Package: tidychangepoint (via r-universe)

August 20, 2024

Title A Tidy Framework for Changepoint Detection Analysis

Version 0.0.1

Description Changepoint detection algorithms for R are widespread but have different interfaces and reporting conventions. This makes the comparative analysis of results difficult. We solve this problem by providing a tidy, unified interface for several different changepoint detection algorithms. We also provide consistent numerical and graphical reporting leveraging the 'broom' and 'ggplot2' packages.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports broom, changepoint, cli, dplyr, GA, ggplot2, lifecycle, memoise, methods, patchwork, purrr, scales, stringr, tibble, tidyr, tsibble, vctrs, wbs, xts, zoo

Depends R (>= 4.1)

LazyData true

Suggests bench, knitr, here, readr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

URL https://beanumber.github.io/tidychangepoint/

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

RemoteUrl https://github.com/beanumber/tidychangepoint

RemoteRef HEAD

RemoteSha 1f7c4a61052db82e641b435629d379270c137537

Contents

`as.model` *Convert, retrieve, or verify a model object*

Description

Convert, retrieve, or verify a model object

Usage

```
as.model(object, ...)

## Default S3 method:
as.model(object, ...)

## S3 method for class 'tidycpt'
as.model(object, ...)

is_model(x, ...)
```

Arguments

`object`	A tidycpt object, typically returned by `segment()`
`...`	currently ignored
`x`	An object, typically returned by `fit_*()`

Details

tidycpt objects have a `model` component. The functions documented here are convenience utility functions for working with the `model` components. `as.model()` is especially useful in pipelines to avoid having to use the `$` or `[` notation for subsetting.

When applied to a tidycpt object, `as.model()` simply returns the `model` component of that object. However, when applied to a `segmenter` object, `as.model()` attempts to converts that object into a mod_cpt model object.

`is_model()` checks to see if a model object implements all of the S3 methods necessary to be considered a model.

Value

- `as.model()` returns a mod_cpt model object
- `is_model()` a `logical` vector of length 1

See Also

Other tidycpt-generics: as.segmenter(), changepoints(), diagnose(), fitness(), model_name()

Examples

```
# Segment a time series using PELT
x <- segment(CET, method = "pelt")

# Retrieve the model component
x |>
  as.model()

# Explicitly convert the segmenter to a model
x |>
  as.segmenter() |>
  as.model()

# Is that model valid?
x |>
  as.model() |>
  is_model()


# Fit a model directly, without using [segment()]
x <- fit_nhpp(CET, tau = 330)
is_model(x)
```

---

as.segmenter — *Convert, retrieve, or verify a segmenter object*

Description

Convert, retrieve, or verify a segmenter object

Usage

```
as.segmenter(object, ...)

as.seg_cpt(object, ...)

## S3 method for class 'seg_basket'
as.seg_cpt(object, ...)

## S3 method for class 'seg_cpt'
as.seg_cpt(object, ...)

## S3 method for class 'tidycpt'
as.segmenter(object, ...)
```

```
## S3 method for class 'ga'
as.seg_cpt(object, ...)

## S3 method for class 'cpt'
as.seg_cpt(object, ...)

## S3 method for class 'wbs'
as.seg_cpt(object, ...)

is_segmenter(object, ...)
```

Arguments

`object`	A tidycpt or `segmenter` object
`...`	Arguments passed to methods

Details

tidycpt objects have a `segmenter` component (that is typically created by a class to `segment()`). The functions documented here are convenience utility functions for working with the `segmenter` components. `as.segmenter()` is especially useful in pipelines to avoid having to use the `$` or `[` notation for subsetting.

`as.segmenter()` simply returns the segmenter of a `tidycpt` object.

`as.seg_cpt()` takes a wild-caught `segmenter` object of arbitrary class and converts it into a seg_cpt object.

`is_segmenter()` checks to see if a segmenter object implements all of the S3 methods necessary to be considered a segmenter.

Value

- `as.segmenter()` returns the `segmenter` object of a `tidycpt` object. Note that this could be of any class, depending on the class returned by the segmenting function.
- `as.seg_cpt()` returns a seg_cpt object
- `is_segmenter()` a `logical` vector of length 1

See Also

Other tidycpt-generics: `as.model()`, `changepoints()`, `diagnose()`, `fitness()`, `model_name()`

Other segmenter-functions: `fitness()`, `model_args()`, `seg_params()`

Examples

```
# Segment a time series using PELT
x <- segment(CET, method = "pelt")

# Return the segmenter component
```

```
x |>
  as.segmenter()

# Note the class of this object could be anything
x |>
  as.segmenter() |>
  class()

# Convert the segmenter into the standardized seg_cpt class
x |>
  as.segmenter() |>
  as.seg_cpt()

# Is the segmenter valid?
x |>
  as.segmenter() |>
  is_segmenter()
```

---

as_year *Convert a date into a year*

Description

Convert a date into a year

Usage

```
as_year(x)
```

Arguments

x	an object coercible into a base::Date. See `base::as.Date()`.

Value

A `character` vector representing the years of the input

Examples

```
# Retrieve only the year
as_year("1988-01-01")
```

`binary2tau` *Convert changepoint sets to binary strings*

Description

Convert changepoint sets to binary strings

Usage

```
binary2tau(x)

tau2binary(tau, n)
```

Arguments

`x`	A binary string that encodes a changepoint set. See `GA::gabin_Population()`.
`tau`	a numeric vector of changepoint indices
`n`	the length of the original time series

Details

In order to use `GA::ga()` in a genetic algorithm, we need to encoude a changepoint set as a binary string.

`binary2tau()` takes a binary string representation of a changepoint set and converts it into a set of changepoint indices.

`tau2binary()` takes a set of changepoint indices the number of observations in the time series and converts them into a binary string representation of that changepoint set.

Value

- `binary2tau()`: an `integer` vector
- `tau2binary()`: an `integer` vector of length `n`

Examples

```
# Recover changepoint set indices from binary strings
binary2tau(c(0, 0, 1, 0, 1))
binary2tau(round(runif(10)))

# Recover binary strings from changepoint set indices
tau2binary(c(7, 17), n = 24)
tau2binary(binary2tau(c(0, 0, 1, 1, 0, 1)), n = 6)
```

BMDL *Bayesian Maximum Descriptive Length*

Description

Generic function to compute the Bayesian Maximum Descriptive Length for a changepoint detection model.

Usage

```
BMDL(object, ...)

## Default S3 method:
BMDL(object, ...)

## S3 method for class 'nhpp'
BMDL(object, ...)
```

Arguments

`object`	any object from which a log-likelihood value, or a contribution to a log-likelihood value, can be extracted.
`...`	some methods for this generic function require additional arguments.

Details

Currently, the BMDL function is only defined for the NHPP model (see `fit_nhpp()`). Given a changepoint set τ, the BMDL is:

$$BMDL(\tau, NHPP(y|\hat{\theta}_\tau) = P_{MDL}(\tau) - 2\ln L_{NHPP}(y|\hat{\theta}_\tau) - 2\ln g(\hat{\theta}_\tau)$$

where $P_{MDL}(\tau)$ is the `MDL()` penalty.

