Journal of Statistical Software

May 2024, Volume 109, Issue 7. doi: 10.18637/jss.v109.i07

cpop: Detecting Changes in Piecewise-Linear Signals

Paul Fearnhead
Lancaster Universtity

Daniel Grose
Lancaster University

Abstract

Changepoint detection is an important problem with a wide range of applications. There are many different types of changes that one may wish to detect, and a wide range of algorithms and software for detecting them. However there are relatively few approaches for detecting changes-in-slope in the mean of a signal plus noise model. We describe the R package **cpop**, available on the Comprehensive R Archive Network (CRAN). This package implements CPOP, a dynamic programming algorithm, to find the optimal set of changes that minimizes an L_0 penalized cost, with the cost being a weighted residual sum of squares. The package has extended the CPOP algorithm so it can analyse data that is unevenly spaced, allow for heterogeneous noise variance, and allows for a grid of potential change locations to be different from the locations of the data points. There is also an implementation that uses the CROPS algorithm to detect all segmentations that are optimal as you vary the L_0 penalty for adding a change across a continuous range of values.

Keywords: changepoints, change-in-slope, dynamic programming, piecewise linear models, structural breaks.

1. Introduction

The detection of change in sequences of data is important across many applications, for example changes in volatility in finance (Andreou and Ghysels 2002), changes in genomic data that represent copy number variation (Niu and Zhang 2012), changes in calcium imaging data that correspond to neurons firing (Jewell, Hocking, Fearnhead, and Witten 2020) or changes in climate data (Reeves, Chen, Wang, Lund, and Lu 2007), amongst many others. Depending on the application, interest can be in detecting changes in different features of the data, and there has been a corresponding wide range of methods that have been developed. See Aminikhanghahi and Cook (2017), Truong, Oudre, and Vayatis (2020), Fearnhead and

Rigaill (2020) and Shi, Gallagher, Lund, and Killick (2022) for recent reviews of changepoint methods and their applications.

For some applications we have data on a piece-wise linear mean function, and we wish to detect the times at which the slope of the mean changes. This is the change-in-slope problem: see the top-left plot of Figure 1 for example simulated data. This is a particularly challenging problem for the following reasons. First, a simple approach to detecting changes in slope is to take first differences of the data, as this transforms a change-in-slope into a change-in-mean, and then apply one of the many methods for detecting changes in mean. However this removes much of the information in the data about the location of changes and such an approach can perform poorly. This can be seen by comparing the raw data in the top-left plot of Figure 1 with the first differenced data in the top-right plot of Figure 1. By eye it is easy to see the rough location of the changes in slope in the former, but almost impossible to see any changes in mean in the latter. Running the pruned exact linear time (PELT) change-in-mean algorithm Killick, Fearnhead, and Eckley (2012) on the first differenced data leads to poor estimates of the location of any changes. Second, the most common approach to detecting multiple changepoints is to use binary segmentation (Scott and Knott 1974) or one of its variants (Fryzlewicz 2014; Kovács, Li, Bühlmann, and Munk 2023). These repeatedly apply a test for a single change-in-slope. However Baranowski, Chen, and Fryzlewicz (2019) shows that such binary segmentation methods do not work for the change-in-slope problem as if you fit a single change-in-slope to data simulated with multiple changes, it will often detect the change at a location near the middle of a segment between changes. Third, dynamic programming algorithms that minimize an L_0 penalized cost, such as optimal partitioning (Jackson, Scargle, Barnes, Arabhi, Alt, Gioumousis, Gwin, Sangtrakulcharoen, Tan, and Tsai 2005) or PELT (Killick *et al.* 2012) cannot be applied to the change-in-slope problem due to dependencies in the model across changepoints from the continuity of the mean at each change.

Despite these challenges, there are three methods developed specifically for detecting changes-in-slope: Trend-filtering (Kim, Koh, Boyd, and Gorinevsky 2009; Tibshirani 2014) which minimizes the residual sum of squares of fit to the data plus an L_1 penalty on the changes-in-slope; narrowest over threshold (NOT, Baranowski *et al.* 2019) that repeatedly performs a test for a single change-in-slope on subsets of the data and combines the results using the narrowest-over-threshold procedure; and continuous-piecewise-linear pruned optimal partitioning (CPOP, Fearnhead, Maidstone, and Letchford 2019) which uses a novel variant of dynamic programming to minimize the residual sum of squares plus an L_0 penalty, i.e., a constant penalty for adding each change. The difference between the L_1 penalty of trend-filtering and the L_0 penalty of CPOP is important in practice: as the former allows one to fit a single change in slope with multiple changes of the same sign. This can lead to either over-fitting the number of changes, or, if a large enough penalty is used to detect the changes accurately, over-smoothing the mean function: see the middle row of plots in Figure 1 for an example. The main difference between CPOP and NOT is that the former fits all changes simultaneously. See Fearnhead *et al.* (2019) for an empirical comparison of the three methods.

A related problem to detecting changes-in-slope is that of fitting piecewise polynomial functions. Yu, Chatterjee, and Xu (2022) present a method for detecting such changes by minimizing a measure of fit to the data with an L_0 penalty, which is the same as used by CPOP. However they do not require the fitted polynomial functions to be continuous at the change-points, which means that it is simple to minimize this criteria using standard dynamic

