tidychangepoint: a unified framework for analyzing changepoint detection in univariate time series

Benjamin S. Baumer Biviana Marcela Suarez Sierra

2024-06-01

We present tidychangepoint, a new R package for changepoint detection analysis. tidychangepoint leverages existing packages like changepoint, GA, tsibble, and broom to provide tidyverse-compliant tools for segmenting univariate time series using various changepoint detection algorithms. In addition, tidychangepoint also provides model-fitting procedures for commonly-used parametric models, tools for computing various penalized objective functions, and graphical diagnostic displays. tidychangepoint wraps both deterministic algorithms like PELT, and also flexible, randomized, genetic algorithms that can be used with any compliant model-fitting function and any penalized objective function. By bringing all of these disparate tools together in a cohesive fashion, tidychangepoint facilitates comparative analysis of changepoint detection algorithms and models.

Source: Article Notebook

1 Introduction

Let $y = \{y_1, \ldots, y_n\}$ be a series of observations made over time points $t = 1, \ldots, n$. Define $\tau = \{\tau_1, \ldots, \tau_m\} \subseteq \{y_t\}$ to be a subset of the original observations known as *changepoints*, for some $0 \leq m \leq n$. The m changepoints divide the time series into $m + 1$ regions, with the idea that the behavior of the time series within each region does not change, and the changepoints represent points in time at which the behavior of the underlying system changes in some meaningful way. If we adopt the convention that $\tau_0 = 1$ and $\tau_{m+1} = n + 1$ then the union of the set of half-open intervals $[\tau_j, \tau_{j+1})$ for $0 \leq j \leq m$ contains the time series y.

For example, Figure 1 shows the difference in home runs per plate appearance between the American and National Leagues of Major League Baseball (MLB) from 1925–2022. The introduction of the designated hitter rule in the American League in 1973 led to noticeable

differences in the rates at which batted balls became home runs relative to the National League (which did not adapt the rule until 2022), because the designated hitter employed by American League teams was always a better hitter than the pitcher (who was forced to bat in the National League). The years 1973 and 2022 provide known changepoints, because the data generating process changed before and after those years, with the American League expected to have higher home run rates during the intervening period. In Section 2 we will investigate how well several algorithms detect these distinct time periods.

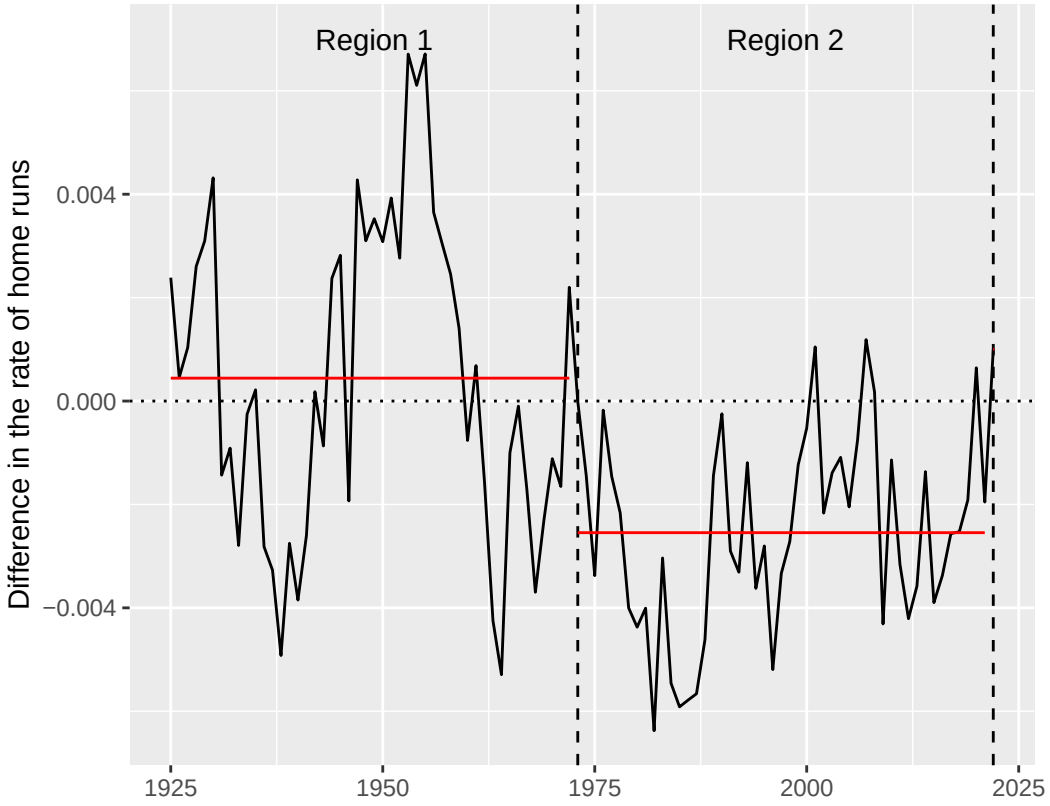


Figure 1: Difference in the rate of home runs per plate appearance between the American and National Leagues in Major League Baseball, 1925–2022. The designated hitter rule was adopted in 1973 by the American League, but not until 2022 by the National League. Note how, during the period from 1973 to 2022 when only the American League employed a designated hitter, the difference in home run rate was nearly always negative.

Source: Article Notebook

As there as 2^n possible changepoint sets, the Changepoint Detection Problem (CDP)—which is the problem of finding the optimal changepoint set (according to some metric)—has attracted

considerable attention in the literature. Aminikhanghahi and Cook (2017) provide an excellent survey of relevant results. In unsupervised settings, there is no known true changepoint set, and various models, metrics, and algorithms have been proposed, some of which are in widespread use. We briefly highlight the previous work most relevant to ours in Section 1.1, and review existing software implementations of these algorithms in Section 1.2. In any case, all attempts to solve the CDP considered in this paper involve three essential components: 1) a parametric model $M(y|\theta)$ that describes the data-generating process of the time series; 2) an algorithm A to search the exponentially large space of possible changepoint sets; and 3) a metric $f : (\tau, M) \to \mathbb{R}$ (a penalized objective function) for evaluating the suitability of any given changepoint set. We define $\tau^*_{f,M,\theta}$ to be the changepoint set that minimizes the value of the penalized objective function f on the model M with parameters θ over the space of all possible changepoint sets. That is,

$$\tau^*_{f,M,\theta_\tau} = \min_{\tau \in \subseteq \{y_t\}} f(\tau, M(y|\theta_\tau))$$

Note that the value of the parameters θ depend on the changepoint set τ, so we may write θ_τ.

For example, perhaps the simplest and most common parametric model for changepoint detection is the normal (Gaussian) *meanshift* model, in which the time series is assumed to be generated by random variables $Y_j \sim N(\mu_j, \sigma^2)$ for each region $1 \leq j \leq m+1$. Thus the series mean μ_j is constant within each of the $m+1$ regions, but varies across the regions, while the variance σ^2 is constant over the entire time series. We fit the normal meanshift model using Maximium Likelihood Estimation (MLE), resulting in the estimated parameters $\hat{\theta} = \{\hat{\sigma}^2, \hat{\mu}_0, \ldots, \hat{\mu}_m\}$. If we choose the Bayesian Information Criterion (BIC) (Schwarz 1978) as our penalty, then the penalized objective function becomes:

$$BIC(\tau, M(y|\hat{\theta}_\tau)) = k_M(\tau) \ln n - 2 \ln L_M(y|\hat{\theta}_\tau),$$

where $k_M(\tau)$ is the number of parameters estimated by the model (in this case, $m+2$) and $L_M(y|\hat{\theta}_\tau)$ is the likelihood function for M, which in this case is (see Li and Lund (2012) for details):

$$L_M(y|\hat{\theta}_\tau) = -\frac{n}{2}(\ln \hat{\sigma}^2 + 1 + \ln 2\pi)\,.$$

Thus, as applied to the baseball data mentioned above, $\tau^*_{BIC,meanshift,\hat{\theta}}$ is the changepoint set that minimizes the BIC as applied to the normal meanshift model with estimated parameters $\hat{\theta}$ over all possible changepoint sets. While in general, the true value of $\tau^*_{BIC,meanshift,\hat{\theta}}$ is unknown, under reasonable assumptions about the distribution of the changepoints the PELT algorithm (Killick, Fearnhead, and Eckley 2012) will find the exact value in polynomial time.

More generally, given a time series y, a parametric model $M(y|\theta)$, and a penalized objective function f, the goal of any changepoint detection algorithm A is to search the exponentially large space of possible changepoints sets for the one τ^* that minimizes the value of $f(\tau, M(y|\hat{\theta}_\tau))$. Of course, in the process it must also estimate $\hat{\theta}$.

