations, for the package or for the engine that was asked; field `n`), `ggchangepoint_wrong_dimension` (a univariate method handed a matrix, or the reverse), `ggchangepoint_non_finite` (NA, NaN or Inf where the method needs finite data), `ggchangepoint_bad_type` (a factor, text, a survival object or timestamps where a numeric series was expected) and `ggchangepoint_bad_argument` (an argument value outside its documented range or vocabulary).

`ggchangepoint_unsupported` The request is well formed but not something this method offers: a `change_in`, family or capability the registry does not list for it. Fields `method`, `requested` and `supported` where they apply. Subclasses: `ggchangepoint_capability_absent` (the result carries no interval, statistic, path or posterior to read), `ggchangepoint_planned_method` (named in `cpt_methods()` but not wired) and `ggchangepoint_unknown_method`.

`ggchangepoint_engine_missing` A suggested engine package is not installed. Field `package`, and `install`, the call that installs it.

`ggchangepoint_engine_error` The upstream engine failed. Fields `engine`, `method` and, where it was caught, `parent`, the engine's own condition.

`ggchangepoint_upstream_bug` A known, version-guarded defect in an engine, refused rather than returned. Fields `package` and `version`.

`ggchangepoint_internal_error` A state the package should never reach: a bug here, worth reporting.

The functions that run many fits (`cpt_benchmark()`, `cpt_consensus()`) record a failure that belongs to one method (`ggchangepoint_engine_error`, `ggchangepoint_engine_missing`, `ggchangepoint_unsupported`, `ggchangepoint_upstream_bug`, or a series of the wrong shape or length for that method) and carry on; any other error stops them, since every method would have hit it.

Warnings

Every warning inherits from `ggchangepoint_warning`. The ones worth handling by class: `ggchangepoint_assumption` and its subclasses `ggchangepoint_scale_sensitive`, `ggchangepoint_data_type` and `ggchangepoint_dependence` (the method's assumptions look violated for this series); `ggchangepoint_degenerate_segmentation` (a changepoint after every observation); `ggchangepoint_implausible_count` (more changepoints than the series can plausibly hold); `ggchangepoint_change_in_routed` (the request was answered with the method's native change type); `ggchangepoint_argument_ignored`; `ggchangepoint_level_not_applied`; `ggchangepoint_selection_unadjusted` (p-values computed at data-chosen locations); `ggchangepoint_collapsed_ladd`; `ggchangepoint_replicates_failed`; `ggchangepoint_cp_dropped` (locations normalised away by `as_ggcpt()`); `ggchangepoint_irregular_index`; `ggchangepoint_na_omitted`; `ggchangepoint_wide_interval`; `ggchangepoint_large_fit`; `ggchangepoint_short_series_warning` (a series too short for the result to be interpretable); `ggchangepoint_dropped_input` (constant coordinates or out-of-range locations left out); `ggchangepoint_engine_failed` (some methods failed and were left out); `ggchangepoint_recycled_input`; `ggchangepoint_constraint` and `ggchangepoint_deprecated`. Anything else is plain `ggchangepoint_warning`.

Messages

Informational messages inherit from `ggchangepoint_message`, so `suppressMessages(classes = "ggchangepoint_message")` silences this package without silencing the engines.

See Also

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

Examples

```
res <- tryCatch(cpt_detect(c(1, 2), method = "pelt"),
               ggchangepoint_short_series = function(e) "too short")
res

# Branch on the kind of failure, not on its wording:
tryCatch(
  cpt_detect(rnorm(50), method = "pelt", change_in = "network"),
  ggchangepoint_unsupported = function(e) {
    paste("unsupported:", e$method, "offers", toString(e$unsupported))
  }
)
```

ggchangepoint-ggproto *ggproto objects for the changepoint layers*

Description

The layers are real **ggplot2** extensions: each `geom_*()` is a layer built on one of these, with its own default aesthetics, required aesthetics and legend glyph, so they take part in scales, guides and the position system like any geom, and can be extended with `ggplot2::ggproto()`. Until 0.6.0 every layer was a thin wrapper around a stock geom and borrowed its glyph.

GeomChangepoint a vertical rule at each xintercept ([geom_changepoint\(\)](#)); key: a vertical rule.
 GeomCptSegment a segment level ([geom_cpt_segment\(\)](#)).
 GeomCptCi a horizontal interval with caps ([geom_cpt_ci\(\)](#)); key: the same interval.
 GeomCptRegion a band from xmin to xmax, full height unless ymin/ymax are mapped ([geom_cpt_region\(\)](#)); key: a shaded band.
 GeomCptLabel a labelled interval behind the series ([geom_cpt_label\(\)](#)).
 GeomCptEvent a dotted rule with its label ([geom_cpt_event\(\)](#)); key: a dotted rule with a flag.
 StatChangepoint, StatCptRegion detection inside the layer ([stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)).

Format

ggproto objects.

See Also

Other ggplot2 layers: [geom_changepoint\(\)](#), [geom_cpt_ci\(\)](#), [geom_cpt_event\(\)](#), [geom_cpt_label\(\)](#), [geom_cpt_region\(\)](#), [geom_cpt_segment\(\)](#), [stat_changepoint\(\)](#), [stat_cpt_region\(\)](#)

Examples

```
library(ggplot2)
class(GeomChangepoint)
GeomCptRegion$default_aes
```

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

<code>x</code>	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 <code>cpt_batch()</code> for a panel of series.
<code>methods</code>	Character vector of method names (passed to <code>cpt_detect</code>).
<code>layout</code>	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 <code>tidy()</code> , and treat the overlay as a picture of agreement rather than of location.
<code>change_in</code>	What to detect change in. Passed to each detector.
<code>seed</code>	Optional seed for reproducible parallelism. Passed to <code>future::future_lapply()</code> as <code>future.seed</code> , and to <code>set.seed()</code> when running sequentially. Left NULL under a parallel plan, <code>future.seed = TRUE</code> is used, so the workers get parallel-safe streams but the run is not reproducible. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	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()`.

Choosing the method by its answer

Running several methods and reporting the one that found a change, as if it were the only one run, is a garden of forking paths with a plot attached, and fifty methods behind one call make it easier here than anywhere else. A method chosen because of its answer carries no valid p-value, and neither does any test at its changepoints: the selection across methods is a second selection on top of the one each method already makes (see `cpt_test()`'s `selection_adjusted`). Choose the method before seeing the data (`cpt_recommend()` answers from what you know about the problem, not from the series), or report every method you ran. Agreement across methods, as `cpt_consensus()` counts it, is a robustness display, not a correction: the methods share engines, costs and assumptions.