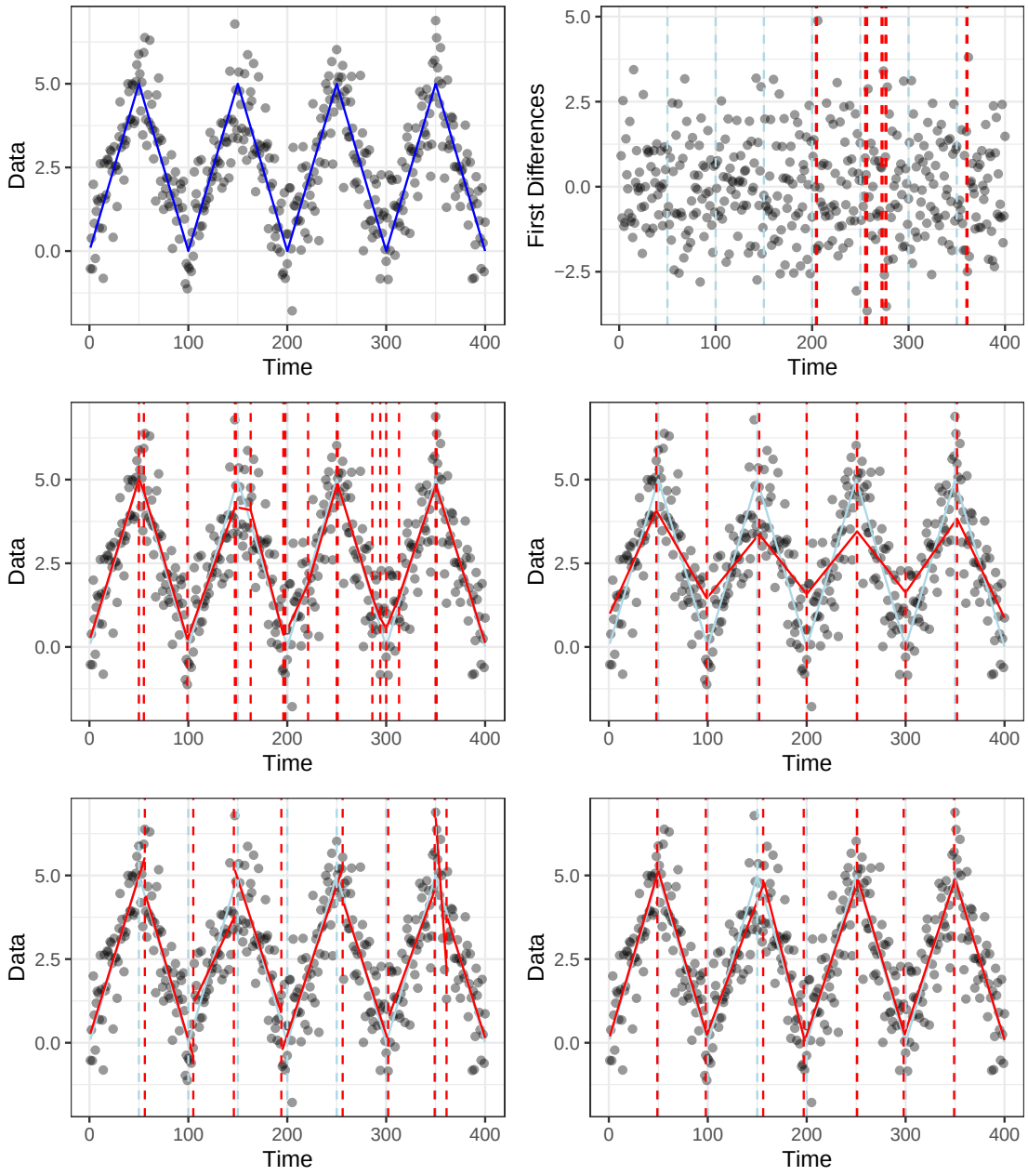

Figure 1: Example data simulated from a change-in-slope model (top left), and results from applying a change-in-mean algorithm to the first differences (top right) or from using trend-filtering (middle row), fitting a piece-wise linear function without imposing continuity (bottom left) and from CPOP (bottom right). In each case the true mean function (solid line) and change locations (vertical dashed lines) are shown in blue or light-blue (to aid visualization), and the estimates in red. For trend filtering we chose the L_1 penalty value based on cross-validation (middle left) or so that it obtained the correct number of changes (middle right). In the former case we over-estimate the number of changes, while in the latter we obtain a poor estimate of the mean. When we do not impose continuity we see we lose some accuracy in estimating the number and location of changes, and in estimating the mean function (compare bottom left and bottom right).

programming recursions (Auger and Lawrence 1989; Jackson *et al.* 2005; Killick *et al.* 2012; Maidstone, Hocking, Rigaill, and Fearnhead 2017). However, ignoring the continuity constraint when it is appropriate can lead to a loss of information and reduce accuracy when estimating the number and location of the changes or the underlying mean function. For example, see the results of this method for fitting a piecewise linear function to the data in the bottom left figure of Figure 1. The trend-filtering methodology mentioned above can be used to detect changes in higher order piecewise polynomial functions with continuity constraints, and constraints on the continuity of derivatives. See also Fearnhead and Liu (2011) for a Bayesian approach to detect changes in such models.

The purpose of this paper is to describe the **cpop** package (Grose and Fearnhead 2024), which is written in R (R Core Team 2024) and available from CRAN at `https://CRAN.R-project.org/package=cpop`, and implements the CPOP algorithm. The latest version of the package was developed in response to an applied challenge, see Section 5, where the data was unevenly spaced and the noise was not homoscedastic, aspects that previous implementations of change-in-slope algorithms could not handle. How the CPOP algorithm is extended to deal with these features is described in Section 2, together with allowing the locations of the changes in slope to not coincide with the observations. This latter aspect can be helpful in reducing the computational cost of the CPOP algorithm for high frequency data by, e.g., searching for segmentations that only allow changes at a smaller grid of possible locations. Section 3 describes the basic functionality of the package, with the extensions to allow for unevenly spaced, heteroscedastic data described, and to specify the grid of potential change locations, in Section 4. This latter section also shows how to impose a minimum segment length and how to implement CPOP within the changepoints for a range of penalties (CROPS) algorithm (Haynes, Eckley, and Fearnhead 2017a) to obtain all segmentations as we vary the value of L_0 penalty. An application of CPOP to analyse decay of spectra from ocean models is shown in Section 5.

1.1. Software for changepoint detection

Currently most software for changepoint detection is available in R (the main exception being the **ruptures** Python package of Truong, Oudre, and Vayatis 2018, which has similar functionality to the **changepoint** described below). There are both many different types of change that one may wish to detect, and many different approaches to detecting multiple changes. Consequently there are a wide range of change algorithms with associated packages in R. For example the **changepoint** package (Killick and Eckley 2014) implements dynamic programming algorithms, such as PELT, for detecting changes in mean, variance or mean and variance. Other dynamic programming algorithms include **fpop** and **gfpop** (Runge, Hocking, Romano, Afghah, Fearnhead, and Rigaill 2023) that implements the functional optimal partitioning algorithm (Maidstone *et al.* 2017) for detecting changes in mean, with the latter package allowing for flexibility as to how the mean changes (such as monotonically increasing) and for different loss functions for measuring fit to data. The **breakfast** package (Anastasiou, Chen, Cho, and Fryzlewicz 2022) implements a range of methods based on recursively applying a test to detect a single change, for example wild binary segmentation (Fryzlewicz 2014) and IsolateDetect (Anastasiou and Fryzlewicz 2022); whilst **mosum** (Meier, Kirch, and Cho 2021) implements the MOSUM procedure. Packages **stepR** (Pein, Hotz, and Sieling 2023) and **FDRseg** implement the multiscale approaches of Frick, Munk, and Sieling (2014), Pein, Sieling, and Munk (2017) and Li, Munk, and Sieling (2016).