1.1 Existing changepoint detection algorithms

Aminikhanghahi and Cook (2017) provide a comprehensive survey of different models for changepoint detection in time series. This includes classifications of offline vs. online algorithms, supervised vs. unsupervised problems, computational concerns, robustness, open problems, and the distribution of changepoints within long time series. Guédon (2015) quantifies the uncertainty in the segmentation of diverse sets of changepoints, using both Bayesian and non-Bayesian techniques.

The Pruned Exact Linear Time (PELT) algorithm developed by Killick, Fearnhead, and Eckley (2012) builds on a dynamic programming algorithm from Jackson et al. (2005) to offer a linear time, exact algorithm for changepoint detection. PELT offers order of magnitude improvements in computational cost (i.e., $O(n)$ vs. the $O(n \log n)$ performance of the binary segmentation algorithm (Scott and Knott 1974) and the $O(n^3)$ performance of the segmented neighborhood algorithm (Auger and Lawrence 1989)), and is capable of optimizing over different parametric models (e.g., meanvar, meanshift, etc.) and different penalized objective functions (e.g., BIC, mBIC, etc.). However, while the assumptions that PELT requires for an optimal solution are mild (namely, that the number of changepoints grows linearly with the length of the time series), it does not succeed in all cases, nor can it work with all models and penalty functions.

Li and Lund (2012) illustrate how a genetic algorithm can provide good results by employing Darwinian principles (e.g., survival of the fittest) to search the space of all possible changepoint sets intelligently. For a given parametric model and penalized objective function, a genetic algorithm starts with an initial "generation" of (usually randomly generated) candidate changepoint sets, evaluates them all using the penalized objective function, then probabilistically "mates" the most promising candidate changepoint sets to produce a new generation of the same size. This process is repeated until a stopping criteria is met, and the single best (fittest) changepoint set is returned. While genetic algorithms are neither exact, nor deterministic, nor computationally efficient, they are more easily generalizable than PELT, and can return competitive results in practice (Shi et al. 2022). Li and Lund (2012) also develop the notion of the Minimum Descriptive Length (MDL) as a penalty function based on information theory.

Bai and Perron (1998) use linear regression models fit by least squares to detect changepoints. In particular, they construct a framework for testing the alternative hypothesis that there is one additional changepoint. Cho and Fryzlewicz (2015) propose the Sparsified Binary Segmentation (SBS) algorithm, which uses CUSUM statistics to analyze high dimensional time series. This algorithm uses the concept of a threshold above which statistics are kept—an idea that resurfaces in Suárez-Sierra, Coen, and Taimal (2023). Cho (2016) explores the utility of a double CUSUM statistic for panel data. Hocking et al. (2013) use machine learning techniques to determine the optimal number of changepoints.

Suárez-Sierra, Coen, and Taimal (2023) detail the implementation of a changepoint detection

heuristic we rebrand as Coen's algorithm in this paper. Coen's algorithm (Section 4.3) is genetic and uses a non-homogeneous Poisson process model (Section 5.3) to model the exceedances of a threshold over time, and a Bayesian Minimum Descriptive Length (BMDL, Section 6.3) penalized objective function. Taimal, Suárez-Sierra, and Rivera (2023) discuss modifications and performance characteristics of Coen's algorithm.

1.2 Existing changepoint detection R packages

Likely the most prominent R package for changepoint detection is changepoint (Killick and Haynes 2022), which implements the algorithms PELT, Binary Segmentation, and Segmented Neighborhood. PELT employs one of three different models: meanshift, meanvar, and varshift (see Section 5). For the meanvar model (the most versatile) the data-generating model can follow a Normal, Poisson, Gamma, or Exponential distribution. Penalty functions (see Section 6) implemented by PELT include AIC, BIC, and MBIC, but not MDL.

The wbs (Baranowski and Fryzlewicz 2019) package implements the Wild Binary Segmentation and standard Binary Segmentation algorithms, the first of which is incorporated into tidychangepoint. The ggchangepoint (Yu 2022) package provides some wrapper functions and graphical tools for changepoint detection analysis that are similar to ours, but more limited in scope and usability. Nonparametric methods for changepoint detection are present in the ecp (N. A. James, Zhang, and Matteson 2023) and changepoint.np (Haynes and Killick 2022) packages. We consider these beyond our scope at this time, which is focused on parameteric models for changepoint detection. The qcc (Scrucca 2017) package is more broadly focused on quality control and statistical process control, neither of which are specific to changepoint detection.

1.3 Our contribution

In this paper, we present several substantive improvements to the existing state-of-the-art. First, the tidychangepoint package provides functionality that makes working with existing changepoint detection algorithms in a tidyverse-compliant syntax easy (Section 3). This reduces friction for users already comfortable working within the tidyverse. Second, the architecture of the package is easily extended to incorporate new changepoint detection algorithms (Section 4), new parametric models (Section 5), and new penalized objective functions (Section 6). Indeed, our careful separation of these three essential elements of changepoint detection is in and of itself clarifying. To the best of our knowledge, tidychangepoint is the first R package to make genetic algorithms for changepoint detection first-class citizens, and the flexible design allows models and penalty functions to be mixed-and-matched freely. We illustrate the utility of the package in Section 2 and Section 7 and conclude in Section 8.

2 Example of tidychangepoint applied to MLB

In this section, we illustrate how tidychangepoint assists in changepoint detection analysis. As noted in Section 1, the true, known changepoints for the MLB example are 1973 and 2022.

First, we apply the PELT algorithm for a change in means and variances. This algorithm finds two changepoints, but they are not the true changepoints.

```
mlb_pelt <- mlb_hrs |>
  segment(method = "pelt")
changepoints(mlb_pelt, use_labels = TRUE)
```

```
[1] "1946-01-01" "1959-01-01"
```

Source: Article Notebook

We can try PELT again, this time specifying a model that allows only the mean to vary across regions. Unfortunately, it detects no changepoints.

```
mlb_pelt_meanshift <- mlb_hrs |>
  segment(method = "pelt", model_fn = fit_meanshift_norm)
changepoints(mlb_pelt_meanshift)
```

```
integer(0)
```

Source: Article Notebook

Next, we try a genetic algorithm using a normal meanshift model with autoregressed errors, as employed by Shi et al. (2022) in their analysis of temperatures in Central England. This algorithm will stop after 500 generations (`maxiter`), or after 50 consecutive generations with no improvement (`run`), whichever comes first.

```
mlb_ga_shi <- mlb_hrs |>
  segment(method = "ga-shi", maxiter = 500, run = 50)
```

Source: Article Notebook

Source: Article Notebook

```
changepoints(mlb_ga_shi, use_labels = TRUE) |>
  as_year()
```

```
[1] "1931" "1947" "1960"
```

Source: Article Notebook

This algorithm finds 3 changepoints, and creates regions of time that conform somewhat to eras traditionally demarcated by baseball analysts and historians. For example, within the time period we consider, an analysis of the metric On-Base Plus Slugging Percentage (OPS) by Woltring, Rost, and Jubenville (2018) suggests the changepoints 1941, 1960, 1976, 1993, and 2005. B. James (2014) posits the changepoints 1946, 1968, and 1992.

Finally, we employ Coen's algorithm (Suárez-Sierra, Coen, and Taimal 2023). This algorithm correctly identifies the true changepoint of 2022, but this is a curious result, in that it is an endpoint of the time series, and thus does not define even a second region.

```
mlb_coen <- mlb_hrs |>
  segment(method = "ga-coen", maxiter = 500, run = 50)
```

Source: Article Notebook

Source: Article Notebook

```
changepoints(mlb_coen, use_labels = TRUE)
```

```
Date of length 0
```