Examples

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

ggcpt_compare_table	<i>Comparison table</i>
---------------------	-------------------------

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	<i>Evaluation visualization</i>
------------	---------------------------------

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 = NULL, margin = 5)
```

Arguments

pred	Predicted changepoint indices, or a ggcpt fit (whose changepoints and series are used).
truth	Ground truth changepoint indices.
data_vec	The original data vector (for context). Taken from pred when it is a fit.
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	<i>Interactive changepoint plot</i>
-------------------	-------------------------------------

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

<code>x</code>	A <code>ggcpt</code> object or a <code>ggplot</code> object.
<code>engine</code>	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 <code>ggplot</code> itself to interactive SVG, so faceting and every layer survive and the result composes with other <code>htmlwidgets</code> . Neither is a dependency; whichever you ask for must be installed.
<code>width_svg</code> , <code>height_svg</code>	Figure size in inches for <code>engine = "ggiraph"</code> .
<code>...</code>	Additional arguments passed to <code>autoplot()</code> when <code>x</code> is a <code>ggcpt</code> object.

Value

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

See Also

Other plotting: `ggcpt_compare_table()`, `ggcpt_eval()`, `ggcptplot()`, `ggccpplot()`

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

<code>x</code>	A ggcpt object.
<code>...</code>	Additional arguments passed to methods.
<code>.name_repair</code>	Ignored (the changepoints tibble already has valid, unique names); present for signature compatibility with the generic.
<code>row.names, optional</code>	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	<i>Base plot() methods for ggchange point result objects</i>
--------------------	--

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_effect'
plot(x, ...)

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

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

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

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

Arguments

<code>x</code>	A result object created by one of the package's <code>cpt_*</code> () functions.
<code>...</code>	Passed to the corresponding <code>autoplot()</code> 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	<i>Posterior probability plot for Bayesian results</i>
-----------------	--

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	<i>Run-length posterior heatmap for Bayesian online results</i>
-----------------	---

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)
```

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. For a change in mean (and for np) minseglen defaults to 2 rather than the engine's 1, so no segment is a single observation; pass it to choose another value. SegNeigh keeps the engine's default, which is the only one it implements.
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)))
```

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

<code>data</code>	A numeric vector (for univariate) or matrix/data.frame (for multivariate).
<code>algorithm</code>	Either <code>divisive</code> or <code>agglo</code> . <code>divisive</code> is the default.
<code>min_size</code>	Minimum number of observations between change points. By default is 2. This argument is only applied when <code>algorithm = "divisive"</code> .
<code>...</code>	Extra arguments to pass on either from <code>e.divisive()</code> or <code>e.agglo()</code> .
<code>cptline_alpha</code>	The value of alpha for the vertical changepoint line(s), default is 1, meaning no transparency.
<code>cptline_color</code>	The color for the vertical changepoint line(s), default is blue.
<code>cptline_type</code>	The linetype for the vertical changepoint line(s), default is solid.
<code>cptline_linewidth</code>	The linewidth for the vertical changepoint line(s), default is 0.5.
<code>cptline_size</code>	Deprecated. Use <code>cptline_linewidth</code> instead.
<code>index</code>	Optional. A vector of x-axis labels (e.g. dates) of the same length as <code>data</code> .
<code>show_points</code>	Logical. Whether to draw data points. Defaults to <code>TRUE</code> when <code>length(data) <= 500</code> , <code>FALSE</code> otherwise.
<code>show_line</code>	Logical. Whether to draw the line. Defaults to <code>TRUE</code> .

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

```
ggcplot(c(rnorm(100,0,1),rnorm(100,0,10)))
ggcplot(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) and `engine_version` (the version of the engine package that produced the result, recorded when it was made, or NA for a result with no engine package, such as one built by `as_ggcpt()`; see `cpt_verify()`) and `family` (the distribution family the fit was asked for with `cpt_detect(family = )`, or NA for the method's own default).

Examples

```
set.seed(2026)
fit <- cpt_detect(c(rnorm(60), rnorm(60, 3)), method = "pelt")
glance(fit)
```

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

<code>x</code>	A numeric matrix or data frame, rows as time points.
<code>threshold</code>	Detection threshold on the CUSUM statistic. When <code>NULL</code> (the default) it is calibrated by permutation: the time order is shuffled <code>n_perm</code> times, which destroys any changepoint while preserving the marginal distributions, and the threshold is the $1 - \alpha$ quantile of the largest statistic seen.
<code>alpha</code>	Family-wise level for the permutation threshold. Defaults to <code>0.05</code> .
<code>n_perm</code>	Permutations used to calibrate the threshold. Defaults to <code>20</code> ; raise it for a sharper threshold at proportional cost.
<code>delta</code>	Minimum spacing between changepoints. Defaults to <code>max(10, floor(n / 20))</code> .
<code>seed</code>	Optional seed (the permutation calibration is random). The seed is scoped to this call: <code>.Random.seed</code> 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()`, `eclip_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdreg_wrapper()`, `idetector_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

<code>x</code>	A numeric matrix or data frame of covariates, rows as time points.
<code>response</code>	A numeric vector of responses, one per row of <code>x</code> . Required. Reachable through <code>cpt_detect()</code> as <code>cpt_detect(X, method = "hdreg", response = y)</code> .
<code>gamma_set</code>	Candidate values of the ℓ_0 tuning parameter. Defaults to a small grid scaled by the series length.
<code>lambda_set</code>	Candidate lasso penalties. Defaults to <code>c(0.01, 0.1, 1)</code> .
<code>delta</code>	Minimum spacing. Defaults to <code>max(5, floor(n / 20))</code> .
<code>...</code>	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()`, `fabiseach_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `idetector_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	<i>Isolate-Detect wrapper</i>
-----------------	-------------------------------

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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to <code>IDetect::ID()</code> .

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

inspect_wrapper	<i>inspect wrapper: high-dimensional changepoints via sparse projection</i>
-----------------	---

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

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

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\(\)](#), [idetector_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\(\)](#), [as_json\(\)](#), [cpt_annotations\(\)](#), [cpt_export\(\)](#), [cpt_verify\(\)](#), [ggchangepoint-conditions](#), [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

<code>x</code>	A numeric vector, matrix, or data frame (columns are variables).
<code>running_stat</code>	Which running statistic to monitor: "mean", "var", "autocorr", or "corr" (correlation requires at least two columns). Defaults to "mean".
<code>wsize</code>	Sliding window size for the running statistic. Defaults to 25.
<code>nperm</code>	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.
<code>kmax</code>	Maximum number of changepoints considered. Defaults to 10.
<code>alpha</code>	Significance level of the permutation test. Defaults to 0.05.
<code>seed</code>	Optional seed for reproducibility of the permutation test. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	Additional arguments passed to <code>kcpRS::kcpRS()</code> .

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()`, `fabiseach_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: <code>.Random.seed</code> 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](https://doi.org/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()`, `fabiseach_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetector_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmjo_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

<code>x</code>	A numeric vector.
<code>change_in</code>	Shorthand for the segment model when <code>model</code> is not given: "mean" fits a change in intercept (<code>list(y ~ 1, ~ 1)</code>), "slope" a change in slope (<code>list(y ~ 1 + t, ~ 0 + t)</code>), "var" a change in residual standard deviation (<code>list(y ~ 1, ~ 0 + sigma(1))</code>).
<code>n_changepoints</code>	Number of changepoints in the shorthand model. Defaults to 1.
<code>model</code>	An explicit mcp model: a list of formulas. Overrides <code>change_in</code> and <code>n_changepoints</code> , and is the reason to reach for this engine at all.
<code>prior</code>	Optional named list of priors, passed to <code>mcp::mcp()</code> .
<code>iter, adapt, chains</code>	Sampler settings, passed through.