Value

A `double` vector of length 1

See Also

Other penalty-functions: `MBIC()`, `MDL()`

Examples

```
# Compute the BMDL
BMDL(fit_nhpp(DataCPSim, tau = NULL))
BMDL(fit_nhpp(DataCPSim, tau = c(365, 830)))
```

`bogota_pm` *Particulate matter in Bogotá, Colombia*

Description

Particulate matter of less than 2.5 microns of diameter in Bogotá, Colombia.

Usage

```
bogota_pm
```

Format

An object of class `xts` (inherits from `zoo`) with 1096 rows and 1 columns.

Details

Daily readings from 2018-2020 are included.

Examples

```
class(bogota_pm)
```

`build_gabin_population` *Initialize populations in genetic algorithms*

Description

Build an initial population set for genetic algorithms

Usage

```
build_gabin_population(x, ...)

log_gabin_population(x, ...)
```

Arguments

`x`	a numeric vector coercible into a stats::ts object
`...`	arguments passed to methods

Details

Genetic algorithms require a method for randomly generating initial populations (i.e., a first generation). The default method used by GA::ga() for changepoint detection is usually GA::gabin_Population(), which selects candidate changepoints uniformly at random with probability 0.5. This leads to an initial population with excessively large candidate changepoint sets (on the order of $n/2$), which makes the genetic algorithm slow.

- build_gabin_population() takes a ts object and runs several fast changepoint detection algorithms on it, then sets the initial probability to 3 times the average value of the size of the changepoint sets returned by those algorithms. This is a conservative guess as to the likely size of the optimal changepoint set.
- log_gabin_population() takes a ts object and sets the initial probability to the natural logarithm of the length of the time series.

Value

A function that can be passed to the population argument of GA::ga() (through segment_ga())

See Also

GA::gabin_Population(), segment_ga()

Examples

```
# Build a function to generate the population
f <- build_gabin_population(CET)

# Segment the time series using the population generation function
segment(CET, method = "ga", population = f, maxiter = 5)
f <- log_gabin_population(CET)
segment(CET, method = "ga", population = f, maxiter = 10)
```

CET — *Hadley Centre Central England Temperature*

Description

Mean annual temperatures in Central England

Usage

```
CET
```

Format

An object of class xts (inherits from zoo) with 362 rows and 1 columns.

Details

The CET time series is perhaps the longest instrumental record of surface temperatures in the world, commencing in 1659 and spanning 362 years through 2020. The CET series is a benchmark for European climate studies, as it is sensitive to atmospheric variability in the North Atlantic (Parker et al. 1992). This record has been previously analyzed for long-term changes (Plaut et al. 1995; Harvey and Mills 2003; Hillebrand and Proietti 2017); however, to our knowledge, no detailed changepoint analysis of it has been previously conducted. The length of the CET record affords us the opportunity to explore a variety of temperature features.

Source

https://www.metoffice.gov.uk/hadobs/hadcet/

References

- Shi, et al. (2022, doi:10.1175/JCLID210489.1),
- Parker, et al. (1992, doi:10.1002/joc.3370120402)

`changepoints` *Extract changepoints*

Description

Retrieve the indices of the changepoints identified by an algorithm or model.

Usage

```
changepoints(x, ...)

## Default S3 method:
changepoints(x, ...)

## S3 method for class 'mod_cpt'
changepoints(x, ...)

## S3 method for class 'seg_basket'
changepoints(x, ...)

## S3 method for class 'seg_cpt'
changepoints(x, ...)

## S3 method for class 'tidycpt'
changepoints(x, use_labels = FALSE, ...)

## S3 method for class 'ga'
changepoints(x, ...)
```

```
## S3 method for class 'cpt'
changepoints(x, ...)

## S3 method for class 'wbs'
changepoints(x, ...)
```

Arguments

x	A tidycpt, `segmenter`, or mod_cpt object
...	arguments passed to methods
use_labels	return the time labels for the changepoints instead of the indices.

Details

tidycpt objects, as well as their `segmenter` and `model` components, implement `changepoints()` methods.

Note that this function is not to be confused with `wbs::changepoints()`, which returns different information.

For the `default` method, `changepoints()` will attempt to return the `cpt_true` attribute, which is set by `test_set()`.

Value

a numeric vector of changepoint indices, or, if `use_labels` is `TRUE`, a `character` of time labels.

See Also

`wbs::changepoints()`

Other tidycpt-generics: `as.model()`, `as.segmenter()`, `diagnose()`, `fitness()`, `model_name()`

Examples

```
cpts <- segment(DataCPSim, method = "ga", maxiter = 5)
changepoints(cpts$segmenter)

cpts <- segment(DataCPSim, method = "wbs")
changepoints(cpts$segmenter)
```

`compare_models` *Compare various models or algorithms for a given changepoint set*

Description

Compare various models or algorithms for a given changepoint set

Usage

```
compare_models(x, ...)

compare_algorithms(x, ...)
```

Arguments

`x`	A tidycpt object
`...`	currently ignored

Details

A tidycpt object has a set of changepoints returned by the algorithm that segmented the time series. That changepoint set was obtained using a specific model. Treating this changepoint set as fixed, the `compare_models()` function fits several common changepoint models to the time series and changepoint set, and returns the results of `glance()`. Comparing the fits of various models could lead to improved understanding.

Alternatively, `compare_algorithms()` runs several fast changepoint detection algorithms on the original time series, and consolidates the results.

Value

A tibble::tbl_df

Examples

```
# Segment a times series using PELT
x <- segment(CET, method = "pelt")

# Compare models
compare_models(x)

# Compare algorithms
compare_algorithms(x)
```

cut_inclusive *Use a changepoint set to break a time series into regions*

Description

Use a changepoint set to break a time series into regions

Usage

```
cut_inclusive(x, tau)

split_by_tau(x, tau)
```

Arguments

x	A numeric vector
tau	a numeric vector of changepoint indices

Details

A changepoint set tau of length k breaks a time series of length n into $k + 1$ non-empty regions. These non-empty regions can be defined by half-open intervals, starting with 1 and ending with $n + 1$.

cut_inclusive() splits a set of indices into a base::factor() of half-open intervals

split_by_tau() splits a time series into a named base::list() of numeric vectors

Value

- cut_inclusive() a base::factor() of half-open intervals
- split_by_tau() a named base::list() of numeric vectors

Examples

```
n <- length(CET)

# Return a factor of intervals
cut_inclusive(1:n, tau = pad_tau(c(42, 81, 330), n))

# Return a list of observations
split_by_tau(DataCPSim, c(365, 826))
```

`DataCPSim` *Simulated time series data*

Description

Randomly-generated time series data, using the `stats::rlnorm()` function.

- For `rlnorm_ts_1`, there is one changepoint located at 826.
- For `rlnorm_ts_2`, there are two changepoints, located at 366 and 731.
- For `rlnorm_ts_3`, there are three changepoints, located at 548, 823, and 973.