Separately there are packages that perform non-parametric change detection, for example **ecp** (James and Matteson 2015) implements the method of Matteson and James (2014), while **changepoint.np** implements the method of Haynes, Fearnhead, and Eckley (2017b). There are also methods for analyzing multiple dimensional data streams, such as **InspectChangepoint** (Wang and Samworth 2018), and **changepoint.geo** (Grundy, Killick, and Mihaylov 2020); Bayesian methods, such as **bcp** (Erdman and Emerson 2008); and methods that implement online procedures such as **CPM** (Ross 2015) and **FoCUS** (Romano, Eckley, Fearnhead, and Rigaill 2023). The **changepoints** package (Xu, Padilla, Wang, and Li 2022) implements a range of changepoint methods for univariate data, including change in mean and (discontinuous) change in polynomial regression, and multivariate data.

However, as mentioned above, there are more limited methods for specifically detecting changes-in-slope. The trend filtering algorithm can be implemented using the `trendfilter` function from the **genlasso** (Arnold and Tibshirani 2022) package, and the NOT algorithm can be implemented using the **not** package (Baranowski, Chen, and Fryzlewicz 2023) or is available within **breakfast**. However current implementations of these do not allow for unevenly spaced, heterogeneous observations or minimum segment lengths, which are all features that can be included within the latest release of the **cpop** package that is described in this article. (Though there is flexibility within the **genlasso** package for implementing general lasso algorithms, and these can be constructed to fit a trend-filtering model to unevenly spaced data).

2. Detecting changes in slope

Assume we have data points $(x_1, y_1), \ldots, (x_n, y_n)$, ordered so that $x_1 < x_2 < \cdots < x_n$. In many applications x_i will be a time-stamp of when response y_i is obtained, whilst in, say, genomic applications, x_i may correspond to a location along the genome at which observation y_i is taken. We wish to model the response, y, as a signal plus noise where the signal is modeled as a continuous piecewise linear function of x. That is

$$y_i = f(x_i) + \epsilon_i, \tag{1}$$

where $f(x)$ is a continuous piecewise linear function, and ϵ_i is noise. If the function $f(x)$ has K changes in slope in the open interval (x_1, x_n), and these occur at x-values $\tau_1, \ldots, \tau_K$, and we define τ_0 and τ_{K+1} to be arbitrary values such that $\tau_0 \leq x_1$ and $\tau_{K+1} \geq x_n$ then we can uniquely define $f(x)$ on $[x_1, x_n]$ by specifying the values $f(\tau_i)$ for $i = 0, \ldots, K+1$. The function $f(x)$ can then be obtained via straight-line interpolation between the points $(\tau_i, f(\tau_i))$.

Our interest is in estimating the number of changes in slope, K, their locations, $\tau_1, \ldots, \tau_K$, and the underlying signal. The latter is equivalent to estimating $f(\tau_i)$ for $i = 0, \ldots, K+1$. To simplify notation we will denote these values by $\alpha_0, \ldots, \alpha_{K+1}$, so $\alpha_i = f(\tau_i)$ for $i = 0, \ldots, K+1$. Also, for this and other quantities we will use the shorthand $\alpha_{i:j}$ for integers $i \leq j$ to be the ordered set of values, $\alpha_i, \ldots, \alpha_j$.

2.1. An L_0 penalized criteria

To estimate the number and locations of the changes-in-slope, and the underlying signal, we will first introduce a grid of x-values, $g_{1:N}$ with these ordered so that $g_i < g_j$ if and only if

$i < j$. Our estimate for $f(x)$ will be restricted to piecewise-linear functions whose slope is only allowed to change at these grid-points. We will define our estimator of $f(x)$ as the function that minimizes a penalized cost that is a sum of the fit of the function to data, measured in terms of a weighted residual sum of squares, plus a penalty for each change-in-slope. That is we solve the following minimization problem

$$\min_{K,\tau_{1:K}\in g_{1:N},\alpha_{0:K+1}} \left\{ \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left(y_i - \alpha_{j(i)} - (\alpha_{j(i)+1} - \alpha_{j(i)}) \frac{x_i - \tau_{j(i)}}{\tau_{j(i)+1} - \tau_{j(i)}} \right)^2 + K\beta \right\}, \qquad (2)$$

where $\beta > 0$ is a user chosen penalty for adding a changepoint, $j(i)$ is such that $\tau_{j(i)} \leq x_i < \tau_{j(i)+1}$, and $\sigma_{1:n}^2$ are user specified constants that are estimates of the variances of the noise $\epsilon_{1:n}$. The cost that we are minimizing consists of two terms. The first is the measure of fit to the data, and is a residual sum of squares, but with the residuals weighted by the inverse of the variance of the noise for that observation. The expression in this term that depends on $\alpha_{0:K+1}$ and $\tau_{0:K+1}$ is just an expression for $f(x_i)$ given that $f(x)$ is defined as the linear interpolation between the points (τ_i, α_i) for $i = 0, \ldots, K+1$. The second term is the penalty for the number of changes-in-slope, with a penalty of β for each change.

This approach for estimating changes-in-slope was first proposed in Fearnhead *et al.* (2019), but they assumed that the locations of the data points were evenly spaced, so $x_i = i$, the grid-points were equal to the locations of the data points, so $N = n$ and $g_{1:N} = x_{1:n}$, and that the noise was homogeneous so $\sigma_i^2 = \sigma^2$, for some constant σ^2, for all i.

Before we describe how to extend the approach in Fearnhead *et al.* (2019) to this more general estimator, we first give some comments on this and related approaches to estimating changes-in-slope. This approach is a common one for estimating changes (Jackson *et al.* 2005; Killick *et al.* 2012) and the cost is often termed an L_0 penalized cost. This is to contrast it with L_1 penalized costs, such as implemented in trend-filtering (Kim *et al.* 2009; Tibshirani 2014) which are of similar form, except that the cost for adding a change-in-slope is linear in the size of the change-in-slope. The advantage of using an L_1 penalized cost is that solving the resulting minimization problem is simpler – however such an approach tends to over-estimate the number of changes, as shown in the introduction. An alternative approach to estimating changes-in-slope is to perform tests for a change-in-slope on data from randomly chosen intervals of x-values and to combine the results of these tests using the narrowest-over-threshold procedure of Baranowski *et al.* (2019). The current formulation and implementation of these alternative methods also make the simplifying assumptions of evenly spaced locations for the data, change-in-slope only at data point locations, and that the noise variance is constant.

The choice of β in (2) is important for accurate estimates, with lower values of β leading to larger estimates of K, the number of changes. If the noise is approximately Gaussian and independent, and the estimate of the noise variance is good, then $\beta = 2 \log n$ is an appropriate choice (Fearnhead *et al.* 2019). In general these assumptions will not hold, and often larger values of β are required to compensate for positively auto-correlated noise. Where possible we recommend evaluating sets of estimated changepoints obtained for a range of β values, and these can be obtained in a computationally efficient manner using the CROPS algorithm of Haynes *et al.* (2017a).