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

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 $x \geq cp$), 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 (<code>mosum::multiscale.localPrune()</code>) instead of a single bandwidth? Defaults to FALSE.
seed	Optional seed for reproducibility. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to <code>mosum::mosum()</code> or <code>mosum::multiscale.localPrune()</code> .

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](https://doi.org/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\(\)](#), [idetector_wrapper\(\)](#), [inspect_wrapper\(\)](#), [kcp_wrapper\(\)](#), [kwc_wrapper\(\)](#), [mcp_wrapper\(\)](#), [network_wrapper\(\)](#), [not_wrapper\(\)](#), [npmjo_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	<i>Dynamic-network changepoints</i>
-----------------	-------------------------------------

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

<code>x</code>	The network sequence: an $n \times p^2$ matrix of vectorised adjacency matrices, or an $n \times p \times p$ array.
<code>copy2</code>	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.
<code>n_intervals</code>	Number of random intervals. Defaults to 100.
<code>threshold</code>	Detection threshold. When NULL, calibrated by permutation exactly as in hdcov_wrapper() .
<code>alpha, n_perm</code>	Level and number of permutations for that calibration.
<code>delta</code>	Minimum spacing. Defaults to <code>max(5, floor(n / 20))</code> .
<code>seed</code>	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> 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

<code>changepoints</code>	A tibble with columns <code>cp</code> and <code>cp_value</code> .
<code>segments</code>	A tibble with segment information: <code>seg_id</code> , <code>start</code> , <code>end</code> , <code>n</code> , <code>param_estimate</code> . <code>param_estimate</code> 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 <code>change_in = "var"</code> result therefore has a <code>param_estimate</code> 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: <code>augment()</code> 's <code>.fitted/ .resid</code> , <code>cpt_gt()</code> 's level columns, <code>summary()</code> , and the residual construction the bootstrap in <code>cpt_confint()</code> and <code>cpt_stability()</code> uses.
<code>data</code>	A tibble with <code>index</code> and <code>value</code> .
<code>method</code>	Character. The detection method used. A length-one string; defaults to <code>NA_character_</code> . (A zero-length value would make <code>glance()</code> return zero rows instead of its documented single row, because every other column would be recycled against it.)
<code>change_in</code>	Character. What was detected (e.g. <code>"mean"</code> , <code>"var"</code> , <code>"meanvar"</code>). A length-one string; defaults to <code>NA_character_</code> .
<code>penalty</code>	A list with <code>type</code> and <code>value</code> .

<code>fit</code>	The raw upstream object. Every wrapper stores one except "ecp": <code>ecp::e.agglo()</code> 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 <code>ecp::e.divisive()</code> or <code>ecp::e.agglo()</code> directly if you need it. A few of the engines that <i>are</i> kept are still large relative to the data: measured on a 2000-point series, <code>strucchange</code> costs about 135 MB (a triangular $O(n^2)$ RSS matrix), <code>bfast</code> about 53 MB and <code>bocpd</code> 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: <code>cpt_batch(keep_fit = FALSE)</code> drops them, or assign <code>res\$fit <- NULL</code> yourself.
<code>call</code>	The matched call.
<code>cp_convention</code>	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".
<code>runtime</code>	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()`, `as_json()`, `cpt_annotations()`, `cpt_export()`, `cpt_verify()`, `ggchangepoint-conditions`, `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	<i>NOT wrapper: Narrowest-Over-Threshold</i>
-------------	--

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: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
...	Additional arguments passed to <code>not::not()</code> .

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	<i>Nonparametric MOSUM wrapper (NP-MOJO)</i>
----------------	--

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()`, `idetct_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 α . 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 $1 - \alpha$, <i>every</i> returned interval contains a changepoint. Defaults to 0.1.
variant	Which NSP procedure to run: "poly" (default) <code>nsp::nsp_poly()</code> : piecewise polynomial signal, Gaussian noise of constant variance. "selfnorm" <code>nsp::nsp_poly_selfnorm()</code> : self-normalised, for heavy tails and heteroscedasticity. Slower. "ar" <code>nsp::nsp_poly_ar()</code> : autoregressive noise of order <code>ord</code> . "tvreg" <code>nsp::nsp_tvreg()</code> : a general linear model whose coefficients change; requires <code>covariates</code> .
change_in	"mean" (a piecewise-constant signal, <code>deg = 0</code>) or "slope" (piecewise linear, <code>deg = 1</code>). Ignored when <code>deg</code> is given explicitly, and when <code>variant = "tvreg"</code> (which takes its model from <code>covariates</code>).
deg	Degree of the piecewise polynomial. Derived from <code>change_in</code> when <code>NULL</code> .
M	Number of intervals drawn. Defaults to 1000; the engine's own default.
covariates	A design matrix for <code>variant = "tvreg"</code> , with one row per observation.
ord	AR order for <code>variant = "ar"</code> . 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: <code>.Random.seed</code> 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()`, `idetector_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmajo_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 $\max(20, \text{floor}(0.2 * n))$, capped at $n/2$.
thresh	Threshold specification passed to <code>ocd::ChangepointDetector()</code> ; "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 <code>c(diag =, off_d =, off_s =)</code> , 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 <code>ocd::ChangepointDetector()</code> .

Value

A `ggcpt` object. Because the detector is online, it declares a change without estimating where it began. `cp` is the declaration time *minus one*: the latest changepoint consistent with the alarm in this package's convention (the last observation before the change), and exact when the change is declared on the first observation after it; a later declaration makes `cp` late by the detection delay. The `declared_at` column holds the declaration time itself. (Before 0.6.0 `cp` was the declaration time, one position late even at zero delay, which the convention test found.) Compare [cpm_wrapper\(\)](#), whose engine estimates the location separately from its `detection_time`.

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()`, `idetector_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmjo_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), 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

<code>x</code>	A numeric matrix or data frame with rows as time points and columns as coordinates.
<code>threshold_d_const</code> , <code>threshold_bj_const</code> , <code>threshold_partial_const</code>	Leading constants of the dense, Berk-Jones and partial-sum thresholds.
<code>empirical</code>	Calibrate the thresholds by Monte Carlo rather than using the theoretical values? Slower but sharper; defaults to FALSE.
<code>N</code>	Monte Carlo samples when <code>empirical = TRUE</code> .
<code>seed</code>	Optional seed (used by the empirical calibration). The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	Additional arguments passed to <code>HDCD::Pilliat()</code> .