Usage

```
DataCPSim

rlnorm_ts_1

rlnorm_ts_2

rlnorm_ts_3
```

Format

An object of class `numeric` of length 1096.

An object of class `ts` of length 1096.

An object of class `ts` of length 1096.

An object of class `ts` of length 1096.

Details

- `DataCPSim`: Simulated time series of the same length as bogota_pm.

See Also

bogota_pm

`stats::ts()`, `test_set()`

Examples

```
plot(rlnorm_ts_1)
plot(rlnorm_ts_2)
plot(rlnorm_ts_3)
changepoints(rlnorm_ts_1)
```

deg_free *Retrieve the degrees of freedom from a* logLik *object*

Description

Retrieve the degrees of freedom from a logLik object

Usage

```
deg_free(x)
```

Arguments

x	An object that implements a method for stats::logLik().

Value

The df attribute of the stats::logLik() of the given object.

Examples

```
# Retrieve the degrees of freedom model a changepoint model
DataCPSim |>
  segment() |>
  as.model() |>
  deg_free()
```

diagnose *Diagnose the fit of a segmented time series*

Description

Depending on the input, this function returns a diagnostic plot.

Usage

```
diagnose(x, ...)

## S3 method for class 'mod_cpt'
diagnose(x, ...)

## S3 method for class 'seg_basket'
diagnose(x, ...)

## S3 method for class 'tidycpt'
```

```
diagnose(x, ...)

## S3 method for class 'nhpp'
diagnose(x, ...)
```

Arguments

`x`	A tidycpt object, or a `model` or `segmenter`
`...`	currently ignored

Value

A `ggplot2::ggplot()` object

See Also

Other tidycpt-generics: `as.model()`, `as.segmenter()`, `changepoints()`, `fitness()`, `model_name()`

Examples

```
# For meanshift models, show the distribution of the residuals by region
fit_meanshift_norm(CET, tau = 330) |>
  diagnose()

# For Coen's algorithm, show the histogram of changepoint selections
x <- segment(DataCPSim, method = "coen", num_generations = 3)
x |>
  as.segmenter() |>
  diagnose()


# Show various iterations of diagnostic plots
diagnose(segment(DataCPSim))
diagnose(segment(DataCPSim, method = "single-best"))
diagnose(segment(DataCPSim, method = "pelt"))

# Show diagnostic plots for test sets
diagnose(segment(test_set()))
diagnose(segment(test_set(n = 2, sd = 4), method = "pelt"))

# For NHPP models, show the growth in the number of exceedances
diagnose(fit_nhpp(DataCPSim, tau = 826))
diagnose(fit_nhpp(DataCPSim, tau = 826, threshold = 200))
```

`exceedances` *Compute exceedances of a threshold for a time series*

Description

Compute exceedances of a threshold for a time series

Usage

```
exceedances(x, ...)

## Default S3 method:
exceedances(x, ...)

## S3 method for class 'nhpp'
exceedances(x, ...)

## S3 method for class 'ts'
exceedances(x, ...)

## S3 method for class 'double'
exceedances(x, threshold = mean(x, na.rm = TRUE), ...)
```

Arguments

`x`	a numeric vector coercible into a stats::ts object
`...`	arguments passed to methods
`threshold`	A value above which to exceed. Default is the `mean()`

Value

An ordered `integer` vector giving the indices of the values of `x` that exceed the `threshold`.

Examples

```
# Retrieve exceedances of the series mean
fit_nhpp(DataCPSim, tau = 826) |>
  exceedances()

# Retrieve exceedances of a supplied threshold
fit_nhpp(DataCPSim, tau = 826, threshold = 200) |>
  exceedances()
```

file_name — *Obtain a descriptive filename for a tidycpt object*

Description

Obtain a descriptive filename for a tidycpt object

Usage

```
file_name(x, data_name_slug = "data")
```

Arguments

x	A tidycpt object
data_name_slug	character string that will identify the data set used in the file name

Details

`file_name()` generates a random, unique string indicating the algorithm and `fitness()` for a tidycpt object.

Value

A `character` string giving a unique file name.

Examples

```
# Generate a unique name for the file
DataCPSim |>
  segment(method = "pelt") |>
  file_name()
```

fitness — *Retrieve the optimal fitness (or objective function) value used by an algorithm*

Description

Retrieve the optimal fitness (or objective function) value used by an algorithm

Usage

```
fitness(object, ...)

## S3 method for class 'seg_basket'
fitness(object, ...)

## S3 method for class 'seg_cpt'
fitness(object, ...)

## S3 method for class 'tidycpt'
fitness(object, ...)

## S3 method for class 'ga'
fitness(object, ...)

## S3 method for class 'cpt'
fitness(object, ...)

## S3 method for class 'wbs'
fitness(object, ...)
```

Arguments

`object`	A `segmenter` object.
`...`	currently ignored

Details

Segmenting algorithms use a **fitness** metric, typically through the use of a penalized objective function, to determine which changepoint sets are more or less optimal. This function returns the value of that metric for the changepoint set implied by the object provided.

Value

A named `double` vector with the fitness value.

See Also

Other tidycpt-generics: `as.model()`, `as.segmenter()`, `changepoints()`, `diagnose()`, `model_name()`

Other segmenter-functions: `as.segmenter()`, `model_args()`, `seg_params()`

Examples

```
# Segment a times series using a genetic algorithm
x <- segment(DataCPSim, method = "ga", maxiter = 10)

# Retrieve its fitness value
fitness(x)
```

```
# Segment a time series using Wild Binary Segmentation
x <- segment(DataCPSim, method = "wbs")

# Retrive its fitness
fitness(x)
```

`fit_lmshift` *Regression-based model fitting*

Description

Regression-based model fitting

Usage

```
fit_lmshift(x, tau, deg_poly = 0, ...)

fit_lmshift_ar1(x, tau, ...)

fit_trendshift(x, tau, ...)

fit_trendshift_ar1(x, tau, ...)
```

Arguments

`x`	A time series
`tau`	a set of indices representing a changepoint set
`deg_poly`	integer indicating the degree of the polynomial spline to be fit. Passed to `stats::poly()`.
`...`	arguments passed to `stats::lm()`

Details

These model-fitting functions use `stats::lm()` to fit the corresponding regression model to a time series, using the changepoints specified by the `tau` argument. Each changepoint is treated as a categorical fixed-effect, while the `deg_poly` argument controls the degree of the polynomial that interacts with those fixed-effects. For example, setting `deg_poly` equal to 0 will return the same model as calling `fit_meanshift_norm()`, but the latter is faster for larger changepoint sets because it doesn't have to fit all of the regression models.

Setting `deg_poly` equal to 1 fits the `trendshift` model.