2.2. Dynamic programming recursion

Solving (2) is non-trivial as it involves minimizing a non-convex function. Furthermore, standard dynamic programming algorithms for change points (Maidstone *et al.* 2017), e.g., those that recurse based on conditioning on the location of the most recent changepoint, cannot be used because of the dependence across changepoints due to the continuity constraint. Thus we follow Fearnhead *et al.* (2019) and develop a dynamic programming recursion that conditions both on the location of the changepoints and the value of the mean at that changepoint.

Remember that we are allowing changepoints only at grid-points, $g_{1:N}$. We introduce a set of functions, each associated with a grid-point, $F_t(\alpha)$ for $t = 1, \ldots, N$, defined as the minimum value of the penalized cost for data up to and including grid-point g_t conditional on the signal at g_t being α, i.e., $f(g_t) = \alpha$. To define this formally, define a set of segment cost functions

$$\mathcal{C}_{s,t}(\alpha', \alpha) = \sum_{i=n_s+1}^{n_t} \left(y_i - \alpha' - (\alpha - \alpha') \frac{x_i - g_s}{g_t - g_s} \right)^2,$$

which is the cost of fitting a linear signal to data points between the s-th and t-th grid-points, i.e., (x_i, y_i) with $g_s < x_i \leq g_t$, with the signal taking the value α' at g_s and α at g_t. In the summation we use the notation n_s to denote the index of the last data-point located at or before g_s. If $n_t = n_s$, that is there are no data-points between g_s and g_t then this cost is set to 0.

Using this definition, the L_0 penalized criteria (2) can be written as

$$\min_{K, i_{1:K} \in 1:N, \alpha_{0:K+1}} \left\{ \sum_{k=0}^{K} \mathcal{C}_{i_k, i_{k+1}}(\alpha_k, \alpha_{k+1}) + K\beta \right\}. \tag{3}$$

Furthermore, we can define the function $F_l(\alpha)$ for $l = 1, \ldots, N$ as

$$F_l(\alpha) = \min_{K, i_{1:K} \in 1:l-1, \alpha_{0:K}} \left\{ \sum_{k=0}^{K-1} \mathcal{C}_{i_k, i_{k+1}}(\alpha_k, \alpha_{k+1}) + K\beta + \mathcal{C}_{i_K, l}(\alpha_K, \alpha) \right\},$$

which is of the form of (3) but with $\tau_{K+1} = g_l$ and $\alpha_{K+1} = \alpha$, as for $F_l(\alpha)$ we are analyzing only data up to g_l and we are fixing $\alpha_{K+1} = f(g_l) = \alpha$.

Using the same argument as in Fearnhead *et al.* (2019) we can then derive a recursion for $F_l(\alpha)$. For $l = 1, \ldots, N$,

$$F_l(\alpha) = \min_{k \in 0:(l-1)} \left\{ \min_{\alpha'} \left[F_k(\alpha') + \mathcal{C}_{k,l}(\alpha', \alpha) + \beta \right] \right\},$$

with $F_0(\alpha) = -\beta$. The idea of this recursion is that we condition on the location of the most recent changepoint, g_k, and the value of the signal at that changepoint, α'. Conditional on this information the optimal segmentation prior to the most recent changepoint is independent of the data since that changepoint, and the minimum of the penalized cost is $F_k(\alpha') + \mathcal{C}_{l-1,l}(\alpha', \alpha) + \beta$: the sum of the minimum cost for the data prior to g_k, plus the segment cost for the data from g_k to g_l plus the penalty for adding a changepoint. Finally we obtained $F_l(\alpha)$ by minimizing over k and α'.

Solving this recursion is possible as the functions $F_l(\alpha)$ can be summarized as the pointwise minimum of a set of quadratics. For each k, we can solve the inner minimization over α'

analytically. Doing so for each k will define $F_l(\alpha)$ as the pointwise minimum of a large set of quadratics, and we can use a line search to then prune quadratics that do not contribute to this minimum (which is important to reduce the computational cost of the algorithm). See Fearnhead *et al.* (2019) for full details. As noted there, it is possible to further reduce the computational cost of solving the recursion by using PELT pruning ideas from Killick *et al.* (2012). This pruning enables us to reduce the search space of k within the recursion. Finally, whilst we have described how to find the minimum value of the penalized cost, our main interest is in the locations of the changepoints and the value of the signal that gives that minimum cost. However extracting this information is trivial once we have solved the recursions – again see Fearnhead *et al.* (2019) for details.

The novelty relative to Fearnhead *et al.* (2019) is that for this recursion we have decoupled the grid of potential changepoints from the locations of the data points. Furthermore, our setting allows for unevenly spaced data and for the noise variance to be heterogeneous. These impact that definition of $\mathcal{C}_{l-1,l}(\alpha', \alpha)$ and how we perform the inner minimization over α'. Full details are given in Appendix A.

One further extension of this approach to detecting changes is to allow for a minimum segment length. This can be done by optimizing the penalized cost (2) only over sets of changepoints that are consistent with the prescribed minimum segment length. To minimize the cost subject to such a constraint involves adapting the dynamic programming recursion so that we only search over values of k for the location of the most recent changepoint that are consistent with the minimum segment length. However one drawback with imposing a minimum segment length for the change-in-slope problem is that it makes the PELT pruning ideas invalid. In our implementation of `cpop`, if a minimum segment length is specified we allow the algorithm to be run both with and without the PELT pruning. If run without PELT pruning, the algorithm will be slower but guaranteed to find the optimal segmentation under our condition. Running with PELT pruning is quicker but the algorithm may output a slightly sub-optimal segmentation. In practice we have observed that this happens rarely.

3. The cpop package

The **cpop** package has functions to simulate data from a change-in-slope model, implement the CPOP algorithm to estimate the location of the changes, and various functions for summarizing and plotting the estimates of the change locations and the mean function.

3.1. Generating simulated data

The `simchangeslope` function allows for simulating data from a change-in-slope model (1):

```
simchangeslope(x, changepoints, change.slope, sd = 1)
```

It takes the following arguments:

- `x`: A numeric vector containing the locations of the data.
- `changepoints`: A numeric vector of changepoint locations.
- `change.slope`: A numeric vector indicating the change in slope at each changepoint. The initial slope is assumed to be 0.

- `sd`: The residual standard deviation. Can be a single numerical value or a vector of values for the case of varying residual standard deviation. Default value is 1.

It returns a vector y of simulated values which correspond to the locations x. The mean function of the data goes through the origin – but to add an intercept we just add a constant to all output values. It is possible to get the value of the mean function at the x-values of the data by setting `sd = 0`.