Source: Article Notebook

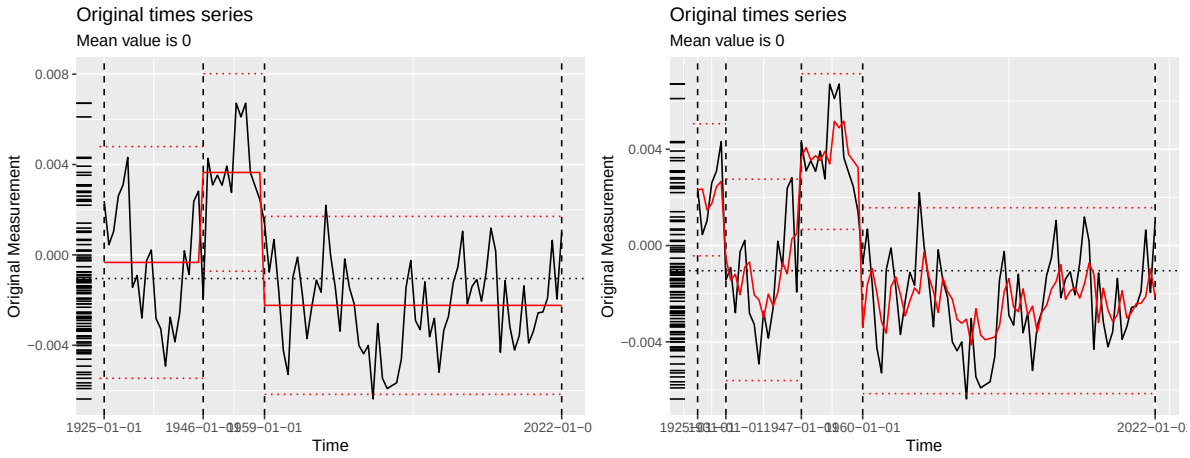
Because each of the resulting objects are of class `tidycpt` (see Section 3), comparing the results across the four algorithms is easy.

```
list(mlb_pelt, mlb_pelt_meanshift, mlb_ga_shi, mlb_coen) |>
  map(glance) |>
  list_rbind()
```

```
# A tibble: 0 x 1
# i 1 variable: elapsed_time <drtn>
```

Source: Article Notebook

In Figure 2, we compare the changepoint sets found by the PELT algorithm with the meanvar model and Shi's genetic algorithm. Both algorithms created a region from 1946–1959 (or 1947–1960), and largely missed the true changepoint of 1973. Note that because the errors in Shi's algorithm are autoregressed, the fitted values (shown in red) are not constant in each region.



(a) The PELT algorithm using the mean and variance model and the MBIC penalty.

(b) Shi's algorithm using the meanshift model and the BIC penalty.

Figure 2: Comparison of changepoint sets returned by two algorithms.

Source: Article Notebook

The remainder of this paper provides greater detail about how one can use tidychangepoint.

3 Architecture of tidychangepoint

The tidychangepoint package implements a simple, flexible structure for working with a myriad of changepoint detection algorithms. tidychangepoint makes extensive use of the S3 object-oriented class system in R. In addition to creating a few new generic functions, it provides methods for a handful of generic functions from other packages (most notably stats and broom) for three new classes of objects (`tidycpt`, `seg_cpt`, and `mod_cpt`), as well as analogous methods for existing objects created by other R packages for changepoint detection (e.g., changepoint). We provide more details in this section.

Every `tidycpt` object is created by a call to the `segment()` function. The `method` argument indicates the algorithm we want to use, and any subsequent arguments (e.g., `penalty`) are passed to the corresponding function (in this case, `changepoint::cpt.meanvar()`). Here, we pass the `penalty` argument to force the algorithm to use the BIC penalty function.

```
mlb_pelt_meanvar_bic <- mlb_hrs |>
  segment(method = "pelt", model_fn = fit_meanvar, penalty = "BIC")
class(mlb_pelt_meanvar_bic)
```

```
[1] "tidycpt"
```

Source: Article Notebook

3.1 Object structure

Every `tidycpt` object contains three sub-objects:

- `segmenter`: An object that stores information about an optimal set of changepoints found by an algorithm. A `segmenter` object could be a `cpt` object returned by the `cpt.meanvar()` function from changepoint, a `ga` object returned by the `ga()` function from GA, or in principle, any object that provides methods for a few simple generic functions outlined in Section 4.2.
- `model`: A model object inheriting from `mod_cpt`, an internal class for representing model objects. Model objects are created by model-fitting functions, all of whose names start with `fit_` (see Section 5). The `model` of a `tidycpt` object is the model object returned by the `fit_*()` function that corresponds to the one used by the segmenter. Because the models described in Section 5 can be fit to any time series based solely on a specified set of changepoints, the information in this object does *not* depend on the algorithm used to segment the times series: it only depends on the set of changepoints returned by the algorithm.
- `elapsed_time`: the clock time that elapsed while the algorithm was running.

3.2 Methods available

`tidycpt` objects implement methods for the generic functions `as.ts()`, `changepoints()`, `diagnose()`, `fitness()`, `model_name()`, and `plot()`, as well as the three generic functions from the broom package `augment()`, `tidy()`, and `glance()`.

```
methods(class = "tidycpt")
```

```
 [1] as.model     as.segmenter as.ts        augment      changepoints
 [6] diagnose     fitness      glance       model_name   plot
[11] print        tidy
see '?methods' for accessing help and source code
```

Source: Article Notebook

For the most part, methods for `tidycpt` objects typically defer to the methods defined for either the `segmenter` or the `model`, depending on the user's likely intention. To that end, both *segmenters* and *models* implement methods for the generic functions `as.ts()`, `changepoints()`, `model_name()`, and `nobs()`.

```
[1] "as.ts"        "changepoints" "model_name"   "nobs"
```

Source: Article Notebook

The `changepoints()` function return the *indices* of the changepoint set, unless the `use_labels` argument is set to `TRUE`.

```
changepoints(mlb_pelt_meanvar_bic)
```

```
[1]  6 22 28 31 34 54 64
```

```
changepoints(mlb_pelt_meanvar_bic, use_labels = TRUE) |>
  as_year()
```

```
[1] "1930" "1946" "1952" "1955" "1958" "1978" "1988"
```

Source: Article Notebook

For example, the `plot()` method for `tidycpt` simply calls the `plot()` method for the `model` of the `tidycpt` object. The plot shown in Figure 3 uses ggplot2 to illustrate how the proposed changepoint set segments the time series.

```
plot(mlb_pelt_meanvar_bic)
```

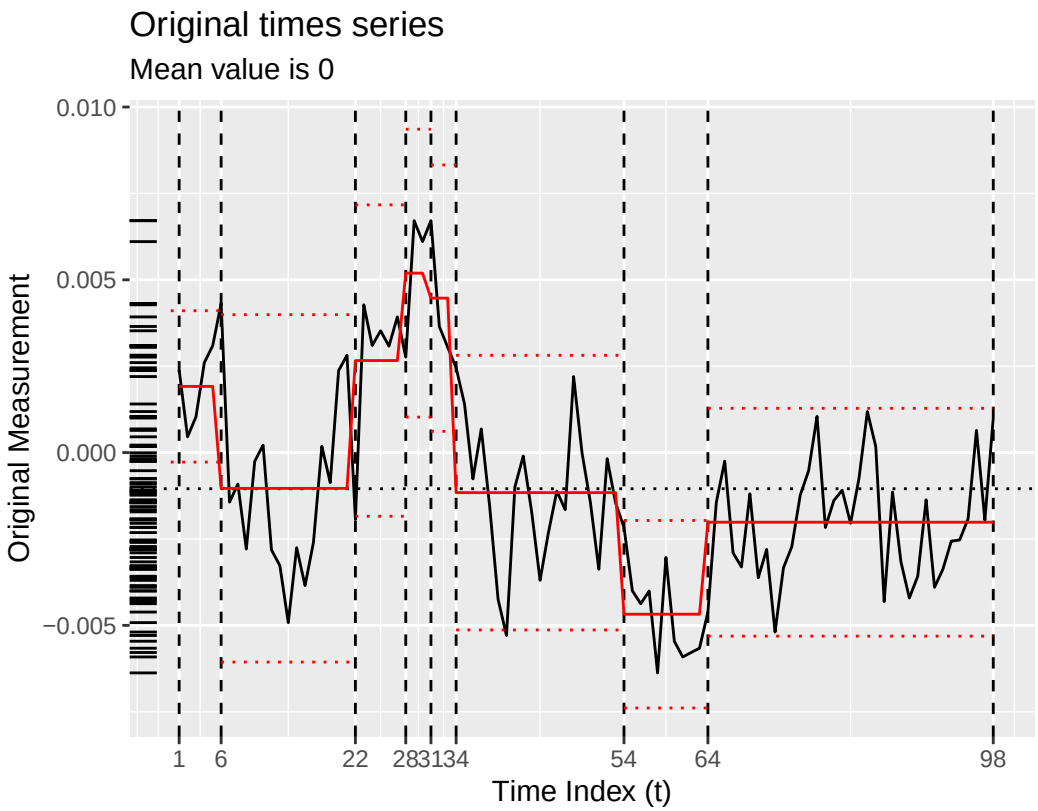


Figure 3: The MLB time series, as segmented by the PELT algorithm using the normal mean-var model and the BIC penalty.

Source: Article Notebook

Changepoint detection models in tidychangepoint follow the design interface of the broom package. Therefore, `augment()`, `tidy()`, and `glance()` methods exist for `mod_cpt` objects.