- `fit_lmshift_ar1()`: will apply auto-regressive lag 1 errors
- `fit_trendshift()`: will fit a line in each region
- `fit_trendshift_ar1()`: will fit a line in each region and autoregress lag 1 errors

Value

A mod_cpt object

See Also

Other model-fitting: `fit_meanshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_args()`, `model_name()`, `new_fun_cpt()`, `whomademe()`

Examples

```
# Manually specify a changepoint set
tau <- c(365, 826)

# Fit the model
mod <- fit_lmshift(DataCPSim, tau)

# Retrieve model parameters
logLik(mod)
deg_free(mod)

# Manually specify a changepoint set
cpts <- c(1700, 1739, 1988)
ids <- time2tau(cpts, as_year(time(CET)))

# Fit the model
mod <- fit_lmshift(CET, tau = ids)

# View model parameters
glance(mod)
glance(fit_lmshift(CET, tau = ids, deg_poly = 1))
glance(fit_lmshift_ar1(CET, tau = ids))
glance(fit_lmshift_ar1(CET, tau = ids, deg_poly = 1))
glance(fit_lmshift_ar1(CET, tau = ids, deg_poly = 2))

# Empty changepoint sets are allowed
fit_lmshift(CET, tau = NULL)

# Duplicate changepoints are removed
fit_lmshift(CET, tau = c(42, 42))
```

`fit_meanshift` *Fast implementation of meanshift model*

Description

Fast implementation of meanshift model

Usage

```
fit_meanshift(x, tau, distribution = "norm", ...)

fit_meanshift_norm(x, tau, ...)

fit_meanshift_lnorm(x, tau, ...)

fit_meanshift_norm_ar1(x, tau, ...)
```

Arguments

x	A time series
tau	a set of indices representing a changepoint set
distribution	A character indicating the distribution of the data. Should match R distribution function naming conventions (e.g., "norm" for the Normal distribution, etc.)
...	arguments passed to `stats::lm()`

Details

`fit_meanshift_norm()` returns the same model as `fit_lmshift()` with the `deg_poly` argument set to 0. However, it is faster on large changepoint sets.

`fit_meanshift_lnorm()` fit the meanshift model with the assumption of log-normally distributed data.

`fit_meanshift_norm_ar1()` applies autoregressive errors.

Value

A mod_cpt object.

Author(s)

Xueheng Shi, Ben Baumer

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_args()`, `model_name()`, `new_fun_cpt()`, `whomademe()`

Examples

```
# Manually specify a changepoint set
tau <- c(365, 826)

# Fit the model
mod <- fit_meanshift_norm_ar1(DataCPSim, tau)

# View model parameters
logLik(mod)
```

```
deg_free(mod)

# Manually specify a changepoint set
cpts <- c(1700, 1739, 1988)
ids <- time2tau(cpts, as_year(time(CET)))

# Fit the model
mod <- fit_meanshift_norm(CET, tau = ids)

# Review model parameters
glance(mod)

# Fit an autoregressive model
mod <- fit_meanshift_norm_ar1(CET, tau = ids)

# Review model parameters
glance(mod)
```

---

fit_meanvar — *Fit a model for mean and variance*

Description

Fit a model for mean and variance

Usage

```
fit_meanvar(x, tau, ...)
```

Arguments

x	A time series
tau	a set of indices representing a changepoint set
...	currently ignored

Details

In a mean-variance model, both the means and variances are allowed to vary across regions. Thus, this model fits a separate μ_j and σ_j for each region j.

Value

A mod_cpt object.

See Also

changepoint::cpt.meanvar()

Other model-fitting: fit_lmshift(), fit_meanshift(), fit_nhpp(), model_args(), model_name(), new_fun_cpt(), whomademe()

Examples

```
# Fit a mean-variance model
fit_meanvar(CET, tau = c(42, 330))
```

`fit_nhpp` *Fit a non-homogeneous Poisson process model to the exceedances of a time series.*

Description

Fit a non-homogeneous Poisson process model to the exceedances of a time series.

Usage

```
fit_nhpp(x, tau, ...)
```

Arguments

`x`	A time series
`tau`	A vector of changepoints
`...`	currently ignored

Details

Any time series can be modeled as a non-homogeneous Poisson process of the locations of the exceedances of a threshold in the series. This function uses the BMDL criteria to determine the best fit parameters for each region defined by the changepoint set `tau`.

Value

An nhpp object, which inherits from mod_cpt.

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanshift()`, `fit_meanvar()`, `model_args()`, `model_name()`, `new_fun_cpt()`, `whomademe()`

Examples

```
# Fit an NHPP model using the mean as a threshold
fit_nhpp(DataCPSim, tau = 826)

# Fit an NHPP model using other thresholds
fit_nhpp(DataCPSim, tau = 826, threshold = 20)
fit_nhpp(DataCPSim, tau = 826, threshold = 200)

# Fit an NHPP model using changepoints determined by PELT
fit_nhpp(DataCPSim, tau = changepoints(segment(DataCPSim, method = "pelt")))
```

`iweibull` *Weibull distribution functions*

Description

Weibull distribution functions

Usage

```
iweibull(x, shape, scale = 1)

mweibull(x, shape, scale = 1)

parameters_weibull(...)
```

Arguments

`x`	A numeric vector
`shape`	Shape parameter for Weibull distribution. See `stats::dweibull()`.
`scale`	Scale parameter for Weibull distribution. See `stats::dweibull()`.
`...`	currently ignored

Details

Intensity function for the Weibull distribution.

$$iweibull(x) = \left(\frac{shape}{scale}\right) \cdot \left(\frac{x}{scale}\right)^{shape-1}$$

Mean intensity function for the Weibull distribution.

$$mweibull(x) = \left(\frac{x}{scale}\right)^{shape}$$

`parameters_weibull()` returns a `list()` with two components: `shape` and `scale`, each of which is a `list()` of distribution parameters. These parameters are used to define the prior distributions for the hyperparameters.

Value

A numeric vector

See Also

`stats::dweibull()`

`stats::dgamma()`

Examples

```
# Compute the intensities and plot them
iweibull(1, shape = 1, scale = 1)
plot(x = 1:10, y = iweibull(1:10, shape = 2, scale = 2))

# Compute various values of the distribution
mweibull(1, shape = 1, scale = 1)
plot(x = 1:10, y = mweibull(1:10, shape = 1, scale = 1))
plot(x = 1:10, y = mweibull(1:10, shape = 1, scale = 2))
plot(x = 1:10, y = mweibull(1:10, shape = 0.5, scale = 2))
plot(x = 1:10, y = mweibull(1:10, shape = 0.5, scale = 100))
plot(x = 1:10, y = mweibull(1:10, shape = 2, scale = 2))
plot(x = 1:10, y = mweibull(1:10, shape = 2, scale = 100))

# Generate prior distribution hyperparameters
parameters_weibull()
```

MBIC *Modified Bayesian Information Criterion*

Description

Generic function to compute the Modified Bayesian Information Criterion for a changepoint detection model.

Usage

```
MBIC(object, ...)

## Default S3 method:
MBIC(object, ...)

## S3 method for class 'logLik'
MBIC(object, ...)
```

Arguments

`object`	any object from which a log-likelihood value, or a contribution to a log-likelihood value, can be extracted.
`...`	some methods for this generic function require additional arguments.