The following code demonstrates the `simchangeslope` function and displays the data along with the (true) line segments and the locations of the changes in slope (see Figure 2).

```
R> library("cpop")
R> library("ggplot2")
R> changepoints <- c(0, 25, 50, 100)
R> change.slope <- c(0.2, -0.3, 0.2, -0.1)
R> x <- 1:200
R> sd <- 0.8
R> y <- simchangeslope(x, changepoints, change.slope, sd)
R> df <- data.frame("x" = x, "y" = y)
R> p <- ggplot(data = df, aes(x = x, y = y))
R> p <- p + geom_point(alpha = 0.4)
R> p <- p + geom_vline(xintercept = changepoints, color = "red",
+    linetype = "dashed")
R> mu <- simchangeslope(x, changepoints, change.slope, sd = 0)
R> p <- p + geom_line(aes(y = mu), color = "blue")
R> p <- p + theme_bw()
R> print(p)
```

3.2. Determining changes in slope

The function `cpop` is used to determine the locations of changes in slope.

```
cpop(y, x = 1:length(y) - 1, grid = x, beta = 2 * log(length(y)),
  sd =  sqrt(mean(diff(diff(y))^2)/6), minseglen = 0, prune.approx = FALSE)
```

It takes the following arguments:

- `y`: A vector of length n containing the data.
- `x`: A vector of length n containing the times/locations of data points. Default value is `NULL`, in which case the locations are set to be $0, 1, \ldots, n-1$, corresponding to evenly spaced data.
- `grid`: An ordered vector of possible locations for the change points. If this is `NULL`, then this is set to x, the vector of times/locations of the data points.
- `beta`: A positive real value for the penalty, β in (3), incurred for adding a changepoint. The larger the penalty, the fewer changepoints will be detected. The default value is `beta = 2 * log(length(y))`.

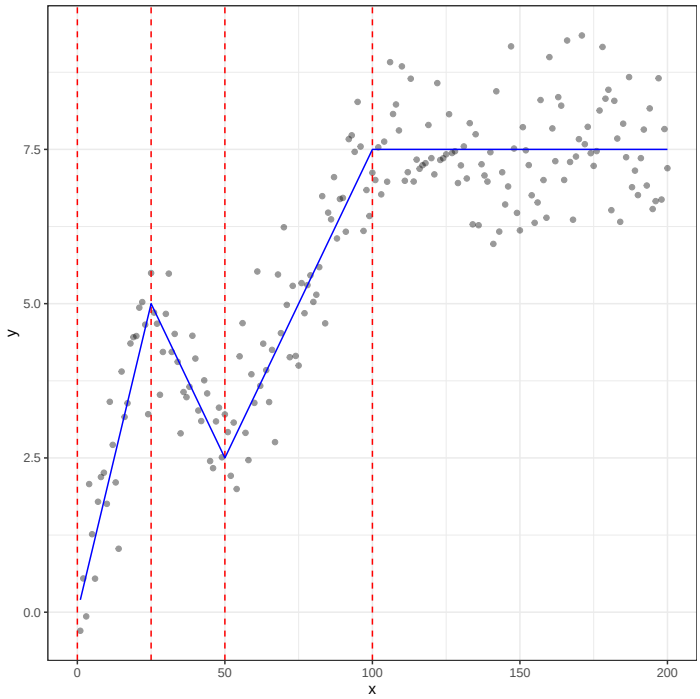

Figure 2: Simulated data (black dots) with true mean (blue dashed line) and changepoints (vertical red dashed lines).

- `sd`: Estimate of residual standard deviation. Can be a single numerical value if it is the same for all data points, or a vector of n values for the case of varying standard deviation. The default value is `sd = sqrt(mean(diff(diff(y))^2)/6)`.

- `minseglen`: The minimum allowable segment length, that is the distance between successive changepoints. The default is that no minimum segment length is imposed.

- `prune.approx`: Only relevant if a minimum segment length is set. If `TRUE`, `cpop` will use an approximate pruning algorithm that will speed up computation but may occasionally lead to a sub-optimal solution in terms of the estimated changepoint locations. If the minimum segment length is 0, then an exact pruning algorithm is possible and is used.

The `cpop` function returns an S4 object for which a number of generic methods, including `plot` and `summary`, are provided. The default value for `sd` is based on a simple estimator that is appropriate for regularly-spaced data. In this case, taking the first difference of `y` twice will remove the linear trend within each segment, and the variance of the resulting twice-differenced data is an estimator of 6 times the noise variance.

The following demonstrates how the `cpop` function can be used to determine changes in slope, by analyzing the data we simulated and plotted in Figure 2. It uses the default penalty, $\beta = 2 \log n$, and we assume that the true noise standard deviation, 0.8, is known. The `summary` function is used to provide an overview of the analysis parameters along with estimated changepoint locations, corresponding fitted line segments, and the (weighted) residual sum of squares (RSS) for each segment.

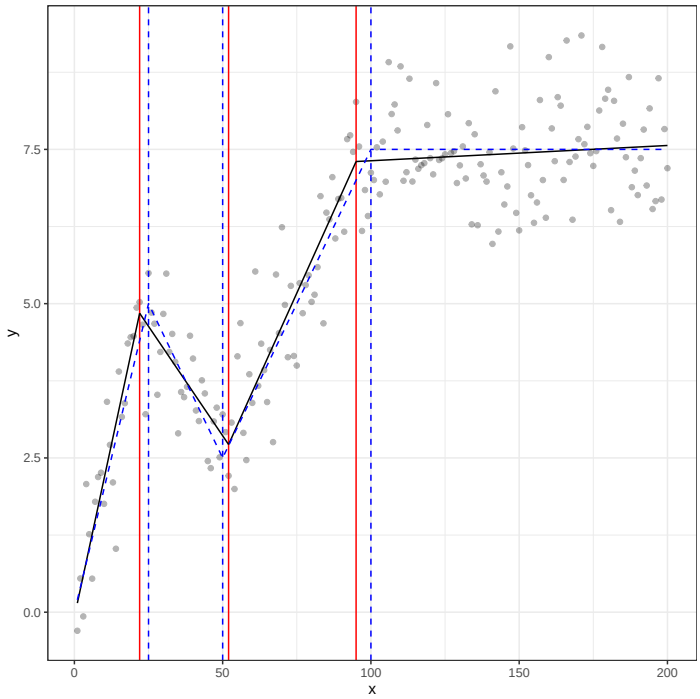

Figure 3: Example output of `cpop` for simulated data from Figure 2. The true mean and changepoints are given in blue dashed lines, together with estimated mean (black full line) and changepoints (red full lines).

```
R> res <- cpop(y, x, beta = 2 * log(length(y)), sd = 0.8)
R> summary(res)

cpop analysis with n = 200 and penalty (beta)  = 10.59663

3  changepoints detected at x =
 22 52 95
fitted values :
  x0       y0  x1       y1     gradient   intercept      RSS
1  1 0.147335  22 4.844725  0.223685242 -0.07635023 10.07761
2 22 4.844725  52 2.717661 -0.070902123  6.40457180 10.38813
3 52 2.717661  95 7.303644  0.106650750 -2.82817758 25.09463
4 95 7.303644 200 7.563413  0.002473995  7.06861408 61.78303

overall RSS = 107.3434
cost = 199.514
```

The predicted change in slope (changepoint) locations and corresponding line segments can be displayed using `plot`. The `plot` function returns a 'ggplot2' object which can be augmented to include additional features such as the true change in slope locations and line segments (see Figure 3).

```
R> p <- plot(res)
```

```
R> p <- p + geom_vline(xintercept = changepoints[-1], color = "blue",
+    linetype = "dashed")
R> p <- p + geom_line(aes(y = mu), color = "blue", linetype = "dashed")
R> print(p)
```

The last two lines of code add the true changepoint locations and the true mean function to the plot. For plotting the changepoint location we omit the first element of `changepoints`, which was 0, as that was included just to set the initial slope of the mean and does not correspond to a change-in-slope.