- `augment()` returns a `tsibble` (Wang, Cook, and Hyndman 2020) that is grouped according to the regions defined by the changepoint set.

```
augment(mlb_pelt_meanvar_bic)
```

```
# A tsibble: 98 x 5 [1]
# Groups:    region [8]
   index         y region  .fitted    .resid
   <int>     <dbl> <fct>     <dbl>     <dbl>
 1     1  0.00239  [0,6)   0.00192  0.000471
 2     2  0.000456 [0,6)   0.00192 -0.00146
```

```
 3     3  0.00103  [0,6)   0.00192 -0.000881
 4     4  0.00260  [0,6)   0.00192  0.000689
 5     5  0.00310  [0,6)   0.00192  0.00118
 6     6  0.00432  [6,22) -0.00104  0.00535
 7     7 -0.00143  [6,22) -0.00104 -0.000398
 8     8 -0.000911 [6,22) -0.00104  0.000124
 9     9 -0.00279  [6,22) -0.00104 -0.00176
10    10 -0.000256 [6,22) -0.00104  0.000780
# i 88 more rows
```

Source: Article Notebook

- `tidy()` returns a `tibble` (Müller and Wickham 2023) that provides summary statistics for each region. These include any parameters that were fit, which are prefixed in the output by `param_`.

```
tidy(mlb_pelt_meanvar_bic)
```

```
# A tibble: 8 x 10
  region  num_obs       min      max     mean      sd begin   end param_mu
  <chr>     <int>     <dbl>    <dbl>    <dbl>   <dbl> <dbl> <dbl>    <dbl>
1 [0,6)         5  0.000456  0.00310  0.00192 0.00112     0     6  0.00192
2 [6,22)       16 -0.00492   0.00432 -0.00104 0.00256     6    22 -0.00104
3 [22,28)       6 -0.00193   0.00428  0.00266 0.00230    22    28  0.00266
4 [28,31)       3  0.00276   0.00671  0.00519 0.00213    28    31  0.00519
5 [31,34)       3  0.00305   0.00671  0.00447 0.00197    31    34  0.00447
6 [34,54)      20 -0.00529   0.00245 -0.00116 0.00203    34    54 -0.00116
7 [54,64)      10 -0.00638  -0.00216 -0.00468 0.00138    54    64 -0.00468
8 [64,98]      35 -0.00519   0.00119 -0.00201 0.00168    64    98 -0.00201
# i 1 more variable: param_sigma_hatsq <dbl>
```

Source: Article Notebook

- `glance()` returns a one-row `tibble` that provides summary statistics for the algorithm. This includes the fitness, which is the value of the penalized objective function (see Section 6) that was used.

```
glance(mlb_pelt_meanvar_bic)
```

```
# A tibble: 0 x 1
# i 1 variable: elapsed_time <drtn>
```

Source: Article Notebook

4 Segmenters

In the example above, the `segmenter` is of class `cpt`, because `segment()` simply wraps the `cpt.meanvar()` function from the changepoint package.

```
mlb_pelt_meanvar_bic |>
  as.segmenter() |>
  class()
```

```
[1] "cpt"
attr(,"package")
[1] "changepoint"
```

Source: Article Notebook

In addition to the generic functions listed above, *segmenters* implement methods for the generic functions `as.segmenter()`, `fitness()`, `model_args()`, and `seg_params()`.

```
[1] "as.seg_cpt" "fitness"    "model_args" "seg_params"
```

Source: Article Notebook

Note that while tidychangepoint uses only S3 classes, segmenters (such as those of class `cpt`) may belong to S4 classes. [This is one reason why `as.segmenter()`, which coverts a segmenter of any class to a `mod_cpt` object, is necessary.] `fitness()` returns a named vector with the type and value of the penalized objective function used by the segmenting algorithm.

```
fitness(mlb_pelt_meanvar_bic)
```

```
      BIC
-998.7452
```

Source: Article Notebook

4.1 Segmenting algorithms provided

Currently, `segment()` wraps the following algorithms:

- PELT and Binary Segmentation as implemented by changepoint (Killick and Haynes 2022)
- Wild Binary Segmentation as implemented by wbs (Baranowski and Fryzlewicz 2019)
- A variety of genetic algorithms as implemented by GA (Scrucca 2024). Specific variants include Shi's algorithm (Shi et al. 2022) and Coen's algorithm (Suárez-Sierra, Coen, and Taimal 2023).
- Trivial algorithms for no changepoints, manually input changepoints, and randomly selected changepoints

In particular, setting the `method` argument to the `segment()` function to `ga` allows the user to specify any of the model-fitting functions described in Section 5.1 (or a user-defined model-fitting function as in Section 5.2) as well as any of the penalized objective functions in Section 6. While these algorithms can be slow, they are quite flexible.

4.2 Extending tidychangepoint to include new algorithms

In order to a segmenting algorithm to work with tidychangepoint the class of the resulting object must implement, at a minimum, methods for the following generic functions:

- From stats: `as.ts()`, and `nobs()`. These are needed perform basic operations on the time series.
- From tidychangepoint: `changepoints()` must return a single integer vector with the best changepoint set as defined by the algorithm. `fitness()` must return a named double vector with the appropriately named value of the penalized objective function. `as.segmenter()` dumps the contents of the object into a `mod_cpt` object. `model_name()`, `model_args()`, and `seg_params()` provide information about the kind of model that was fit, and any arguments passed to the segmenting function.

These compatibility layers are generally not difficult or time-consuming to write.

4.3 Spotlight: Coen's algorithm

The goal of Coen's algorithm (Suárez-Sierra, Coen, and Taimal 2023) is to find the optimal set of changepoints for the NHPP model described in Section 5.3. That is, given a time series y, find the changepoint set τ that minimizes the value of $BMDL(\tau, NHPP(y|\hat{\theta}_\tau))$. To find good candidates, this genetic algorithm starts with a randomly selected set of `popSize` changepoint sets, and then iteratively "mates" the best (according to the BMDL penalty function, see Section 6.3) pairs of these changepoint sets to recover a new "generation" of

`popSize` changepoint sets. This process is repeated until a stopping criteria is met (most simply, `maxiter` times). Thus, a constant number (`popSize` × `maxiter`) possible changepoint sets are considered, and the one with the lowest BMDL score is selected.

Coen's algorithm is implemented in tidychangepoint via the `segment()` function with the `method` argument set to `ga-coen`. As the code below reveals, this is just a special case of the `segment_ga()` function that wraps `GA::ga()`. Note that the `model_fn` argument is set to `fit_nhpp` and the `penalty_fn` argument is set to `BMDL`. The running time of the back-end function `GA::ga()` is sensitive to the size of the changepoint sets considered, especially in the first generation, and thus we use the `population` argument to inform the selection of the first generation (see below). Coen's algorithm runs in about the same time as a naïve algorithm that randomly selects the same number of changepoints, but produces changepoint sets with significantly better BMDL scores.

```
segment_ga_coen
```

```
function(x, ...) {
  segment_ga(
    x, model_fn = fit_nhpp, penalty_fn = BMDL,
    population = build_gabin_population(x), popSize = 50, ...
  )
}
<bytecode: 0x557097b08338>
<environment: namespace:tidychangepoint>
```

Source: Article Notebook

By default, the function `GA::gabin_Population()` selects candidate changepoints uniformly at random with probability 0.5, leading to candidate changepoint sets of size $n/2$, on average. These candidate changepoint sets are usually poor (since $n/2$ changepoints is ludicrous), and we observe much better and faster performance by seeding the first generation with smaller candidate changepoint sets. To this end, the `build_gabin_population()` function runs several fast algorithms (i.e., PELT, Wild Binary Segmentation, etc.) and sets the initial probability of being selected to three times the average size of the changepoint set found by these algorithms. This results in much more rapid covergence around the optimal changepoint set. Alternatively, tidychangepoint also provides the `log_gabin_population()` function, which sets the initial probability to $\ln n/n$.

5 Models

All `model` objects are created by calls to one of the model-fitting functions listed in Section 5.1, whose name begins with `fit_`. These functions all inherit from the class `fun_cpt`. All `model`

objects inherit from the `mod_cpt` base class.