Value

A `double` vector of length 1

References

Zhang and Seigmmund (2007) for MBIC: doi:10.1111/j.15410420.2006.00662.x

See Also

stats::BIC()

Other penalty-functions: BMDL(), MDL()

---

mcdf — *Cumulative distribution of the exceedances of a time series*

Description

Cumulative distribution of the exceedances of a time series

Usage

```
mcdf(x, dist = "weibull")
```

Arguments

x	An NHPP model returned by fit_nhpp()
dist	Name of the distribution. Currently only weibull is implemented.

Value

a numeric vector of length equal to the exceedances of x

See Also

plot_intensity()

Examples

```
# Fit an NHPP model using the mean as a threshold
nhpp <- fit_nhpp(DataCPSim, tau = 826)

# Compute the cumulative exceedances of the mean
mcdf(nhpp)

# Fit an NHPP model using another threshold
nhpp <- fit_nhpp(DataCPSim, tau = 826, threshold = 200)

# Compute the cumulative exceedances of the threshold
mcdf(nhpp)
```

mde_rain *Rainfall in Medellín, Colombia*

Description

Rainfall in Medellín, Colombia

Usage

```
mde_rain

mde_rain_monthly
```

Format

An object of class spec_tbl_df (inherits from tbl_df, tbl, data.frame) with 185705 rows and 8 columns.

An object of class xts (inherits from zoo) with 444 rows and 1 columns.

Details

Daily rainfall measurements for 13 different weather stations positioned around Medellín, Colombia. Variables:

- station_id:
- lat, long: latitude and longitude for the weather station
- date, year, month, day: date variables
- rainfall: daily rainfall (in cubic centimeters) as measured by the weather station

- mean_rainfall: average rainfall across all weather stations

References

OpenStreetMap

MDL *Maximum Descriptive Length*

Description

Generic function to compute the Maximum Descriptive Length for a changepoint detection model.

Usage

```
MDL(object, ...)

## Default S3 method:
MDL(object, ...)

## S3 method for class 'logLik'
MDL(object, ...)
```

Arguments

`object`	any object from which a log-likelihood value, or a contribution to a log-likelihood value, can be extracted.
`...`	some methods for this generic function require additional arguments.

Details

$$P_{MDL}(\tau) = \frac{a(\theta_\tau)}{2} \cdot \sum_{j=0}^{m} \log\left(\tau_j - \tau_{j-1}\right) + 2\ln m + \sum_{j=2}^{m} \ln \tau_j + (2 + b(\theta_\tau)) \ln n$$

where $a(\theta)$ is the number of parameters in θ that are fit in each region, and $b(\theta)$ is the number of parameters fit to the model as a whole.

These quantites should be `base::attributes()` of the object returned by `logLik()`.

Value

A double vector of length 1

See Also

Other penalty-functions: `BMDL()`, `MBIC()`

Examples

```
MDL(fit_meanshift_norm_ar1(CET, tau = c(42, 330)))
MDL(fit_trendshift(CET, tau = c(42, 81, 330)))
```

mlb_hrs — *Differences between leagues in Major League Baseball*

Description

The difference in home runs hit per plate appearance between the American League and the National League from 1925 to 2022.

Usage

```
mlb_hrs
```

Format

An object of class `xts` (inherits from `zoo`) with 98 rows and 1 columns.

model_args — *Retrieve the arguments that a model-fitting function used*

Description

Retrieve the arguments that a model-fitting function used

Usage

```
model_args(object, ...)

## Default S3 method:
model_args(object, ...)

## S3 method for class 'seg_cpt'
model_args(object, ...)

## S3 method for class 'ga'
model_args(object, ...)

## S3 method for class 'cpt'
model_args(object, ...)

## S3 method for class 'wbs'
model_args(object, ...)
```

Arguments

`object`	A `segmenter` object.
`...`	currently ignored

Details

Every model is fit by a model-fitting function, and these functions sometimes take arguments. `model_args()` recovers the arguments that were passed to the model fitting function when it was called. These are especially important when using a genetic algorithm.

Value

A named `list` of arguments, or `NULL`

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_name()`, `new_fun_cpt()`, `whomademe()`

Other segmenter-functions: `as.segmenter()`, `fitness()`, `seg_params()`

Examples

```
# Segment a time series using Coen's algorithm
x <- segment(CET, method = "ga-coen", maxiter = 3)

# Recover the arguments passed to the model-fitting function
x |>
  as.segmenter() |>
  model_args()
```

---

model_name — *Retrieve the name of the model that a segmenter or model used*

Description

Retrieve the name of the model that a segmenter or model used

Usage

```
model_name(object, ...)

## Default S3 method:
model_name(object, ...)

## S3 method for class 'character'
model_name(object, ...)

## S3 method for class 'mod_cpt'
model_name(object, ...)

## S3 method for class 'seg_basket'
```

```
model_name(object, ...)

## S3 method for class 'seg_cpt'
model_name(object, ...)

## S3 method for class 'tidycpt'
model_name(object, ...)

## S3 method for class 'ga'
model_name(object, ...)

## S3 method for class 'cpt'
model_name(object, ...)

## S3 method for class 'wbs'
model_name(object, ...)
```

Arguments

`object`	A `segmenter` object.
`...`	currently ignored

Details

Every segmenter works by fitting a model to the data. `model_name()` returns the name of a model that can be passed to `whomademe()` to retrieve the model fitting function. These functions must begin with the prefix `fit_`. Note that the model fitting functions exist in `tidychangepoint` are are not necessarily the actual functions used by the segmenter.

Models also implement `model_name()`.

Value

A `character` vector of length 1.

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_args()`, `new_fun_cpt()`, `whomademe()`

Other tidycpt-generics: `as.model()`, `as.segmenter()`, `changepoints()`, `diagnose()`, `fitness()`

Examples

```
# Segment a time series using PELT
x <- segment(CET, method = "pelt")

# Retrieve the name of the model from the segmenter
x |>
  as.segmenter() |>
  model_name()
```

```
# What function created the model?
x |>
  model_name() |>
  whomademe()
model_name(x$segmenter)

# Retrieve the name of the model from the model
x |>
  as.model() |>
  model_name()
```

`model_variance` *Compute model variance*

Description

Compute model variance

Usage

```
model_variance(object, ...)
```

Arguments

`object`	A model object implementing `residuals()` and `nobs()`
`...`	currently ignored

Details

Using the generic functions `residuals()` and `nobs()`, this function computes the variance of the residuals.

Note that unlike `stats::var()`, it does not use $n - 1$ as the denominator.

Value

A `double` vector of length 1

`new_fun_cpt` *Class for model-fitting functions*

Description

Class for model-fitting functions

Usage

```
new_fun_cpt(x, ...)

validate_fun_cpt(x)

fun_cpt(x, ...)
```

Arguments

`x`	a `character` giving the name of a model-fitting function
`...`	currently ignored

Details

All model-fitting functions must be registered through a call to `fun_cpt()`.