The estimate of the number of changes will depend on both the value of the penalty, `beta`, and the assumed standard deviation of the noise, `sd`. By considering the form of the criteria (2) that `cpop` minimizes we see that if we multiply `sd` by some constant c and `beta` by c^2 then we will obtain the same set of estimated changes. The function `cpop` has default values, with `sd` estimated based on the second moment of the second differences of the data (as second differences removes a linear signal), and `beta` set to $2 \log n$, where n is the length of the data. This is a standard default penalty which has good properties if the noise is independent, identically distributed (IID) and Gaussian, and `sd` is a good estimate of the noise standard deviation. How to use `cpop` when these assumptions do not hold, or the noise standard deviation varies or is hard to estimate is discussed in the examples below.

3.3. Other functions

In addition to `plot` and `summary`, the **cpop** package provides functions to evaluate the fitted mean function at specified x-values, and to calculate the residuals of the fitted mean. The primary argument of these functions is `object`, an instance of a '`cpop`' S4 class as produced by the function `cpop`.

The function `changepoints(object)` creates a data frame containing the locations of the changepoints in terms of the their x-values.

```
R> changepoints(res)
```

```
  location
1       22
2       52
3       95
```

The function `estimate(object, x = object@x, ...)` with argument, `x`, that specifies the x-values at which the fit is to be estimated, creates a data frame with two columns containing the locations `x` and the corresponding estimates $\hat{y}$. The default value for `x` is the vector of x locations at which the '`cpop`' object was defined.

```
R> estimate(res, x = c(0.1, 2.7, 51.6))
```

```
   x      y_hat
1  0.1 -0.0539817
2  2.7  0.5275999
3 51.6  2.7460223
```

The function `fitted(object)` creates a data frame containing the endpoint coordinates for each line segment fitted between the detected changepoints. The data frame also contains the gradient and intercept values for each segment and the corresponding residual sum of squares (RSS).

```
R> fitted(res)
```

```
  x0       y0  x1       y1     gradient   intercept      RSS
1  1 0.147335  22 4.844725  0.223685242 -0.07635023 10.07761
2 22 4.844725  52 2.717661 -0.070902123  6.40457180 10.38813
3 52 2.717661  95 7.303644  0.106650750 -2.82817758 25.09463
4 95 7.303644 200 7.563413  0.002473995  7.06861408 61.78303
```

The function `residuals(object)` creates a single column matrix containing the residuals.

```
R> head(residuals(res))
```

```
           [,1]
[1,] -0.4484981
[2,]  0.1758944
[3,] -0.6632084
[4,]  1.2578339
[5,]  0.2215302
[6,] -0.7221359
```

4. Extensions of cpop

4.1. Irregularly sampled data

The **cpop** package allows for irregularly spaced data, both when simulating data and when running the CPOP algorithm. The only change to the previous code that we need to make is to change the definition of `x` that is input to `simchangeslope`.

```
R> x <- (1:200)^2/200
R> changepoints <- c(0, 25, 50, 100)
R> change.slope <- c(0.2, -0.3, 0.2, -0.1)
R> sd <- 0.8
R> y <- simchangeslope(x, changepoints, change.slope, sd)
```

To analyse the data we use `cpop` as before. (The only difference is that for evenly spaced data one can omit the `x` argument – but it must be included for unevenly spaced data.)

```
R> res <- cpop(y, x, beta = 2 * log(length(y)), sd = 0.8)
```

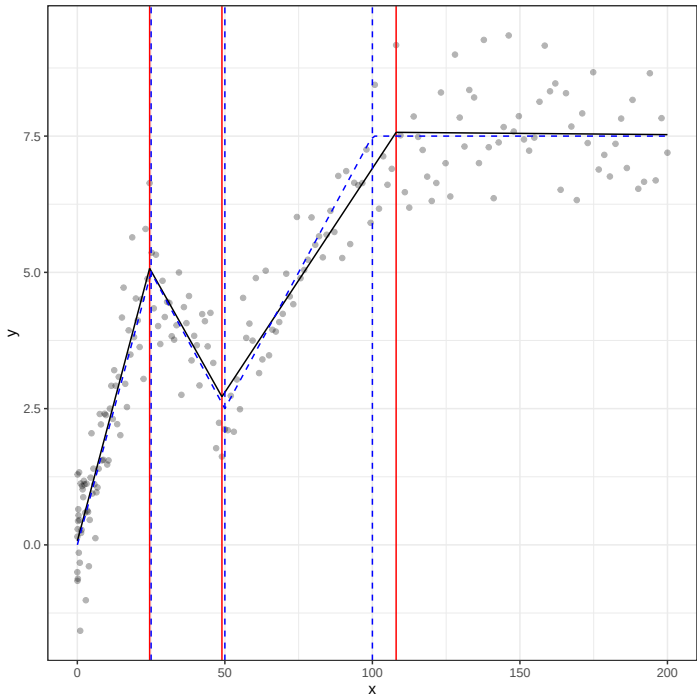

Figure 4: Example output of `cpop` for unevenly spaced simulated data. The true mean and changepoints are given in blue dashed lines, together with estimated mean (black full line) and changepoints (red full lines).

Figure 4 shows a plot of the simulated data and estimated changepoints and mean function.

4.2. Heterogeneous data

To simulate heterogeneous data we just input a vector of the standard deviation of each data point. For example, we can produce a version of the simulation from Section 3 but with the noise standard deviation increasing with x.

```
R> x <- 1:200
R> sd <- x/100
R> y <- simchangeslope(x, changepoints, change.slope, sd)
```

Here the values of `changepoints` and `change.slope` are as before.