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()`, `idetector_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)
```

predict.ggcpt

Forecast from a segment of a segmentation

Description

The point of detecting a regime change is usually that the earlier regime should no longer inform what happens next. `predict()` on a `ggcpt` forecasts from one segment only (by default the last), with the segment’s own model.

Usage

```
## S3 method for class 'ggcpt'
predict(
  object,
  newdata = NULL,
  h = 1,
  segment = "last",
  models = NULL,
  level = 0.95,
  ...
)
```

Arguments

object	A ggcpt object.
newdata	Optional data frame of covariates (for a formula fit, or a segment model with covariates), or of time positions.
h	Forecast horizon, when newdata is not given: the positions $n + 1$ to $n + h$. Defaults to 1.
segment	Which segment to forecast from: "last" (the default) or a segment number.
models	Optional <code>cpt_segment_models()</code> result; built with its defaults when omitted.
level	Level of the prediction interval. Defaults to 0.95.
...	Further arguments for the model's <code>predict()</code> method.

Value

A tibble with time (when forecasting by horizon), `.pred`, and `.lower/.upper` when the model supplies a prediction interval.

See Also

Other segment models: `cpt_segment_models()`

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(100), rnorm(100, 3)), method = "pelt")
predict(fit, h = 3)
```

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\(\)](#), [as_json\(\)](#), [cpt_annotations\(\)](#), [cpt_export\(\)](#), [cpt_verify\(\)](#), [ggchangepoint-conditions](#), [is_ggcpt\(\)](#), [new_ggcpt\(\)](#)

Examples

```
set.seed(2026)
print(cpt_detect(c(rnorm(40), rnorm(40, 4)), method = "pelt"))
```

scale_colour_cpt	<i>Colour-vision-safe scales for changepoint methods</i>
------------------	--

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 <code>ggplot2::discrete_scale()</code> .
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 <code>ggplot2::scale_fill_manual()</code> / <code>ggplot2::scale_colour_manual()</code> .
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	<i>Broken-line regression wrapper (segmented)</i>
-------------------	---

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,
  data = NULL,
  seg_z = NULL,
  family = c("gaussian", "poisson", "binomial"),
  ...
)
```

Arguments

<code>x</code>	A numeric vector (a line in time is segmented), or a model formula (supply data).
<code>npsi</code>	Number of breakpoints to estimate. Defaults to 1.
<code>conf_level</code>	Confidence level for breakpoint intervals. Defaults to 0.95.
<code>seed</code>	Optional seed (the estimator uses bootstrap restarting). The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>data</code>	A data frame, for formula input.
<code>seg_z</code>	For formula input, the covariate whose relationship with the response breaks: its name, or a one-sided formula (<code>~ t</code>) as segmented 's own <code>seg.Z</code> takes it. Defaults to the formula's only numeric covariate, and must be named when there are several.
<code>family</code>	"gaussian" (a linear model, the default), "poisson" or "binomial" (a generalised linear model on the log or logit scale).
<code>...</code>	Additional arguments passed to <code>segmented::segmented()</code> , for example <code>psi</code> (starting values) or <code>fixed.psi</code> .

Details

Called with a numeric vector it segments a line in time. Called with a formula and data it does what the engine exists for: breakpoints in the relationship between the response and a covariate, `seg_z`. The result is then ordered by that covariate, which becomes its index, so `tidy()`'s `cp_index` and the plot speak in the covariate's units, and `psi`, `psi_lower` and `psi_upper` give the breakpoints on that scale exactly.

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. Formula input adds `psi`, `psi_lower` and `psi_upper` (the breakpoints on the covariate's scale) and a `$coefficients` table with the slope of `seg_z` in each segment.

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()`, `idetector_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)

# A breakpoint in a dose-response relationship
d <- data.frame(dose = runif(150, 0, 10))
d$response <- 2 + 0.8 * pmin(d$dose, 6) + rnorm(150, 0, 0.4)
fit <- segmented_wrapper(response ~ dose, data = d)
fit$changepoints[, c("cp", "psi", "psi_lower", "psi_upper")]
```

signal_blocks	<i>Blocks test signal</i>
---------------	---------------------------

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: <code>.Random.seed</code> 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	<i>FMS (Four-Metric-Segments) test signal</i>
------------	---

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: <code>.Random.seed</code> 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	<i>Mix test signal</i>
------------	------------------------

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	<i>Stairs test signal</i>
---------------	---------------------------

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	<i>Teeth test signal</i>
--------------	--------------------------

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: <code>.Random.seed</code> 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 α ; 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", "poisson", "binomial"),
  ...
)
```

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), "hsmuce" (HSMUCE, segment-wise variance), "poisson" (counts: a change in the rate) or "binomial" (0/1 outcomes: a change in the probability). Defaults to "gauss". The two count families run <code>stepR::smuceR()</code> , the multiscale estimator for exponential families, with the same level and the same jump intervals; <code>cpt_detect(x, method = "smuce", family = "poisson")</code> reaches them. stepR documents <code>smuceR()</code> as deprecated but working, and as the only route to these families until <code>stepFit()</code> gains them; should it be removed, the error says so and names the methods that fit the same family. With <code>alpha</code> given, <code>smuceR()</code> takes its threshold from the asymptotic null distribution rather than a simulated one. The remaining <code>stepR</code> families ("jsmurf", "mDependentPS", ...) all require a filter or covariance specification; call <code>stepR::stepFit()</code> 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 <code>cpt_simulate(sd = 0)</code> 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 <code>stepR::stepFit()</code> (<code>stepR::smuceR()</code> for the count families).

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()`, `idetector_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

<code>x</code>	A numeric vector (or a two-column matrix for <code>parameter = "bivcor"</code>).
<code>parameter</code>	Which parameter to test for changes: "mean", "variance", "acf", or "bivcor" (bivariate correlation). Defaults to "mean".
<code>confidence</code>	Confidence level of the self-normalised test, one of 0.9, 0.95, 0.99, 0.995 or 0.999. Defaults to 0.9.
<code>grid_size</code>	Grid size controlling the local-window sweep; when NULL the engine's default is used.
<code>...</code>	Additional arguments passed to <code>SNSeg::SNSeg_Uni()</code> .

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	<i>Changepoint detection stat</i>
------------------	-----------------------------------

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

<code>mapping</code>	Aesthetic mappings.
<code>data</code>	A data frame.
<code>geom</code>	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. <code>inherit.aes</code> 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.
<code>position</code>	Position adjustment.
<code>...</code>	Other arguments passed to the geom.
<code>method</code>	Detection method (passed to <code>cpt_detect</code>).
<code>change_in</code>	What to detect change in (passed to <code>cpt_detect</code>).
<code>na.rm</code>	If FALSE, missing values are removed.
<code>show.legend</code>	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\(\)](#), [ggchangepoint-ggproto](#), [stat_cpt_region\(\)](#)

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")
```

stat_cpt_region	<i>Significance regions computed in the layer</i>
-----------------	---

Description

The region twin of [stat_changepoint\(\)](#): runs detection on the layer's own x/y and draws what the method returns as an interval, so `ggplot(d, aes(t, y)) + geom_line() + stat_cpt_region()` is the whole plot. Narrowest Significance Pursuit (`method = "nsp"`, the default) returns regions, each containing a changepoint at a global significance level; a method with location intervals (`"smuce"`, `"strucchange"`, `"segmented"`, ...) draws those instead.

Usage

```
stat_cpt_region(
  mapping = NULL,
  data = NULL,
  geom = GeomCptRegion,
  position = "identity",
  ...,
  method = "nsp",
  na.rm = FALSE,
  show.legend = NA
)
```

Arguments

mapping	Aesthetic mappings.
data	A data frame.
geom	The geom to draw with. Defaults to geom_cpt_region() 's.
position	Position adjustment.
...	Further arguments: those the detector takes (for example <code>alpha</code> for NSP's level, or <code>seed</code>) go to it, the rest to the geom. With <code>method = "nsp"</code> that makes <code>alpha</code> the level, so set the band's transparency through its fill instead (<code>fill = ggplot2::alpha("steelblue", 0.3)</code>).