All model-fitting functions must take at least three arguments:

- `x`: a time series,
- `tau`: a set of changepoint indices
- `...`: other arguments passed to methods

See `fit_meanshift_norm()`,

Value

A fun_cpt object.

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_args()`, `model_name()`, `whomademe()`

Examples

```
# Register a model-fitting function
f <- fun_cpt("fit_meanvar")

# Verify that it now has class `fun_cpt`
str(f)
```

```
# Use it
f(CET, 42)
```

new_mod_cpt *Base class for changepoint models*

Description

Create changepoint detection model objects

Usage

```
new_mod_cpt(
  x = numeric(),
  tau = integer(),
  region_params = tibble::tibble(),
  model_params = double(),
  fitted_values = double(),
  model_name = character(),
  ...
)

validate_mod_cpt(x)

mod_cpt(x, ...)
```

Arguments

`x`	a numeric vector coercible into a `ts` object
`tau`	indices of the changepoint set
`region_params`	A `tibble::tibble()` with one row for each region defined by the changepoint set `tau`. Each variable represents a parameter estimated in that region.
`model_params`	A numeric vector of parameters estimated by the model across the entire data set (not just in each region).
`fitted_values`	Fitted values returned by the model on the original data set.
`model_name`	A `character` vector giving the model's name.
`...`	currently ignored

Details

Changepoint detection models know how they were created, on what data set, about the optimal changepoint set found, and the parameters that were fit to the model. Methods for various generic reporting functions are provided.

All changepoint detection models inherit from mod_cpt: the base class for changepoint detection models. These models are created by one of the `fit_*()` functions, or by `as.model()`.

Value

A mod_cpt object

See Also

`as.model()`

Examples

```
cpt <- mod_cpt(CET)
str(cpt)
as.ts(cpt)
changepoints(cpt)
```

---

new_seg_basket *Default class for candidate changepoint sets*

Description

Default class for candidate changepoint sets

Usage

```
new_seg_basket(
  x = numeric(),
  algorithm = NA,
  cpt_list = list(),
  seg_params = list(),
  model_name = "meanshift_norm",
  penalty = "BIC",
  ...
)

seg_basket(x, ...)
```

Arguments

`x`	a numeric vector coercible into a `stats::ts()` object
`algorithm`	Algorithm used to find the changepoints
`cpt_list`	a possibly empty `list()` of candidate changepoints
`seg_params`	a possibly empty `list()` of segmenter parameters
`model_name`	character indicating the model used to find the changepoints.
`penalty`	character indicating the name of the penalty function used to find the changepoints.
`...`	currently ignored

Value

A seg_basket() object.

Examples

```
seg <- seg_basket(DataCPSim, cpt_list = list(c(365), c(330, 839)))
str(seg)
as.ts(seg)
changepoints(seg)
fitness(seg)
```

---

new_seg_cpt — *Base class for segmenters*

Description

Base class for segmenters

Usage

```
new_seg_cpt(
  x = numeric(),
  pkg = character(),
  algorithm = NA,
  changepoints = integer(),
  fitness = double(),
  seg_params = list(),
  model_name = "meanshift_norm",
  penalty = "BIC",
  ...
)

seg_cpt(x, ...)
```

Arguments

x	a numeric vector coercible into a stats::ts() object
pkg	name of the package providing the segmenter
algorithm	Algorithm used to find the changepoints
changepoints	a possibly empty list() of candidate changepoints
fitness	A named double vector whose name reflects the penalty applied
seg_params	a possibly empty list() of segmenter parameters
model_name	character indicating the model used to find the changepoints.
penalty	character indicating the name of the penalty function used to find the changepoints.
...	currently ignored

Value

A seg_cpt object.

---

pad_tau *Pad and unpad changepoint sets with boundary points*

Description

Pad and unpad changepoint sets with boundary points

Usage

```
pad_tau(tau, n)

unpad_tau(padded_tau)

is_valid_tau(tau, n)

validate_tau(tau, n)
```

Arguments

`tau`	a numeric vector of changepoint indices
`n`	the length of the original time series
`padded_tau`	Output from `pad_tau()`

Details

If a time series contains n observations, we label them from 1 to n. Neither the 1st point nor the nth point can be a changepoint, since the regions they create on one side would be empty. However, for dividing the time series into non-empty segments, we start with 1, add $n + 1$, and then divide the half-open interval $[1, n + 1)$ into half-open subintervals that define the regions.

`pad_tau()` ensures that 1 and $n + 1$ are included.

`unpad_tau()` removes 1 and $n + 1$, should they exist.

`is_valid_tau()` checks to see if the supplied set of changepoints is valid

`validate_tau()` removes duplicates and boundary values.

Value

- `pad_tau()`: an `integer` vector that starts with 0 and ends in n.
- `unpad_tau()`: an `integer` vector stripped of its first and last entries.
- `is_valid_tau()`: a `logical` if all of the entries are between 2 and $n - 1$.
- `validate_tau()`: an `integer` vector with only the `base::unique()` entries between 2 and $n - 1$, inclusive.

Examples

```
# Anything less than 2 is not allowed
is_valid_tau(0, length(DataCPSim))
is_valid_tau(1, length(DataCPSim))

# Duplicates are allowed
is_valid_tau(c(42, 42), length(DataCPSim))
is_valid_tau(826, length(DataCPSim))

# Anything greater than \eqn{n} (in this case 1096) is not allowed
is_valid_tau(1096, length(DataCPSim))
is_valid_tau(1097, length(DataCPSim))

# Anything less than 2 is not allowed
validate_tau(0, length(DataCPSim))
validate_tau(1, length(DataCPSim))
validate_tau(826, length(DataCPSim))

# Duplicates are removed
validate_tau(c(826, 826), length(DataCPSim))

# Anything greater than \eqn{n} (in this case 1096) is not allowed
validate_tau(1096, length(DataCPSim))
validate_tau(1097, length(DataCPSim))

# Fix many problems
validate_tau(c(-4, 0, 1, 4, 5, 5, 824, 1096, 1097, 182384), length(DataCPSim))
```

---

plot.tidyga *Plot GA information*

Description

Plot GA information

Usage

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

Arguments

x	A tidyga object
...	currently ignored

Value

A ggplot2::ggplot() object.

Examples

```
x <- segment(DataCPSim, method = "ga-coen", maxiter = 5)
plot(x$segmenter)
```

---

plot_best_chromosome   *Diagnostic plots for* seg_basket *objects*

Description

Diagnostic plots for seg_basket objects

Usage

```
plot_best_chromosome(x)

plot_cpt_repeated(x, i = nrow(x$basket))
```

Arguments

x	A seg_basket() object
i	index of basket to show

Details

seg_basket() objects contain baskets of candidate changepoint sets.

plot_best_chromosome() shows how the size of the candidate changepoint sets change across the generations of evolution.

plot_cpt_repeated() shows how frequently individual observations appear in the best candidate changepoint sets in each generation.