It is interesting to compare two estimates of the changepoints, one where we assume a fixed noise standard deviation, and one where we assume the true noise standard deviation. For the former it is natural to set this value so the the average variance of the noise is correct.

```
R> res <- cpop(y, x, beta = 2 * log(length(y)), sd = sqrt(mean(sd^2)))
R> res.true <- cpop(y, x, beta = 2 * log(length(y)), sd = sd)
```

Here `res` contains the results where we assume a fixed noise standard deviation, and `res.true` where we use the true values. Figure 5 shows the results – and we can see that wrongly

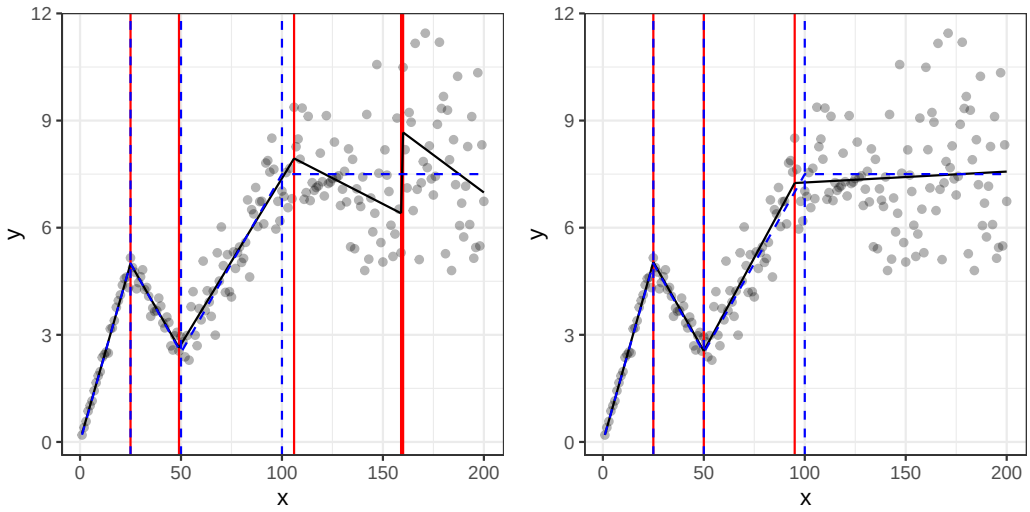

Figure 5: Example output of `cpop` for heterogeneous noise: assuming a constant noise variance (left) and the true noise variance (right). The true mean and changepoints are given in blue dashed lines, together with estimated mean (black full line) and changepoints (red full lines).

assuming homogeneous noise leads to detecting two false positive changepoints in regions where the noise variance is above what was assumed.

One practical issue is how can we estimate the noise variance in the heterogeneous case? In some situations there may be covariate information that tells the relative variance of the noise for different data points (for example due to some data being averages of multiple measurements). Alternatively if we know how the noise variance depends on x we can estimate this by (i) running CPOP assuming a constant variance; (ii) calculating the residuals of the fitted model; (iii) estimating how the noise variance varies with x by fitting an appropriate model to the residuals. An example of this scheme will be seen in Section 5.

4.3. Choice of grid

The computational cost for `cpop` increases with the size of the number of potential changepoint locations. To see the rate of increase we ran `cpop` with the default settings where the grid of potential changepoints is equal to the x values of the data, for data sizes varying from $n = 200$ to $n = 6400$. We considered two scenarios, one where we had a fixed number of changepoints and one where we had a fixed segment size of length 100. The average CPU (central processing unit) cost across 10 runs of `cpop` for each data size are shown in Figure 6. The figure suggests that the computational cost is increasing like $n^{2.5}$ when we have a fixed number of changes, and like $n^{1.7}$ when the number of changes increases linearly with n. By comparison, if we analyse each data set with a grid of 200 evenly spaced potential locations for the changes, the computational cost is roughly constant. We see the computational costs are similar for both regularly and irregularly spaced data.

Thus for large data sets, we can substantially reduce the computational cost of running `cpop` by using a smaller grid of potential change locations. Obviously this comes with the drawback of a potential loss of accuracy with regards to the estimated changepoint locations. However one possible approach is to run `cpop` with a coarse grid, and then re-run the algorithm with a finer grid around the estimated changepoints.

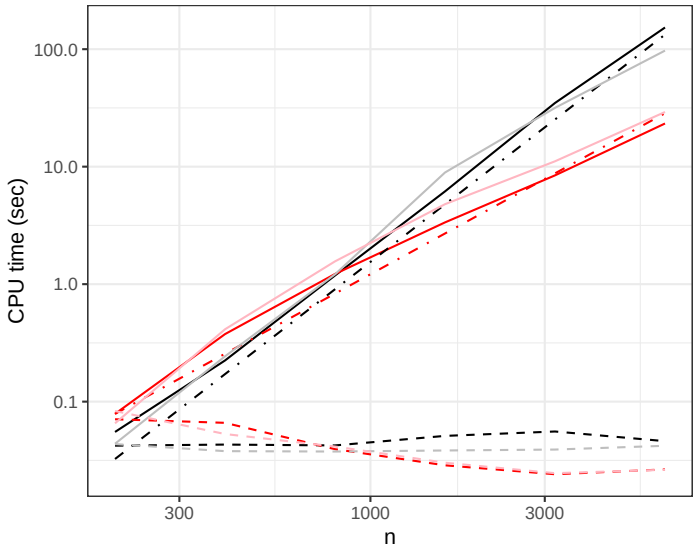

Figure 6: Empirical computational cost for `cpop` as a function of sample size, n, for a grid of size n (full lines) and of size 200 (dashed lines): for regularly spaced data with a single changepoint (black) and for a linearly increasing number of changepoints (red); and for data with x-values simulated from a uniform distribution with a single changepoint (grey) and a linearly increasing number of changepoints (pink). To aid interpretation straight lines for CPU cost proportional to $n^{1.7}$ (red dot-dashed) and $n^{2.5}$ (black dot-dashed) are shown.

To see this we implemented the scheme for a data set with $n = 6400$ and a fixed segment size of 200. We initially ran `cpop` with a grid with potential changes allowed every 16 observations.

```
R> x <- 1:6400
R> y <- simchangeslope(x, changepoints = 0:31 * 200,
+    change.slope = c(0.05, 0.1 * (-1)^(1:31)), sd = 1)
```

We use a smaller value for the penalty due to the smaller grid size, and the fact that this is a preliminary step to find roughly where the changes are: so the key is to avoid missing changes. Spurious changes can still be removed when we perform our final run of `cpop`.

```
R> res.coarse <- cpop(y, x, grid = 1:399 * 16, beta = 2 * log(400), sd = 1)
```

In our example we find 38 changepoints with this coarse grid. We then introduce a finer grid around these putative changes: our new grid includes all x-values within 8 of each putative changepoint.

```
R> cps <- unlist(changepoints(res.coarse))
R> grid <- NULL
R> for(i in 1:length(cps)) {
+    grid <- c(grid, cps[i] + (-7):8)
+  }
R> res.fine <- cpop(y, x, grid, beta = 2 * log(length(x)), sd = 1)
```

This gives a computational saving of between 10 to 100 over the default running of `cpop`. We can evaluate the accuracy of the approach by then comparing the estimated changepoints to the estimates we obtain if we run the default setting of `cpop`. In this case, both runs estimate the same number of changepoints, with the maximum difference in the location of a change being two time-points. The slower, default running of `cpop` gives a segmentation with a marginally lower cost (of 7187.1 as opposed to 7188.0).