The `model` object in our example is created by `fit_meanvar()`, and is of class `mod_cpt`.

```
mlb_pelt_meanvar_bic |>
  as.model() |>
  str()
```

```
List of 6
 $ data         : Time-Series [1:98] from 1 to 98: 0.002387 0.000456 0.001034 0.002604 0.003
 $ tau          : int [1:7] 6 22 28 31 34 54 64
 $ region_params: tibble [8 x 3] (S3: tbl_df/tbl/data.frame)
  ..$ region           : chr [1:8] "[0,6)" "[6,22)" "[22,28)" "[28,31)" ...
  ..$ param_mu         : num [1:8] 0.00192 -0.00104 0.00266 0.00519 0.00447 ...
  ..$ param_sigma_hatsq: Named num [1:8] 9.99e-07 6.17e-06 4.40e-06 3.01e-06 2.58e-06 ...
  .. ..- attr(*, "names")= chr [1:8] "[0,6)" "[6,22)" "[22,28)" "[28,31)" ...
 $ model_params : NULL
 $ fitted_values: num [1:98] 0.00192 0.00192 0.00192 0.00192 0.00192 ...
 $ model_name   : chr "meanvar"
 - attr(*, "class")= chr "mod_cpt"
```

Source: Article Notebook

In addition to the generic functions listed above, models implement methods for the generic functions `coef()`, `fitted()`, `logLik()`, `plot()`, and `residuals()` , as well as `augment()`, `tidy()`, and `glance()`.

```
 [1] "augment"   "coef"      "diagnose"  "fitted"    "glance"    "logLik"
 [7] "plot"      "print"     "residuals" "tidy"
```

Source: Article Notebook

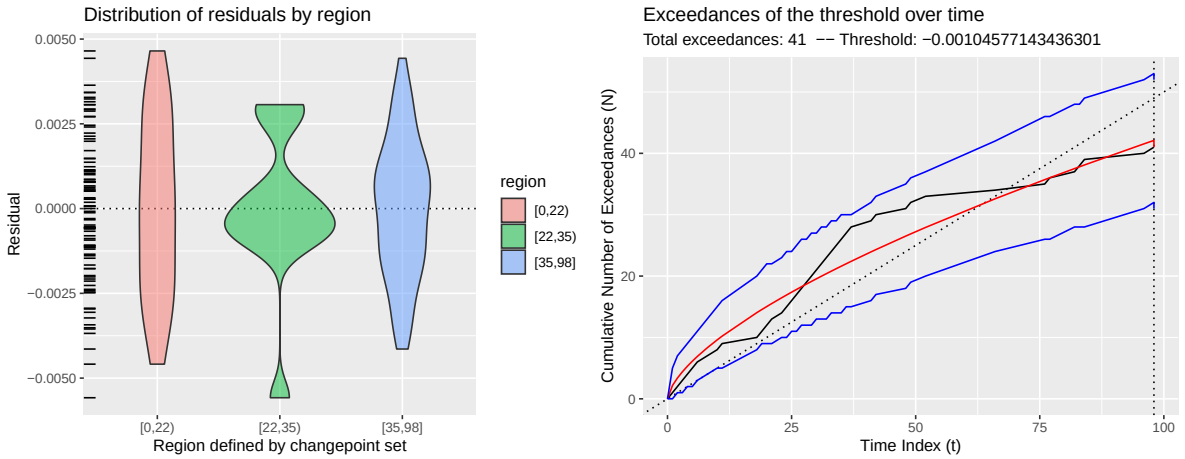
Like other model objects that implement `glance()` methods, the output from the `glance()` function summarizes the fit of the model to the data.

```
mlb_pelt_meanvar_bic |>
  as.model() |>
  glance()
```

```
# A tibble: 1 x 11
  pkg   version algorithm params num_cpts    rmse logLik   AIC   BIC  MBIC   MDL
  <chr> <pckg_> <chr>     <list>    <int>   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
1 tidy~ 0.0.0.~ meanvar   <NULL>        7 0.00186   477. -908. -849. -877. -863.
```

Source: Article Notebook

The `diagnose()` function provides an informative visual understanding of the quality of the fit of the model, which may vary depending on the type of model. In Figure 4, note how Figure 4a summarizes the distribution of the residuals, whereas Figure 4b shows the growth of the cumulative number of exceedances (see Section 5.3) relative to the expected growth based on the NHPP model, with the blues lines showing a 95% confidence interval.



(a) The `diagnose()` function applied to a model object of class `mod_cpt`.

(b) The `diagnose()` function applied to a model object of class `nhpp`.

Figure 4: Output from the `diagnose()` function.

5.1 Model-fitting functions provided

The following model-fitting functions are provided by tidychangepoint:

- `fit_meanshift_norm()`: fits the meanshift model assuming a normal distribution and white noise errors with the $m + 2$ parameters $\theta = \{\sigma^2, \mu_0, \dots, \mu_m\}$.
- `fit_meanshift_lnorm()`: fits the meanshift model assuming a log-normal distribution and white noise errors with the $m + 2$ parameters $\theta = \{\sigma^2, \mu_0, \dots, \mu_m\}$.
- `fit_meanshift_pois()`: fits the meanshift model assuming a Poisson distribution and white noise errors with the $m + 2$ parameters $\theta = \{\sigma^2, \mu_0, \dots, \mu_m\}$.
- `fit_meanshift_norm_ar1()`: fits the meanshift model assuming a normal distribution and autocorrelated AR(1) lagged errors (as decribed in Shi et al. (2022)) with the $m+3$ parameters $\theta = \{\sigma^2, \phi, \mu_0, \dots, \mu_m\}$, where ϕ is the measure of autocorrelation.
- `fit_trendshift()`: fits the trendshift model assuming a normal distribution and white noise errors with the $2m + 3$ parameters $\theta = \{\sigma^2, \mu_0, \dots, \mu_m, \beta_0, \dots, \beta_m\}$. This model is a special case of `fit_lmshift()`.

- `fit_trendshift_ar1()`: fits the trendshift model assuming a normal distribution and autocorrelated errors with the $2m + 4$ parameters $\theta = \{\sigma^2, \phi, \mu_0, \dots, \mu_m, \beta_0, \dots, \beta_m\}$.
- `fit_lmshift()`: fits a polynomial of degree p model assuming a normal distribution and white noise errors with the $p(m + 1) + 1$ parameters $\theta = \{\sigma^2, \beta_{00}, \dots, \beta_{0m}, \beta_{10}, \dots, \beta_{pm}\}$. This model uses the `lm()` function from the stats, which provides flexibility.
- `fit_lmshift_ar1()`: fits a polynomial of degree p model assuming a normal distribution and autocorrelated errors with the $p(m + 1) + 1$ parameters $\theta = \{\sigma^2, \beta_{00}, \dots, \beta_{0m}, \beta_{10}, \dots, \beta_{pm}\}$.
- `fit_meanvar()`: fits the meanvar model with the $2m + 2$ parameters $\theta = \{\sigma_0^2, \dots, \sigma_m^2, \mu_0, \dots, \mu_m\}$.
- `fit_nhpp()`: fits the non-homogeneous Poisson process model described in Section 5.3 with the $2m + 2$ parameters $\theta = \{\alpha_0, \dots, \alpha_m, \beta_0, \dots, \beta_m\}$

5.2 Extending tidychangepoint to include new models

Users can create their own model-fitting functions. The names of these functions should start with `fit_` for consistency, and they must be registered with a call to `fun_cpt()`. The first argument `x` must be the time series and the second argument `tau` must be a subset of the indices of `x`. Any subsequent arguments can be passed through the dots. Every `fit_` function returns an object of class `mod_cpt`, which can be created with the arguments shown below.

```
args(new_mod_cpt)
```

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

Source: Article Notebook

5.3 Spotlight: Non-homogenous Poisson process model

Having already described the meanshift model in Section 1, we now describe mathematically the NHPP model used in Coen's algorithm (see Section 4.3) and created by the `fit_nhpp()` function listed in Section 5.1.

Let $z(y) \subseteq \{t\}$ be a subset of the *indices* of the original times series $\{y_t\}$, for which the observation y_t exceeds some threshold, which by default is the empirical mean $\bar{y}$. More formally, a time index $t \in z$ if and only if $y_t > \bar{y}$. This new time series $z(y)$ is called the *exceedances* of the original time series y. Following Suárez-Sierra, Coen, and Taimal (2023), we model the time series $z(y)$ as a non-homogenous Poisson process (NHPP), using a Weibull distribution

parameterized by shape (α) and scale (β). In a Bayesian setting, those parameters follow a Gamma distribution with their appropriate hyperparameters. Thus, we assume that there exists a set of parameters $\theta = (\alpha_0 \dots \alpha_m, \beta_0, \dots, \beta_m)$ for which the exceedances $z(y)$ of the original time series y are modeled accurately as a non-homogenous Poisson process.