Value

A ggplot2::ggplot() object

Examples

```
# Segment a time series using Coen's algorithm
x <- segment(DataCPSim, method = "coen", num_generations = 3)

# Plot the size of the sets during the evolution
x |>
  as.segmenter() |>
  plot_best_chromosome()


# Segment a time series using Coen's algorithm
```

```
x <- segment(DataCPSim, method = "coen", num_generations = 3)

# Plot overall frequency of appearance of changepoints
plot_cpt_repeated(x$segmenter)

# Plot frequency of appearance only up to a specific generation
plot_cpt_repeated(x$segmenter, 5)
```

`plot_intensity` *Plot the intensity of an NHPP fit*

Description

Plot the intensity of an NHPP fit

Usage

```
plot_intensity(x, ...)
```

Arguments

`x`	An NHPP `model` returned by `fit_nhpp()`
`...`	currently ignored

Value

A `ggplot2::ggplot()` object

Examples

```
# Plot the estimated intensity function
plot_intensity(fit_nhpp(DataCPSim, tau = 826))

# Segment a time series using PELT
mod <- segment(bogota_pm, method = "pelt")

# Plot the estimated intensity function for the NHPP model using the
# changepoints found by PELT
plot_intensity(fit_nhpp(bogota_pm, tau = changepoints(mod)))
```

`segment` *Segment a time series using a variety of algorithms*

Description

A wrapper function that encapsulates various algorithms for detecting changepoint sets in univariate time series.

Usage

```
segment(x, method = "null", ...)

## S3 method for class 'tbl_ts'
segment(x, method = "null", ...)

## S3 method for class 'xts'
segment(x, method = "null", ...)

## S3 method for class 'numeric'
segment(x, method = "null", ...)

## S3 method for class 'ts'
segment(x, method = "null", ...)
```

Arguments

`x`	a numeric vector coercible into a stats::ts object
`method`	a character string indicating the algorithm to use. See Details.
`...`	arguments passed to methods

Details

Currently, `segment()` can use the following algorithms, depending on the value of the `method` argument:

- `pelt`: Uses the PELT algorithm as implemented in `segment_pelt()`, which wraps either `changepoint::cpt.mean()` or `changepoint::cpt.meanvar()`. The `segmenter` is of class `cpt`.
- `binseg`: Uses the Binary Segmentation algorithm as implemented by `changepoint::cpt.meanvar()`. The `segmenter` is of class `cpt`.
- `segneigh`: Uses the Segmented Neighborhood algorithm as implemented by `changepoint::cpt.meanvar()`. The `segmenter` is of class `cpt`.
- `single-best`: Uses the AMOC criteria as implemented by `changepoint::cpt.meanvar()`. The `segmenter` is of class `cpt`.
- `wbs`: Uses the Wild Binary Segmentation algorithm as implemented by `wbs::wbs()`. The `segmenter` is of class `wbs`.

- ga: Uses the Ggnetic algorithm implemented by segment_ga(), which wraps GA::ga(). The segmenter is of class tidyga.
- ga-shi: Uses the genetic algorithm implemented by segment_ga_shi(), which wraps segment_ga(). The segmenter is of class tidyga.
- ga-coen: Uses Coen's heuristic as implemented by segment_ga_coen(). The segmenter is of class tidyga. This implementation supersedes the following one.
- coen: Uses Coen's heuristic as implemented by segment_coen(). The segmenter is of class seg_basket(). Note that this function is deprecated.
- random: Uses a random basket of changepoints as implemented by segment_ga_random(). The segmenter is of class tidyga.
- manual: Uses the vector of changepoints in the tau argument. The segmenter is of class seg_cpt‘.
- null: The default. Uses no changepoints. The segmenter is of class seg_cpt‘.

Value

An object of class tidycpt.

See Also

changepoint::cpt.meanvar(), wbs::wbs(), GA::ga(), segment_ga()

Examples

```
# Segment a time series using PELT
segment(DataCPSim, method = "pelt")

# Segment a time series using PELT and the BIC penalty
segment(DataCPSim, method = "pelt", penalty = "BIC")

# Segment a time series using Binary Segmentation
segment(DataCPSim, method = "binseg", penalty = "BIC")

# Segment a time series using a random changepoint set
segment(DataCPSim, method = "random")

# Segment a time series using a manually-specified changepoint set
segment(DataCPSim, method = "manual", tau = c(826))

# Segment a time series using a null changepoint set
segment(DataCPSim)
```

`segment_ga` *Segment a time series using a genetic algorithm*

Description

Segmenting functions for various genetic algorithms

Usage

```
segment_ga(
  x,
  model_fn = fit_meanshift_norm,
  penalty_fn = BIC,
  model_fn_args = list(),
  ...
)

segment_ga_shi(x, ...)

segment_ga_coen(x, ...)

segment_ga_random(x, ...)
```

Arguments

`x`	A time series
`model_fn`	A `character` or `name` coercible into a fun_cpt function. See, for example, `fit_meanshift_norm()`.
`penalty_fn`	A function that evaluates the changepoint set returned by `model_fn`. We provide `AIC()`, `BIC()`, `MBIC()`, `MDL()`, and `BMDL()`.
`model_fn_args`	A `list()` of parameters passed to `model_fn`
`...`	arguments passed to `GA::ga()`

Details

`segment_ga()` uses the genetic algorithm in `GA::ga()` to "evolve" a random set of candidate changepoint sets, using the penalized objective function specified by `penalty_fn`. By default, the normal `meanshift` model is fit (see `fit_meanshift_norm()`) and the BIC penalty is applied.

- `segment_ga_shi()`: Shi's algorithm is the algorithm used in doi:10.1175/JCLID210489.1. Note that in order to achieve the reported results you have to run the algorithm for a really long time. Pass the values `maxiter` = 50000 and `run` = 10000 to `GA::ga()` using the dots.
- `segment_ga_coen()`: Coen's algorithm is the one used in doi:10.1007/9783031473722_20. Note that the speed of the algorithm is highly sensitive to the size of the changepoint sets under consideration, with large changepoint sets being slow. Consider setting the `population` argument to `GA::ga()` to improve performance. Coen's algorithm uses the `build_gabin_population()` function for this purpose by default.

- segment_ga_random(): Randomly select candidate changepoint sets. This is implemented as a genetic algorithm with only one generation (i.e., maxiter = 1). Note that this function uses log_gabin_population() by default.

Value

A tidyga object. This is just a GA::ga() object with an additional slot for data (the original time series) and model_fn_args (captures the model_fn and penalty_fn arguments).