<code>method</code>	Detection method: "nsp" (the default) or any method whose result carries regions or location intervals (see <code>cpt_methods()</code> \$ci).
<code>na.rm</code>	If FALSE, missing values are removed.
<code>show.legend</code>	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\(\)](#), [ggchangepoint-ggproto](#), [stat_changepoint\(\)](#)

Examples

```
library(ggplot2)
set.seed(2026)
d <- data.frame(t = 1:200, y = c(rnorm(100), rnorm(100, 2)))
ggplot(d, aes(t, y)) + stat_cpt_region(seed = 1) + geom_line()
```

<code>strucchange_wrapper</code>	<i>Bai-Perron structural break wrapper (strucchange)</i>
----------------------------------	--

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 ($y \sim 1$); 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

<code>x</code>	A numeric vector (mean-shift mode), or a model formula (regression mode; supply data too).
<code>data</code>	Optional data frame for formula input.
<code>breaks</code>	Maximum number of breaks; when NULL the number is chosen by BIC.
<code>h</code>	Minimal segment size, as a fraction of the sample size (or an integer count). Defaults to 0.15.
<code>conf_level</code>	Confidence level for the break-date intervals. Defaults to 0.95.
<code>...</code>	Additional arguments passed to <code>strucchange::breakpoints()</code> .

Value

A `ggcpt` object with `ci_lower/ci_upper` columns on the `changepts` 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$changepts`) 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()`, `idetct_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.

Examples

```
set.seed(2026)
summary(cpt_detect(c(rnorm(60), rnorm(60, 3)), method = "pelt"))
```

taylor_wrapper	<i>Taylor's change point analyzer</i>
----------------	---------------------------------------

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

<code>x</code>	A numeric vector.
<code>n_bootstraps</code>	Bootstrap samples per candidate. Defaults to 1000; the engine accepts 100 to 1,000,000.
<code>min_candidate_conf</code>	Minimum confidence for a candidate to be considered, between 0.3 and 1. Defaults to 0.5.
<code>min_conf</code>	Minimum confidence for a changepoint to be reported, between 0.5 and 1. Defaults to 0.9.
<code>conf_level</code>	Confidence level of the reported location intervals, between 0.9 and 0.999 (the engine's range). Defaults to 0.95.
<code>seed</code>	Optional seed (the procedure is bootstrap-based). The seed is scoped to this call: <code>.Random.seed</code> 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/ggchangeoint/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()`, `fabiseach_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()`, `npmajo_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()`, `npmajo_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	<i>Tidy a ggcpt object</i>
------------	----------------------------

Description

Returns the changepoints tibble (one row per changepoint), or the coefficients of every segment.

Usage

```
## S3 method for class 'ggcpt'
tidy(x, what = c("changepoints", "coefficients"), conf_level = 0.95, ...)
```

Arguments

x	A ggcpt object.
what	"changepoints" (the default) or "coefficients": one row per segment and model term, with estimate, std_error and a Wald interval. For a formula fit the terms are the regression's; for a series they are the segment level (and its slope in time, for a change in slope), so a mean-shift result reports each segment's mean with its standard error. Engines that estimate the coefficients themselves (segmented 's slopes, fastcpd 's per-segment parameters) report their own estimates.
conf_level	Confidence level for the coefficient intervals.
...	Additional arguments (ignored).

Value

A tibble. For "changepoints": columns cp, cp_value, and any method-specific columns. For "coefficients": segment, start, end, term, estimate, std_error, conf_low, conf_high (plus coordinate for a multivariate result).

Examples

```
set.seed(1)
fit <- cpt_detect(c(rnorm(80), rnorm(80, 3)), method = "pelt")
tidy(fit)
tidy(fit, "coefficients")
```

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()`, `npmajo_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	<i>VAR(1) changepoints</i>
-------------	----------------------------

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

<code>x</code>	A numeric matrix or data frame, rows as time points.
<code>gamma_set</code>	Candidate values of the ℓ_0 tuning parameter. Defaults to a small grid scaled by the series length.
<code>lambda_set</code>	Candidate lasso penalties. Defaults to <code>c(0.01, 0.1, 1)</code> .
<code>delta</code>	Minimum spacing. Defaults to <code>max(5, floor(n / 20))</code> .
<code>...</code>	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\(\)](#), [npmajo_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))
```

wbs_wrapper	<i>WBS wrapper: Wild Binary Segmentation</i>
-------------	--

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
```

wbs2_wrapper	<i>WBS2 wrapper: Wild Binary Segmentation 2</i>
--------------	---

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 = "sdl1")` 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()`, `idetector_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

<code>x</code>	A numeric vector.
<code>n_intervals</code>	Number of random intervals (M). Defaults to 0, which is the engine's "all dyadic intervals" setting.
<code>cstar</code>	The unbalancedness parameter c_* 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.
<code>lambda</code>	How many wavelet scales are used when <code>scales</code> is NULL: $\text{floor}(3 * \lambda * \log(\log(n)))$ of them, from the finest down. Defaults to the engine's 0.75.
<code>scales</code>	Wavelet scales to use: at least two different whole numbers from 1 to $\text{floor}(\log_2(n)) - 1$. NULL lets the engine choose.
<code>seed</code>	Optional seed. The seed is scoped to this call: <code>.Random.seed</code> is saved and restored, so a seeded call inside a simulation loop does not pin the loop's own stream.
<code>...</code>	Additional arguments passed to <code>wbsts::wbs.lsw()</code> .