The best-fit parameters are determined by maximizing the log posterior function described below

$$\ln\left(g(\theta_\tau)|y\right) \propto \ln L_{NHPP}(y|\theta_\tau) + \ln g(\theta_\tau)\,.$$

where L is the likelihood function and g is the prior probability distribution function. Closed-form expressions for each of these components for a Weibull distribution with two Gamma priors are given in Suárez-Sierra, Coen, and Taimal (2023). The `tidy()` function displays the values of the fitted parameter values, or one can pull the `region_params` object directly from the `model` object.

```
mlb_coen |>
  as.model() |>
  pluck("region_params")
```

```
# A tibble: 1 x 5
  region param_alpha param_beta logPost logLik
  <chr>        <dbl>      <dbl>   <dbl>  <dbl>
1 [0,98]       0.648      0.305   -76.5  -74.3
```

Source: Article Notebook

6 Penalized objective functions

We call the function f a *penalized objective function* because it comes in the form

$$f(\tau, M(y|\theta_\tau)) = P_f(\tau) - 2\ln L_M(y|\theta_\tau)\,,$$

where P_f is a function that guards against overfitting by penalizing large changepoint sets—that is, it is a monotonically increasing function of m. [Occasionally, we may abuse terminology by referring to f as a *penalty function*.] The stats package (R Core Team 2024) provides generic methods for the well-known penalty functions `AIC()` and `BIC()`, which work seamlessly on tidychangepoint models because they all implement methods for `logLik()`. In addition, tidychangepoint provides generic functions and methods for three additional penalty functions: `MBIC()`, `MDL()`, and `BMDL()`.

All penalty functions return 0 for the special case in which $m = 0$. For ease of explanation, let the vector $\ell_j = \tau_{j+1} - \tau_j$ for $1 \le j \le m+1$ encode the lengths of the $m+1$ regions defined by the changepoint set τ.

6.1 Modified Bayesian Information Criterion

Following Zhang and Siegmund (2007) and Li and Lund (2012), we define the MBIC as:

$$P_{MBIC}(\tau) = 3m \ln n + \sum_{j=1}^{m+1} \ln \frac{\ell_j}{n} \,.$$

6.2 Minimum Descriptive Length

As described in Shi et al. (2022) and Li and Lund (2012), we define the MDL as:

$$P_{MDL}(\tau) = \frac{a(\theta_\tau)}{2} \cdot \sum_{j=0}^{m} \log \ell_j + 2 \ln m + \sum_{j=2}^{m} \ln \tau_j + (2 + b(\theta_\tau)) \ln n \,.$$

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. For example, the in the meanshift model, $a(\theta) = 1$ to account for the μ_i's, and $b(\theta) = 1$ to account for σ^2.

The MBIC and MDL differ from the penalties imposed by either the Akaike Information Criterion (AIC) or the BIC in important ways. For example, the penalty for the AIC depends just linearly on the number of changepoints, while the BIC depends on the number of changepoints as well as the length of the original times series. Conversely, the MDL penalty depends not only on the number of changepoints, but the *values* of the changepoints. Because of this, changepoints far from zero can have disproportionate impact.

6.3 Bayesian Minimum Descriptive Length

The Bayesian Minimum Descriptive Length combines the MDL penalty function with the log prior g for the best-fit parameters $\hat{\theta}$ in the NHPP model described in Section 5.3. Currently, the BMDL penalty can only be applied to the NHPP model.

The exact value of the BMDL is then:

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

6.4 Extending tidychangepoint to include new penalty functions

New penalty functions can be contributed to tidychangepoint by defining a new generic function and implementing a method for the `logLik` class. Penalty functions should return a `double` vector of length one.

Note that similar to the `logLik()` method for `lm`, the `logLik()` method for `mod_cpt` embeds relevant information into the object's attributes. These include the quantities necessary to compute `AIC()`, `BIC()`, `MDL()`, etc.

```
mlb_pelt_meanvar_bic |>
  as.model() |>
  logLik() |>
  unclass() |>
  str()
```

```
 num 477
 - attr(*, "num_params_per_region")= int 2
 - attr(*, "num_model_params")= int 0
 - attr(*, "df")= num 23
 - attr(*, "nobs")= int 98
 - attr(*, "tau")= int [1:7] 6 22 28 31 34 54 64
```

Source: Article Notebook

7 Results

In this section we illustrate how tidychangepoint can be useful in determining and analyzing possible sets of changepoints on both simulated and real data.

7.1 Recap of MLB example

In Section 2 and Section 3, we computed created five different `tidycpt` objects, each of which contains information about a different combination of algorithm, model, and penalty function.

```
mods <- list(
  mlb_pelt, mlb_pelt_meanshift, mlb_pelt_meanvar_bic, mlb_ga_shi, mlb_coen
)
```

Source: Article Notebook

Using `map()` (from the purrr package (Wickham and Henry 2023)) and `glance()`, we can easily compare our results, although care must be taken to ensure that those comparisons are meaningful. In this case, we build a `list` object of the various `tidycpt` objects, then iterate the `glance()` function over that list with the `map()` function. Since `glance()` always returns a one-row `tibble`, we can then combine those `tibble`s into one using the `list_rbind()` function. While comparing, say, the elapsed time across these algorithms and models is fair, comparing the fitness values is more fraught. BMDL scores are not comparable to BIC or MBIC scores, for example.

```
mods |>
  map(glance) |>
  list_rbind()
```

```
# A tibble: 0 x 1
# i 1 variable: elapsed_time <drtn>
```

Source: Article Notebook

Consider the distribution of the changepoints identified by our five algorithms.

```
mods |>
  map(changepoints) |>
  list_c() |>
  table()
```

```
 6  7 22 23 28 31 34 35 36 54 64
 1  1  2  1  1  1  1  1  1  1  1
```

Source: Article Notebook

We see that a changepoint at 22 or 23 occurs three times, while a changepoint at 34, 35, or 36 also occurs three times. Discarding the correct but spurious changepoint at 98 (the end of the time series), we also have two votes for a changepoint at 54 or 55 and two votes for a changepoint at 64 or 65. Let us create a vector of candidate changepoints with these values, which we might consider as a sort of algorithmic consensus. These correspond to the years show below, which are in the ballpark with the changepoints suggested by Woltring, Rost, and Jubenville (2018) and B. James (2014), but far from the true changepoints of 1973 and 2022.

```
cpt_algs <- c(22, 35, 55, 65)
mlb_pelt$time_index[cpt_algs] |>
  as_year()
```

```
[1] "1946" "1959" "1979" "1989"
```

```
cpt_true <- c(49)
mlb_pelt$time_index[cpt_true] |>
  as_year()
```

```
[1] "1973"
```

Source: Article Notebook

The PELT algorithm, using the meanvar model and the MBIC criteria, did return a subset of `cpt_algs`. But why weren't the candidate changepoint sets `cpt_algs` or `cpt_true` identified by our algorithms? We can use tidychangepoint to dig deeper.

Our target is the value of the penalized objective function `MBIC()` returned by PELT after searching with the meanvar model. The `fitness()` function gives us this value.

```
fitness(mlb_pelt)
```

```
     MBIC
-920.4666
```

Source: Article Notebook

Because PELT is known to be optimal under mild conditions, all other candidate changepoints sets should return a value that is worse. [Note that we are in the unusual but valid situation in which the log likelihood is positive!] As expected, both of these MBIC values are worse (further from $-\infty$).

```
list(cpt_algs, cpt_true) |>
  map(fit_meanvar, x = mlb_hrs) |>
  map_dbl(MBIC)
```

```
[1] -901.4766 -881.1987
```

Source: Article Notebook

Similarly, we can compare these candidate changepoint sets to the results from Coen's algorithm, which uses the NHPP model and the BMDL penalty. This time, the true changepoint set is closer to the (spurious) optimal changepoint set than our consensus set. Note that Coen's algorithm is only a heuristic, and so unlike in the case of PELT, it is possible that we could manually identify a changepoint set with a lower BMDL than the one found by the algorithm. However, in this case, we don't.

```
fitness(mlb_coen)
```

```
   BMDL
153.082
```

```
list(cpt_algs, cpt_true) |>
  map(fit_nhpp, x = mlb_hrs) |>
  map_dbl(BMDL)
```

```
[1] 225.7081 182.2870
```

Source: Article Notebook

7.2 Particular matter in Bogotá, Colombia

Consider the time series on particulate matter in Bogotá, Colombia collected daily from 2018–2020, discussed in Suárez-Sierra, Coen, and Taimal (2023), and shown in Figure 5. These data are included in the tidychangepoint as `bogota_pm`.