References

Shi, et al. (2022, doi:10.1175/JCLID210489.1)

Taimal, et al. (2023, doi:10.1007/9783031473722_20)

See Also

build_gabin_population()

log_gabin_population()

Examples

```
# Segment a time series using a genetic algorithm
res <- segment_ga(CET, maxiter = 5)
summary(res)
str(res)
plot(res)


# Segment a time series using Shi's algorithm
x <- segment(CET, method = "ga-shi", maxiter = 5)
str(x)

# Segment a time series using Coen's algorithm
y <- segment(CET, method = "ga-coen", maxiter = 5)
changepoints(y)

# Segment a time series using Coen's algorithm and an arbitrary threshold
z <- segment(CET, method = "ga-coen", maxiter = 5,
             model_fn_args = list(threshold = 2))
changepoints(z)

## Not run:
# This will take a really long time!
x <- segment(CET, method = "ga-shi", maxiter = 500, run = 100)
changepoints(x)

# This will also take a really long time!
y <- segment(CET, method = "ga", model_fn = fit_lmshift, penalty_fn = BIC,
  popSize = 200, maxiter = 5000, run = 1000,
  model_fn_args = list(trends = TRUE),
  population = build_gabin_population(CET)
```

```
)

## End(Not run)

## Not run:
x <- segment(method = "ga-coen", maxiter = 50)

## End(Not run)

x <- segment(CET, method = "random")
```

`segment_manual` *Manually segment a time series*

Description

Segment a time series by manually inputting the changepoint set

Usage

```
segment_manual(x, tau, ...)
```

Arguments

`x`	A time series
`tau`	a set of indices representing a changepoint set
`...`	arguments passed to seg_cpt

Details

Sometimes you want to see how a manually input set of changepoints performs. This function takes a time series and a changepoint detection set as inputs and returns a seg_cpt object representing the segmenter. Note that by default `fit_meanshift_norm()` is used to fit the model and `BIC()` is used as the penalized objective function.

Value

A seg_cpt object

Examples

```
# Segment a time series manually
segment_manual(CET, tau = c(84, 330))
segment_manual(CET, tau = NULL)
```

segment_pelt *Segment a time series using the PELT algorithm*

Description

Segmenting functions for the PELT algorithm

Usage

```
segment_pelt(x, model_fn = fit_meanvar, ...)
```

Arguments

x	A time series
model_fn	A character or name coercible into a fun_cpt function. See, for example, `fit_meanshift_norm()`. The default is `fit_meanvar()`.
...	arguments passed to `changepoint::cpt.meanvar()` or `changepoint::cpt.mean()`

Details

This function wraps either `changepoint::cpt.meanvar()` or `changepoint::cpt.mean()`.

Value

A `cpt` object returned by `changepoint::cpt.meanvar()` or `changepoint::cpt.mean()`

Examples

```
# Segment a time series using PELT
res <- segment_pelt(DataCPSim)
res
str(res)

# Segment as time series while specifying a penalty function
segment_pelt(DataCPSim, penalty = "BIC")

# Segment a time series while specifying a meanshift normal model
segment_pelt(DataCPSim, model_fn = fit_meanshift_norm, penalty = "BIC")
```

`seg_params` *Retrieve parameters from a segmenter*

Description

Retrieve parameters from a segmenter

Usage

```
seg_params(object, ...)

## S3 method for class 'seg_cpt'
seg_params(object, ...)

## S3 method for class 'ga'
seg_params(object, ...)

## S3 method for class 'cpt'
seg_params(object, ...)

## S3 method for class 'wbs'
seg_params(object, ...)
```

Arguments

`object`	A segmenter object.
`...`	currently ignored

Details

Most segmenting algorithms have parameters. This function retrieves an informative set of those parameter values.

Value

A named `list` of parameters with their values.

See Also

Other segmenter-functions: `as.segmenter()`, `fitness()`, `model_args()`

Examples

```
# Segment a time series using PELT
x <- segment(CET, method = "pelt")
x |>
  as.segmenter() |>
  seg_params()
```

tau2time *Convert changepoint sets to time indices*

Description

Convert changepoint sets to time indices

Usage

```
tau2time(tau, index)

time2tau(cpts, index)
```

Arguments

tau	a numeric vector of changepoint indices
index	Index of times, typically returned by stats::time()
cpts	Time series observation labels to be converted to indices

Value

- tau2time(): a character of time labels
- time2tau(): an integer vector of changepoint indices

See Also

stats::time(), as_year()

Examples

```
# Recover the years from a set of changepoint indices
tau2time(c(42, 81, 330), index = as_year(time(CET)))

# Recover the changepoint set indices from the years
time2tau(c(1700, 1739, 1988), index = as_year(time(CET)))
```

`tbl_coef` *Format the coefficients from a linear model as a tibble*

Description

Format the coefficients from a linear model as a tibble

Usage

```
tbl_coef(mod, ...)
```

Arguments

`mod`	An `lm` model object
`...`	currently ignored

Value

A tibble::tbl_df object containing the fitted coefficients.

Examples

```
# Convert a time series into a data frame with indices
ds <- data.frame(y = as.ts(CET), t = 1:length(CET))

# Retrieve the coefficients from a null model
tbl_coef(lm(y ~ 1, data = ds))

# Retrieve the coefficients from a two changepoint model
tbl_coef(lm(y ~ (t >= 42) + (t >= 81), data = ds))

# Retrieve the coefficients from a trendshift model
tbl_coef(lm(y ~ poly(t, 1, raw = TRUE) * (t >= 42) + poly(t, 1, raw = TRUE) * (t >= 81), data = ds))

# Retrieve the coefficients from a quadratic model
tbl_coef(lm(y ~ poly(t, 2, raw = TRUE) * (t >= 42) + poly(t, 2, raw = TRUE) * (t >= 81), data = ds))
```

`test_set` *Simulate time series with known changepoint sets*

Description

Simulate time series with known changepoint sets

Usage

```
test_set(n = 1, sd = 1, seed = NULL)
```

Arguments

n	Number of true changepoints in set
sd	Standard deviation passed to `stats::rnorm()`
seed	Value passed to `base::set.seed()`

Value

A `stats::ts()` object

See Also

DataCPSim

Examples

```
x <- test_set()
plot(x)
changepoints(x)
```

---

tidycpt-class *Container class for `tidycpt` objects*

Description

Container class for `tidycpt` objects

Details

Every `tidycpt` object contains:

- `segmenter`: The object returned by the underlying changepoint detection algorithm. These can be of arbitrary class. Use `as.segmenter()` to retrieve them.
- `model`: A model object inheriting from mod_cpt, as created by `as.model()` when called on the `segmenter`.
- `elapsed_time`: The clock time that passed while the algorithm was running.
- `time_index`: If available, the labels for the time indices of the time series.

Value

A tidycpt object.

Examples

```
# Segment a time series using PELT
x <- segment(CET, method = "pelt")
class(x)
str(x)
```

whomademe *Recover the function that created a model*

Description

Recover the function that created a model

Usage

```
whomademe(x, ...)
```

Arguments

x	A `character` giving the name of a model. To be passed to `model_name()`.
...	currently ignored

Details

Model objects (inheriting from mod_cpt) know the name of the function that created them. `whomademe()` returns that function.

Value

A `function`

See Also

Other model-fitting: `fit_lmshift()`, `fit_meanshift()`, `fit_meanvar()`, `fit_nhpp()`, `model_args()`, `model_name()`, `new_fun_cpt()`

Examples

```
# Get the function that made a model
f <- whomademe(fit_meanshift_norm(CET, tau = 42))
str(f)
```

Index