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()`, `eep_wrapper()`, `envcpt_wrapper()`, `esac_wrapper()`, `fabisearch_wrapper()`, `fastcpd_wrapper()`, `fcov_wrapper()`, `fmean_wrapper()`, `fpop_wrapper()`, `geomcp_wrapper()`, `hdcov_wrapper()`, `hdreg_wrapper()`, `idetector_wrapper()`, `inspect_wrapper()`, `kcp_wrapper()`, `kwc_wrapper()`, `mcp_wrapper()`, `mosum_wrapper()`, `network_wrapper()`, `not_wrapper()`, `npmjo_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)
```

Index

* accessibility scales

- scale_colour_cpt, 178
- scale_fill_cpt_label, 179
- theme_ggcpt, 196

* changepoint engines

- bcp_wrapper, 15
- beast_wrapper, 17
- bfast_wrapper, 18
- binsegrcpp_wrapper, 19
- bocpd_wrapper, 21
- cpm_wrapper, 22
- cpop_wrapper, 23
- cpt_wrapper, 113
- decafs_wrapper, 115
- ecp_wrapper, 116
- envcpt_wrapper, 117
- esac_wrapper, 118
- fabisearch_wrapper, 120
- fastcpd_wrapper, 122
- fcov_wrapper, 124
- fmean_wrapper, 126
- fpop_wrapper, 127
- geomcp_wrapper, 136
- hdcov_wrapper, 152
- hdreg_wrapper, 153
- idetect_wrapper, 155
- inspect_wrapper, 156
- kcp_wrapper, 158
- kwc_wrapper, 159
- mcp_wrapper, 161
- mosum_wrapper, 163
- network_wrapper, 164
- not_wrapper, 168
- npmjo_wrapper, 169
- nsp_wrapper, 170
- ocd_wrapper, 172
- pilliat_wrapper, 174
- segmented_wrapper, 180
- smuce_wrapper, 186

- sn_wrapper, 187
- strucchange_wrapper, 191
- taylor_wrapper, 194
- tguh_wrapper, 195
- trend_wrapper, 198
- var_wrapper, 199
- wbs2_wrapper, 201
- wbs_wrapper, 200
- wbsts_wrapper, 203

* datasets

- cpt_calibration, 33
- cpt_data_types, 41
- cpt_invariances, 58
- cpt_noise_benchmark, 76
- cpt_null_sizes, 78
- cpt_runtimes, 91

* ggplot2 layers

- geom_changepoint, 129
- geom_cpt_ci, 130
- geom_cpt_event, 131
- geom_cpt_label, 132
- geom_cpt_region, 133
- geom_cpt_segment, 135
- ggchangepoint-ggproto, 138
- stat_changepoint, 189
- stat_cpt_region, 190

* inference

- cpt_assumptions, 27
- cpt_attribute_event, 28
- cpt_effect, 49
- cpt_gof, 53
- cpt_null_power, 77
- cpt_robustness, 90
- cpt_test_at, 108
- cpt_test_null, 110

* measurement tables

- cpt_calibration, 33
- cpt_data_types, 41
- cpt_invariances, 58

- cpt_noise_benchmark, 76
- cpt_null_sizes, 78
- cpt_runtimes, 91
- * **method registry**
 - cpt_families, 51
- * **plotting**
 - ggcpt_compare_table, 141
 - ggcpt_eval, 142
 - ggcpt_interactive, 143
 - ggcptplot, 148
 - ggccpplot, 149
- * **result class**
 - annotate_segments, 6
 - as_cpt_series, 8
 - as_ggcpt, 9
 - as_json, 10
 - cpt_annotations, 26
 - cpt_export, 50
 - cpt_verify, 112
 - ggchangept-conditions, 137
 - is_ggcpt, 157
 - new_ggcpt, 166
 - print.ggcpt, 177
- * **segment models**
 - cpt_segment_models, 95
 - predict.ggcpt, 176
- * **test signals**
 - cpt_simulate, 100
 - signal_blocks, 182
 - signal_fms, 183
 - signal_mix, 183
 - signal_stairs, 184
 - signal_teeth, 185
- alarms, 5, 73, 75, 76, 111
- annotate_segments, 6, 8, 10, 12, 26, 51, 113, 138, 157, 167, 178
- as.data.frame, 144
- as.data.frame.ggcpt (ggcpt_methods), 144
- as_cpt_labels, 7, 62
- as_cpt_series, 7, 8, 10, 12, 26, 51, 113, 138, 157, 167, 178
- as_ggcpt, 7, 8, 9, 12, 26, 51, 86, 87, 113, 138, 151, 157, 166, 167, 178
- as_json, 7, 8, 10, 10, 26, 51, 89, 113, 138, 157, 167, 178
- as_tibble.ggcpt (ggcpt_methods), 144
- augment.ggcpt, 12, 47, 48
- autoplot.ggcpt, 13, 38, 48, 140, 144, 146
- autoplot.ggcpt_batch (cpt_batch), 29
- autoplot.ggcpt_benchmark (cpt_benchmark), 30
- autoplot.ggcpt_consensus (cpt_consensus), 37
- autoplot.ggcpt_delay (cpt_delay), 43
- autoplot.ggcpt_effect (cpt_effect), 49
- autoplot.ggcpt_events (cpt_annotate_events), 24
- autoplot.ggcpt_influence (cpt_influence), 54
- autoplot.ggcpt_label_curve (cpt_label_error_curve), 60
- autoplot.ggcpt_monitor (cpt_monitor), 73
- autoplot.ggcpt_path (cpt_crops), 39
- autoplot.ggcpt_power (cpt_power), 80
- autoplot.ggcpt_recommendation (cpt_recommend), 83
- autoplot.ggcpt_robustness (cpt_robustness), 90
- autoplot.ggcpt_selection (cpt_select), 96
- autoplot.ggcpt_sensitivity (cpt_sensitivity), 99
- autoplot.ggcpt_stability (cpt_stability), 104
- bcp_wrapper, 15, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 147, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- beast_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 147, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- bfast_wrapper, 8, 16, 17, 18, 20, 21, 23, 24, 102, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- binsegrcpp_wrapper, 16, 17, 19, 19, 21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170,

- [172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- [bocpd_wrapper, 16, 17, 19, 20, 21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 147, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- [coef.ggcpt_penalty_model \(cpt_learn_penalty\), 62](#)
- [cpm_wrapper, 16, 17, 19–21, 22, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172–174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- [cpop_wrapper, 16, 17, 19–21, 23, 23, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- [cpt_annotate_events, 24, 28, 89, 131, 132](#)
- [cpt_annotations, 7, 8, 10, 12, 26, 51, 113, 138, 157, 167, 178](#)
- [cpt_assumptions, 11, 27, 28, 38, 50, 53, 78, 84, 91, 105, 110, 111](#)
- [cpt_attribute_event, 28, 28, 50, 53, 78, 91, 110, 111](#)
- [cpt_batch, 18, 29, 45, 82, 140, 141, 167, 175, 192](#)
- [cpt_benchmark, 18, 30, 42, 65, 69, 70, 82, 93, 94, 137, 175](#)
- [cpt_calibration, 33, 41, 58, 77, 78, 92](#)
- [cpt_cite, 34, 85, 87, 89](#)
- [cpt_confint, 28, 35, 39, 86, 89, 107, 108, 166, 172](#)
- [cpt_consensus, 9, 24, 37, 82, 85, 90, 137, 140, 171](#)
- [cpt_crops, 39, 99, 103](#)
- [cpt_data_types, 34, 41, 58, 77, 78, 83, 92](#)
- [cpt_datasets, 26, 31–33, 42](#)
- [cpt_delay, 6, 43, 68, 73–76, 88](#)
- [cpt_detect, 8, 10, 14, 18, 20, 23, 29, 36, 44, 50–52, 55, 60, 62, 63, 67, 68, 79, 82, 84, 86, 90, 98, 100, 104, 112, 114, 115, 123, 128, 140, 154, 167](#)
- [cpt_effect, 28, 47, 49, 53, 78, 91, 110, 111](#)
- [cpt_export, 7, 8, 10, 12, 26, 50, 113, 138, 157, 167, 178](#)
- [cpt_families, 46, 51, 67, 82, 84](#)
- [cpt_gof, 15, 28, 50, 53, 78, 91, 110, 111](#)
- [cpt_gt, 54, 89, 166](#)
- [cpt_import, 12, 113](#)
- [cpt_import \(cpt_export\), 50](#)
- [cpt_influence, 54, 64, 82, 99, 100](#)
- [cpt_install_engines, 56, 68](#)
- [cpt_invariances, 34, 41, 48, 58, 68, 77, 78, 92](#)
- [cpt_label_error, 14, 59, 62, 132, 133](#)