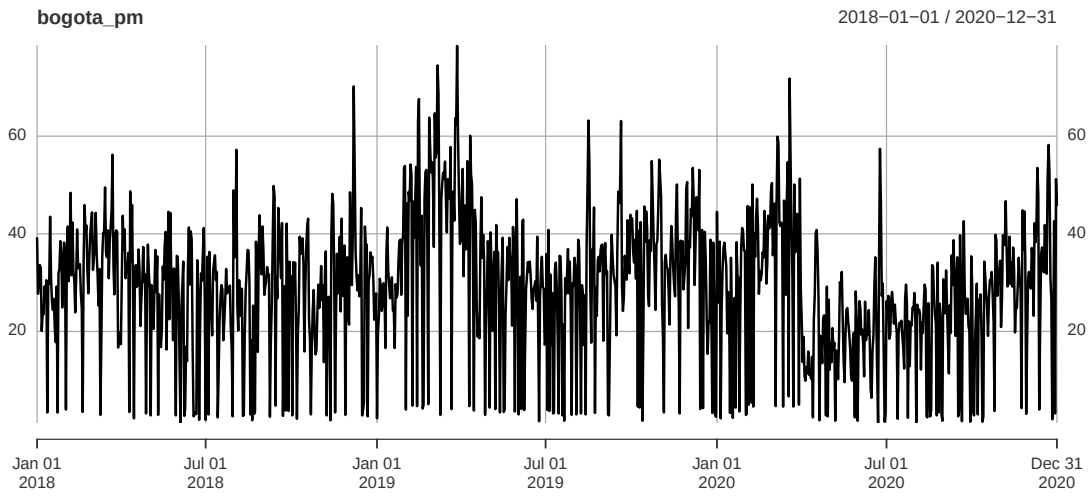


Figure 5: Particular matter above 2.5 microns in Bogotá, Colombia, recorded daily from 2018–2020.

Source: Article Notebook

We can use the PELT algorithm as implemented in the changepoint package through the tidychangepoint interface by calling the `segment()` function and specifying `pelt` as the `method` argument.

```
bog_pelt <- segment(bogota_pm, method = "pelt")
```

Source: Article Notebook

Unfortunately, with the default options PELT performs poorly on these data, identifying an excessive number (7) of changepoints.

```
length(changepoints(bog_pelt))
```

```
[1] 7
```

Source: Article Notebook

However, using the meanvar model instead of the meanshift model improves the situation considerably. Note that we have set the `minseglen` argument to 3 to avoid unreasonably short regions.

```
bog_pelt_meanvar <- bogota_pm |>
  segment(method = "pelt", model_fn = fit_meanvar, minseglen = 3)
length(changepoints(bog_pelt_meanvar))
```

```
[1] 4
```

Source: Article Notebook

The `diagnose()` function produces the informative data graphic shown in Figure 6. The figure shows the original time series (black line) segmented by the changepoints determined by the PELT algorithm (vertical dotted lines). The horizontal red lines indicate the mean and 1.96 times the standard deviation within each region, as per the summary produced by `tidy()`. The lower plot shows the distribution of the residuals.

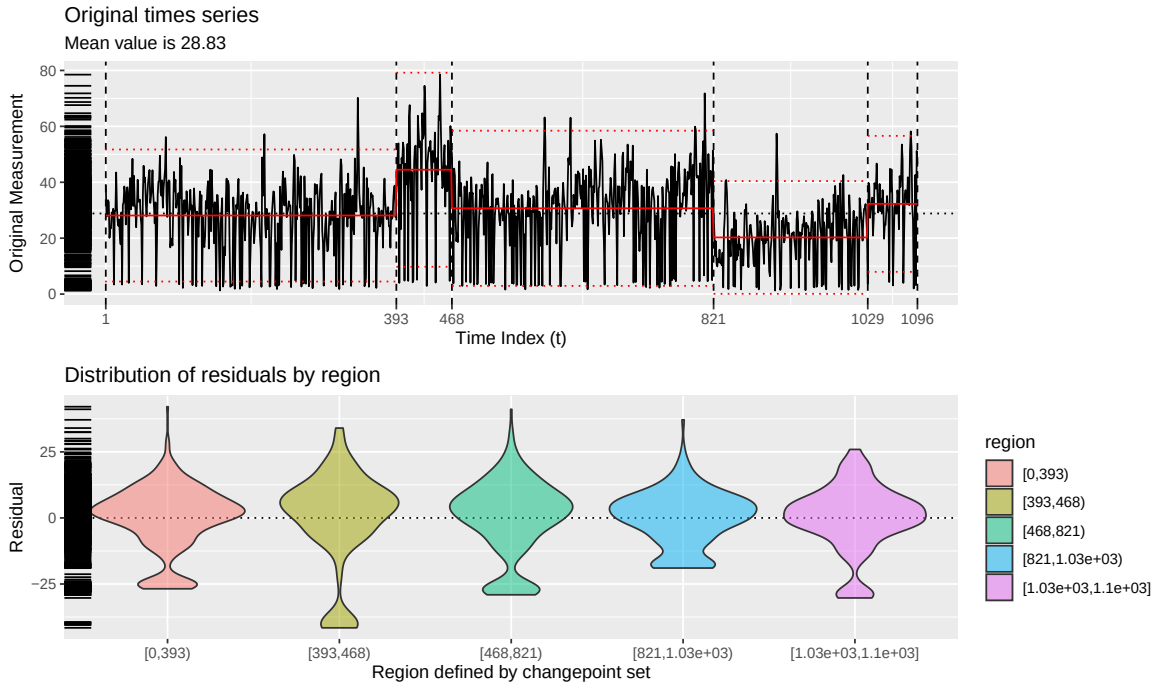


Figure 6: Diagnostic plot for the PELT algorithm applied to the Bogotá particulate matter data.

Source: Article Notebook

A careful look at the data suggest that the trendshift model, which allows for a linear slope within each region, might be more appropriate. Setting the `method` argument to `segment()` to

`ga` allows us to employ a genetic algorithm for which we can specify this model via the `model_fn` argument. Here, we also use the `MDL()` penalty function and the `log_gabin_population()` function.

```
bog_ga <- bogota_pm |>
  segment(
    method = "ga", model_fn = fit_trendshift, penalty_fn = MDL,
    population = log_gabin_population(bogota_pm), maxiter = 500, run = 100
  )
```

Source: Article Notebook

Source: Article Notebook

In this case, the algorithm finds 4 changepoints.

```
changepoints(bog_ga, use_labels = TRUE)
```

```
[1] "2018-11-06" "2019-04-19" "2019-12-20" "2020-03-29"
```

Source: Article Notebook

Note that the MDL score reported by the genetic algorithm is better (lower) than that of the corresponding changepoint set returned by PELT. This may not be surprising, given that PELT minimized the MBIC penalty value fit to a different model, but nevertheless it suggests that genetic algorithms have something to contribute.

```
fitness(bog_ga)
```

```
     MDL
8803.814
```

```
bog_pelt |>
  changepoints() |>
  fit_trendshift(x = bogota_pm) |>
  MDL()
```

```
[1] 8843.263
```

Source: Article Notebook

Note also that, without running the genetic algorithm again, we can determine that adding AR(1) lagged errors to our model results in an even lower MDL score.

```
bogota_pm |>
  fit_trendshift_ar1(tau = changepoints(bog_ga)) |>
  MDL()
```

```
[1] 8778.436
```

Source: Article Notebook

Figure 7 shows that the trendshift model fits very well.

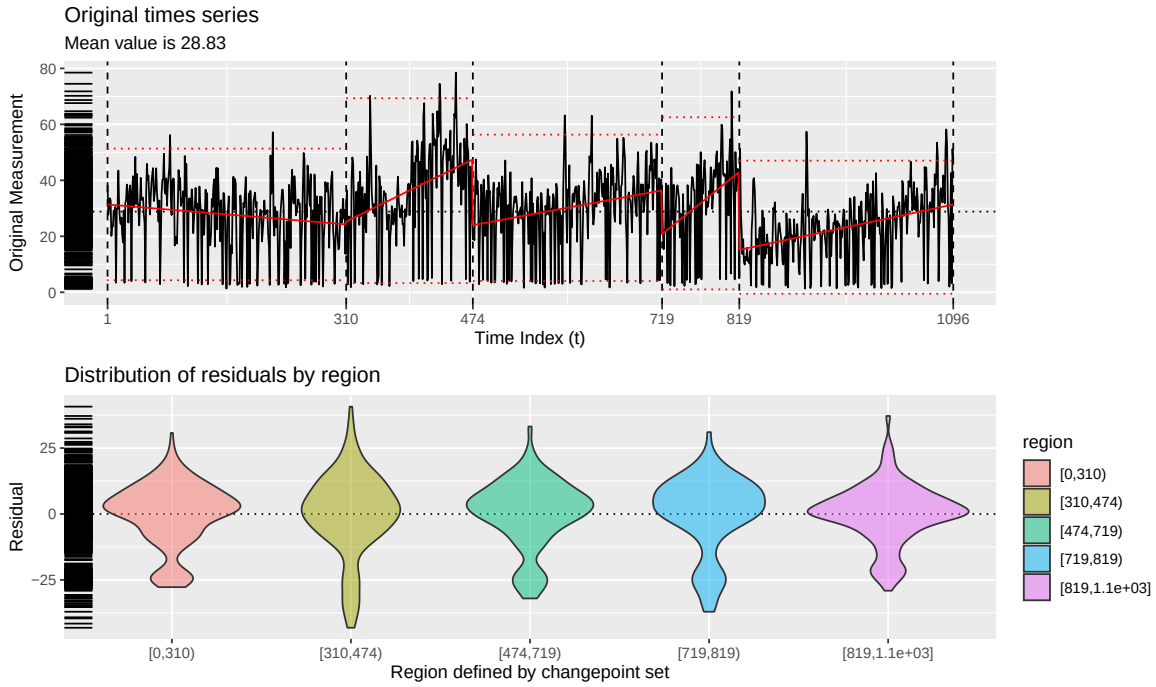


Figure 7: Diagnostic plot for a genetic algorithm with a trendshift model applied to the Bogotá particulate matter data.