- [cpt_label_error_curve, 59, 60, 63, 64](#)
- [cpt_labels, 14, 59, 61, 133](#)
- [cpt_learn_penalty, 60, 61, 62, 62, 79](#)
- [cpt_leverage, 56, 64](#)
- [cpt_load_tcpd, 33, 42, 65](#)
- [cpt_methods, 12, 14, 35, 44, 48, 52, 57, 66, 83, 85, 92, 173, 202](#)
- [cpt_metrics, 7, 9, 24, 30, 33, 37, 39, 43, 68, 70, 71, 142, 171](#)
- [cpt_metrics_annotated, 26, 30, 65, 70, 70](#)
- [cpt_min_detectable, 71, 77, 83](#)
- [cpt_monitor, 6, 22, 44, 67, 68, 73, 88, 111](#)
- [cpt_noise_benchmark, 34, 41, 58, 76, 78, 83, 92](#)
- [cpt_null_power, 28, 50, 53, 77, 91, 110, 111](#)
- [cpt_null_sizes, 34, 41, 58, 77, 78, 92](#)
- [cpt_penalty, 45, 48, 63, 64, 79, 97, 99, 123](#)
- [cpt_power, 71, 73, 77, 80, 84, 94](#)
- [cpt_recommend, 39, 83, 92, 140](#)
- [cpt_regions, 85, 134, 172](#)
- [cpt_register_method, 10, 34, 45, 66–68, 86, 108](#)
- [cpt_registered_methods, 87](#)
- [cpt_registered_methods \(cpt_register_method\), 86](#)
- [cpt_replay, 43, 44, 75, 76, 88](#)
- [cpt_report, 9, 12, 26, 28, 45, 54, 89](#)
- [cpt_robustness, 28, 38, 50, 53, 78, 90, 105, 110, 111](#)
- [cpt_runtimes, 34, 41, 58, 68, 77, 78, 83, 91](#)
- [cpt_scale_space, 67, 92, 106](#)
- [cpt_scenarios, 72, 81, 83, 93](#)
- [cpt_segment_models, 95, 177](#)
- [cpt_select, 61, 80, 96, 100, 113](#)
- [cpt_sensitivity, 56, 82, 91, 99, 99](#)
- [cpt_simulate, 52, 72, 81–83, 94, 100, 102,](#)

- 182–185
- `cpt_solution_path`, 19, 87, 103, 106
- `cpt_stability`, 36, 56, 89, 90, 104, 166
- `cpt_statistic`, 87, 93, 103, 106, 198
- `cpt_test`, 36, 106, 140
- `cpt_test_at`, 28, 50, 53, 78, 91, 108, 111
- `cpt_test_null`, 28, 50, 53, 78, 91, 110, 110
- `cpt_unregister_method`, 87
- `cpt_unregister_method`
(`cpt_register_method`), 86
- `cpt_update`, 73, 75, 76, 111
- `cpt_verify`, 7, 8, 10, 12, 26, 51, 112, 138,
151, 157, 167, 178
- `cpt_wrapper`, 16, 17, 19–21, 23, 24, 113, 115,
117–119, 121, 123, 125, 127, 128,
136, 153–155, 157, 159, 160, 162,
163, 165, 169, 170, 172, 174, 176,
181, 187, 188, 192, 195, 196,
199–202, 204
- `decafs_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 123, 125, 127,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `ecp_wrapper`, 16, 17, 19–21, 23, 24, 114, 115,
116, 118, 119, 121, 123, 125, 127,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `envcpt_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117, 117, 119, 121, 123, 125,
127, 128, 136, 153–155, 157, 159,
160, 162, 163, 165, 169, 170, 172,
174, 176, 181, 187, 188, 192, 195,
196, 199–202, 204
- `esac_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117, 118, 118, 121, 123, 125,
127, 128, 136, 153–155, 157, 159,
160, 162, 163, 165, 169, 170, 172,
174–176, 181, 187, 188, 192, 195,
196, 199–202, 204
- `fabisearch_wrapper`, 16, 17, 19–21, 23, 24,
114, 115, 117–119, 120, 123, 125,
127, 128, 136, 153–155, 157, 159,
160, 162, 163, 165, 169, 170, 172,
174, 176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `fastcpd_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 122, 125, 127,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `fcov_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 123, 124, 127,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `fmean_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 123, 125, 126,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `format.ggcpt` (`ggcpt_methods`), 144
- `fpop_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 123, 125, 127,
127, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204
- `geom_changepoint`, 15, 129, 130, 132–135,
139, 190, 191
- `geom_cpt_ci`, 129, 130, 132–135, 139, 186,
190, 191
- `geom_cpt_event`, 26, 129, 130, 131, 133–135,
139, 190, 191
- `geom_cpt_label`, 59, 62, 129, 130, 132, 132,
134, 135, 139, 190, 191
- `geom_cpt_region`, 86, 129, 130, 132, 133,
133, 135, 139, 167, 172, 190, 191
- `geom_cpt_segment`, 129, 130, 132–134, 135,
139, 190, 191
- `GeomChangepoint`
(`ggchangepoint-ggproto`), 138
- `geomcp_wrapper`, 16, 17, 19–21, 23, 24, 114,
115, 117–119, 121, 123, 125, 127,
128, 136, 153–155, 157, 159, 160,
162, 163, 165, 169, 170, 172, 174,
176, 181, 187, 188, 192, 195, 196,
199–202, 204

- GeomCptCi (ggchangepoint-ggproto), 138
- GeomCptEvent (ggchangepoint-ggproto), 138
- GeomCptLabel (ggchangepoint-ggproto), 138
- GeomCptRegion (ggchangepoint-ggproto), 138
- GeomCptSegment (ggchangepoint-ggproto), 138
- ggchangepoint-conditions, 30, 137
- ggchangepoint-ggproto, 138
- ggchangepoint_error
 - (ggchangepoint-conditions), 137
- ggchangepoint_warning
 - (ggchangepoint-conditions), 137
- ggcpt_compare, 18, 39, 82, 139
- ggcpt_compare_table, 141, 142, 143, 149, 150
- ggcpt_eval, 141, 142, 143, 149, 150
- ggcpt_interactive, 141, 142, 143, 149, 150
- ggcpt_methods, 144, 146
- ggcpt_plot_methods, 15, 144, 145
- ggcpt_posterior, 15, 17, 147, 161
- ggcpt_runlength, 21, 147
- ggcpt_scale_space, 14, 167
- ggcpt_scale_space (cpt_scale_space), 92
- ggcpt_solution_path, 14, 67, 167
- ggcpt_solution_path
 - (cpt_solution_path), 103
- ggcpt_statistic, 14, 67, 167
- ggcpt_statistic (cpt_statistic), 106
- ggcptplot, 141–143, 148, 150
- ggcplot, 141–143, 149, 149
- glance.ggcpt, 47, 48, 113, 151
- glance.ggcpt_delay (cpt_delay), 43
- glance.ggcpt_segment_models
 - (cpt_segment_models), 95
- hdcov_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 152, 154, 155, 157, 159, 160, 162–165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- hdreg_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 153, 155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- idetector_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- inspect_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 156, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- install.packages, 57
- is_ggcpt, 7, 8, 10, 12, 26, 51, 113, 138, 157, 167, 178
- kcp_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 158, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- kwc_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 159, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- mcp_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 161, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- mosum_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204
- network_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 164, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204

- [176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `new_ggcpt`, [7, 8, 10, 12, 26, 47, 51, 113, 138, 157, 166, 178, 193](#)
- `not_wrapper`, [16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 168, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `npmojo_wrapper`, [16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 169, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `nsp_wrapper`, [10, 14, 16, 17, 19–21, 23, 24, 35, 36, 39, 85, 86, 108, 114, 115, 117–119, 121, 123, 125, 127, 128, 133, 134, 136, 153–155, 157, 159, 160, 162, 163, 165, 167, 169, 170, 170, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `ocd_wrapper`, [16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 172, 176, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `p.adjust`, [107](#)
- `pilliat_wrapper`, [16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 174, 181, 187, 188, 192, 195, 196, 199–202, 204](#)
- `plot.default`, [145](#)
- `plot.ggcpt` (`ggcpt_methods`), [144](#)
- `plot.ggcpt_batch` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_benchmark` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_consensus` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_delay` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_effect` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_events` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_influence` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_label_curve` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_monitor` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_path` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_power` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_recommendation` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_robustness` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_selection` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_sensitivity` (`ggcpt_plot_methods`), [145](#)
- `plot.ggcpt_stability` (`ggcpt_plot_methods`), [145](#)
- `predict.ggcpt`, [96, 176](#)
- `predict.ggcpt_penalty_model` (`cpt_learn_penalty`), [62](#)
- `print.cpt_label_error` (`cpt_label_error`), [59](#)
- `print.ggcpt`, [7, 8, 10, 12, 26, 48, 51, 113, 138, 157, 167, 177](#)
- `print.ggcpt_assumptions` (`cpt_assumptions`), [27](#)
- `print.ggcpt_batch` (`cpt_batch`), [29](#)
- `print.ggcpt_benchmark` (`cpt_benchmark`), [30](#)
- `print.ggcpt_consensus` (`cpt_consensus`), [37](#)
- `print.ggcpt_delay` (`cpt_delay`), [43](#)
- `print.ggcpt_effect` (`cpt_effect`), [49](#)
- `print.ggcpt_events` (`cpt_annotate_events`), [24](#)
- `print.ggcpt_gof` (`cpt_gof`), [53](#)
- `print.ggcpt_influence` (`cpt_influence`), [54](#)
- `print.ggcpt_label_curve` (`cpt_label_error_curve`), [60](#)
- `print.ggcpt_min_detectable`

- (cpt_min_detectable), 71
- print.ggcpt_monitor (cpt_monitor), 73
- print.ggcpt_null_power
 - (cpt_null_power), 77
- print.ggcpt_path (cpt_crops), 39
- print.ggcpt_penalty_model
 - (cpt_learn_penalty), 62
- print.ggcpt_power (cpt_power), 80
- print.ggcpt_recommendation
 - (cpt_recommend), 83
- print.ggcpt_robustness
 - (cpt_robustness), 90
- print.ggcpt_segment_models
 - (cpt_segment_models), 95
- print.ggcpt_selection (cpt_select), 96
- print.ggcpt_sensitivity
 - (cpt_sensitivity), 99
- print.ggcpt_stability (cpt_stability), 104
- print.ggcpt_test_at (cpt_test_at), 108
- print.ggcpt_verification (cpt_verify), 112
- print.summary.ggcpt (summary.ggcpt), 193
- rcpt (cpt_simulate), 100
- scale_color_cpt (scale_colour_cpt), 178
- scale_colour_cpt, 178, 179, 197
- scale_colour_cpt_label
 - (scale_fill_cpt_label), 179
- scale_fill_cpt (scale_colour_cpt), 178
- scale_fill_cpt_label, 133, 178, 179, 197
- scale_linetype_cpt (scale_colour_cpt), 178
- segmented_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153–155, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 180, 187, 188, 192, 195, 196, 199–202, 204
- signal_blocks, 102, 182, 183–185
- signal_fms, 102, 182, 183, 184, 185
- signal_mix, 102, 182, 183, 183–185
- signal_stairs, 102, 182, 183, 184, 184, 185
- signal_teeth, 102, 182–184, 185
- smuce_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 186, 188, 192, 195, 196, 199–202, 204
- sn_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 187, 192, 195, 196, 199–202, 204
- stat_changepoint, 129, 130, 132–135, 139, 189, 190, 191
- stat_cpt_region, 129, 130, 132–135, 139, 190, 190
- StatChangepoint
 - (ggchangepoint-ggproto), 138
- StatCptRegion (ggchangepoint-ggproto), 138
- strucchange_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 191, 195, 196, 199–202, 204
- summary, 177
- summary.ggcpt, 48, 178, 193
- taylor_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 194, 196, 199–202, 204
- tguh_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 195, 199–202, 204
- theme_ggcpt, 15, 178, 179, 196
- tidy, 75, 177
- tidy.cpt_label_error (cpt_label_error), 59
- tidy.cpt_labels (cpt_labels), 61
- tidy.ggcpt, 47, 48, 96, 144, 178, 197
- tidy.ggcpt_assumptions
 - (cpt_assumptions), 27
- tidy.ggcpt_batch (cpt_batch), 29
- tidy.ggcpt_benchmark (cpt_benchmark), 30
- tidy.ggcpt_delay (cpt_delay), 43
- tidy.ggcpt_effect (cpt_effect), 49

- tidy.ggcpt_events
 - (cpt_annotate_events), 24
- tidy.ggcpt_gof (cpt_gof), 53
- tidy.ggcpt_influence (cpt_influence), 54
- tidy.ggcpt_label_curve
 - (cpt_label_error_curve), 60
- tidy.ggcpt_min_detectable
 - (cpt_min_detectable), 71
- tidy.ggcpt_monitor (cpt_monitor), 73
- tidy.ggcpt_null_power (cpt_null_power), 77
- tidy.ggcpt_path (cpt_crops), 39
- tidy.ggcpt_penalty_model
 - (cpt_learn_penalty), 62
- tidy.ggcpt_power (cpt_power), 80
- tidy.ggcpt_recommendation
 - (cpt_recommend), 83
- tidy.ggcpt_robustness (cpt_robustness), 90
- tidy.ggcpt_segment_models
 - (cpt_segment_models), 95
- tidy.ggcpt_selection (cpt_select), 96
- tidy.ggcpt_sensitivity
 - (cpt_sensitivity), 99
- tidy.ggcpt_stability (cpt_stability), 104
- tidy.ggcpt_test_at (cpt_test_at), 108
- tidy.ggcpt_verification (cpt_verify), 112
- trend_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 198, 200–202, 204
- var_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199, 199, 201, 202, 204
- wbs2_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199, 200, 201, 201, 204
- wbs_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199, 200, 200, 202, 204
- wbsts_wrapper, 16, 17, 19–21, 23, 24, 114, 115, 117–119, 121, 123, 125, 127, 128, 136, 153, 154, 156, 157, 159, 160, 162, 163, 165, 169, 170, 172, 174, 176, 181, 187, 188, 192, 195, 196, 199–202, 203