Source: Article Notebook

8 Conclusion

8.1 Future work

There is more work to be done to bring additional changepoint detection algorithms, models, and penalty functions into tidychangepoint. Please see the package's GitHub Issues page for a complete list.

One obvious area for improvement in tidychangepoint is computational cost. While the implementations of changepoint detection algorithms in changepoint use pre-compiled C code, all of the code for tidychangepoint is presently written in R. Particularly with respect to genetic algorithms and Bayesian computations, it is likely that pushing some code from R to C could improve performance considerably. Additionally, some algorithms have a known running time in relation to the length of the time series, while others do not.

A second major issue with changepoint detection algorithms is the robustness of the results to a variety of changes in conditions, assumptions, or computational environments. While some algorithms report the probability of an observation being a changepoint, others do not. Some models rely on assumptions about the distribution or frequency of changepoints relative to the length of the time series. In randomized algorithms (such as genetic algorithms), the robustness of the results may be interrelated with stopping criteria, the initial seed or population, or other computational considerations. CUSUM methods operate on a summary of the original time series: are they less susceptible to outliers?

Finally, in this paper we have considered the multiple changepoint detection problem for a univariate time series, but multivariate time series are also being studied in a variety of contexts.

8.2 Summary

We have described how tidychangepoint offers a flexible but standardized architecture for changepoint detection analysis in R in a manner that is consistent with the design principles of the tidyverse. Previously, changepoint detection analysis could be conducted using a variety of R packages, each of which employed its own non-conforming interface. Our unified interface allows users to compare results across different changepoint algorithms, models, and penalty functions with ease.

References

Aminikhanghahi, Samaneh, and Diane J Cook. 2017. "A Survey of Methods for Time Series Change Point Detection." *Knowledge and Information Systems* 51 (2): 339–67. https://doi.org/10.1007/s10115-016-0987-z.

Auger, Ivan E, and Charles E Lawrence. 1989. "Algorithms for the Optimal Identification of Segment Neighborhoods." *Bulletin of Mathematical Biology* 51 (1): 39–54. https://doi.org/10.1016/S0092-8240(89)80047-3.

Bai, Jushan, and Pierre Perron. 1998. "Estimating and Testing Linear Models with Multiple Structural Changes." *Econometrica*, 47–78. https://doi.org/10.2307/2998540.

Baranowski, Rafal, and Piotr Fryzlewicz. 2019. *Wbs: Wild Binary Segmentation for Multiple Change-Point Detection*.

Cho, Haeran. 2016. "Change-point detection in panel data via double CUSUM statistic." *Electronic Journal of Statistics* 10 (2): 2000–2038. https://doi.org/10.1214/16-EJS1155.

Cho, Haeran, and Piotr Fryzlewicz. 2015. "Multiple-Change-Point Detection for High Dimensional Time Series via Sparsified Binary Segmentation." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 77 (2): 475–507. https://doi.org/10.1111/rssb.12079.

Guédon, Yann. 2015. "Segmentation Uncertainty in Multiple Change-Point Models." *Statistics and Computing* 25 (2): 303–20. https://doi.org/10.1007/s11222-013-9433-1.

Haynes, Kaylea, and Rebecca Killick. 2022. *Changepoint.np: Methods for Nonparametric Changepoint Detection*.

Hocking, Toby, Guillem Rigaill, Jean-Philippe Vert, and Francis Bach. 2013. "Learning Sparse Penalties for Change-Point Detection Using Max Margin Interval Regression." In *International Conference on Machine Learning*, edited by Sanjoy Dasgupta and David McAllester, 28:172–80. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR. https://proceedings.mlr.press/v28/hocking13.html.

Jackson, Brad, Jeffrey D Scargle, David Barnes, Sundararajan Arabhi, Alina Alt, Peter Gioumousis, Elyus Gwin, Paungkaew Sangtrakulcharoen, Linda Tan, and Tun Tao Tsai. 2005. "An Algorithm for Optimal Partitioning of Data on an Interval." *IEEE Signal Processing Letters* 12 (2): 105–8. https://doi.org/10.1109/LSP.2001.838216.

James, Bill. 2014. *Fools Rush Inn: More Detours on the Way to Conventional Wisdom*. ACTA Publications.

James, Nicholas A., Wenyu Zhang, and David S. Matteson. 2023. *Ecp: Non-Parametric Multiple Change-Point Analysis of Multivariate Data*.

Killick, Rebecca, Paul Fearnhead, and Idris A Eckley. 2012. "Optimal Detection of Changepoints with a Linear Computational Cost." *Journal of the American Statistical Association* 107 (500): 1590–98. https://doi.org/10.1080/01621459.2012.737745.

Killick, Rebecca, and Kaylea Haynes. 2022. *Changepoint: Methods for Changepoint Detection*. https://github.com/rkillick/changepoint/.

Li, Shanghong, and Robert Lund. 2012. "Multiple Changepoint Detection via Genetic Algorithms." *Journal of Climate* 25 (2): 674–86. https://doi.org/10.1175/2011JCLI4055.1.

Müller, Kirill, and Hadley Wickham. 2023. *Tibble: Simple Data Frames*. https://tibble.tidyverse.org/.

R Core Team. 2024. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Schwarz, Gideon. 1978. "Estimating the Dimension of a Model." *The Annals of Statistics*, March, 461–64. https://doi.org/10.1214/aos/1176344136.

Scott, Andrew Jhon, and Martin Knott. 1974. "A Cluster Analysis Method for Grouping Means in the Analysis of Variance." *Biometrics* 30 (3): 507–12. https://doi.org/10.2307/2529204.

Scrucca, Luca. 2017. *Qcc: Quality Control Charts*. https://github.com/luca-scr/qcc.

———. 2024. *GA: Genetic Algorithms*. https://luca-scr.github.io/GA/.

Shi, Xueheng, Claudie Beaulieu, Rebecca Killick, and Robert Lund. 2022. "Changepoint Detection: An Analysis of the Central England Temperature Series." *Journal of Climate* 35 (19): 6329–42. https://doi.org/10.1175/JCLI-D-21-0489.1.

Suárez-Sierra, Biviana Marcela, Arrigo Coen, and Carlos Alberto Taimal. 2023. "Genetic Algorithm with a Bayesian Approach for Multiple Change-Point Detection in Time Series of Counting Exceedances for Specific Thresholds." *Journal of the Korean Statistical Society* 52 (4): 982–1024. https://doi.org/10.1007/s42952-023-00227-2.

Taimal, Carlos A, Biviana Marcela Suárez-Sierra, and Juan Carlos Rivera. 2023. "An Exploration of Genetic Algorithms Operators for the Detection of Multiple Change-Points of Exceedances Using Non-Homogeneous Poisson Processes and Bayesian Methods." In *Colombian Conference on Computing*, 230–58. Springer. https://doi.org/10.1007/978-3-031-47372-2_20.

Wang, Earo, Dianne Cook, and Rob J Hyndman. 2020. "A New Tidy Data Structure to Support Exploration and Modeling of Temporal Data." *Journal of Computational and Graphical Statistics* 29 (3): 466–78. https://doi.org/10.1080/10618600.2019.1695624.

Wickham, Hadley, and Lionel Henry. 2023. *Purrr: Functional Programming Tools*. https://purrr.tidyverse.org/.

Woltring, Mitchell T, Jim K Rost, and Colby B Jubenville. 2018. "Examining Perceptions of Baseball's Eras: A Statistical Comparison." *Sport Journal*. https://thesportjournal.org/article/examining-perceptions-of-baseballs-eras/.

Yu, Youzhi. 2022. *Ggchangepoint: Combines Changepoint Analysis with Ggplot2*.

Zhang, Nancy R, and David O Siegmund. 2007. "A Modified Bayes Information Criterion with Applications to the Analysis of Comparative Genomic Hybridization Data." *Biometrics* 63 (1): 22–32. https://doi.org/10.1111/j.1541-0420.2006.00662.x.