4.4. Imposing a minimum segment length

The `cpop` function allows the user to specify a minimum segment length – and this is defined as the minimum x-distance allowed between two estimated changepoints. Specifying a minimum segment length can make the method more robust to point outliers or noise that is heavier-tailed than Gaussian: as minimizing (2) can lead to over-fitting in such scenarios and this over-fitting tends to be through adding clusters of changepoints close together to fit the noise in the data. There are two disadvantages of imposing a minimum segment length. First it can cause true changes to be missed if they are closer together than the specified minimum segment length. Second `cpop` is slower when a minimum segment length is imposed.

To see these issues, we simulated data as in Section 3, except that we assumed the noise was t_4 distributed. We cannot simulate such data directly with `simchangeslope`, so we need to first use `simchangeslope` to calculate the mean function, and then add the noise:

```
R> changepoints <- c(0, 25, 50, 100)
R> change.slope <- c(0.2, -0.3, 0.2, -0.1)
R> x <- 1:200
R> mu <- simchangeslope(x, changepoints, change.slope, 0.0)
R> y <- mu + rt(length(x), df = 4)
```

We then estimated the changepoint locations both without a minimum segment length, and with minimum segment lengths of 10, 30 and 40. To run `cpop` with a minimum segment length of 10:

```
R> res.min <- cpop(y, x, beta = 2 * log(length(y)), minseglen = 10,
+    sd = sqrt(2))
```

The argument `sd = sqrt(2)` is because t_4 distributed noise has a variance of 2.

Results from CPOP with different minimum segment lengths are shown in Figure 7. If we do not impose a minimum segment length, then we estimate 11 changepoints, including three cluster of changes that overfit to the noise. By imposing a minimum segment length of 10 we avoid the over-fitting. For this example, the computational cost of running `cpop` with the minimum segment length is about 6 times larger than when we do not assume a minimum segment length.

Assuming a minimum segment length of 30 or 40 shows what can happen when our minimum segment length assumption does not hold. A minimum segment length of 30 leads to estimates of the first two changes at time-points 30 and 60 – the closest possible given the assumption. As we increase the minimum segment length to 40 we miss the first changepoint all together.

4.5. Choice of penalty

The choice of the penalty, `beta`, in `cpop` can have a substantial impact on the number of

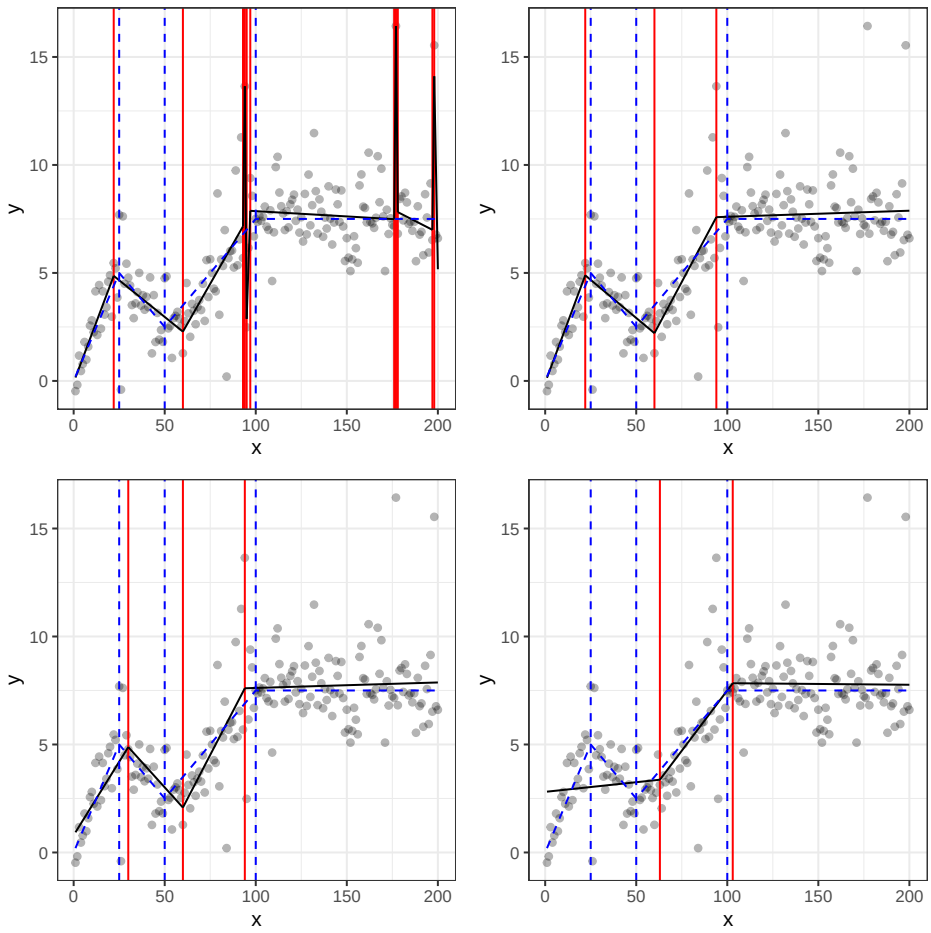

Figure 7: Results of analyzing data with t_4 noise with no minimum segment length (top left) and minimum segment lengths of 10 (top right), 30 (bottom left) and 40 (bottom right). In each plot we show data (grey dots), true mean (blue dashed line), true changepoints (blue vertical dashed lines), estimated mean (black line) and estimated changepoints (red vertical liens)

changepoints detected and the accuracy of the estimated mean function. This is common to all changepoint methods, where there will be at least one tuning parameter that specifies the evidence for a change that is needed before a change is added. The default choice of penalty, $2 \log n$ where n is the data size, is based on assumptions that the noise is IID Gaussian with known variance. When these assumptions do not hold, it is recommended to look at the segmentations obtained as the penalty value is varied: this can be done efficiently using the CROPS algorithm of Haynes *et al.* (2017a).

The idea of CROPS is that it allows a penalized cost method to be implemented for all penalty values in an interval. This is implemented within the **cpop** package by the function:

```
cpop.crops(y, x = 1:length(y), grid = x, beta_min = 1.5 * log(length(y)),
  beta_max = 2.5 * log(length(y)), sd =  sqrt(mean(diff(diff(y))^2)/6),
  minseglen = 0, prune.approx = FALSE)
```

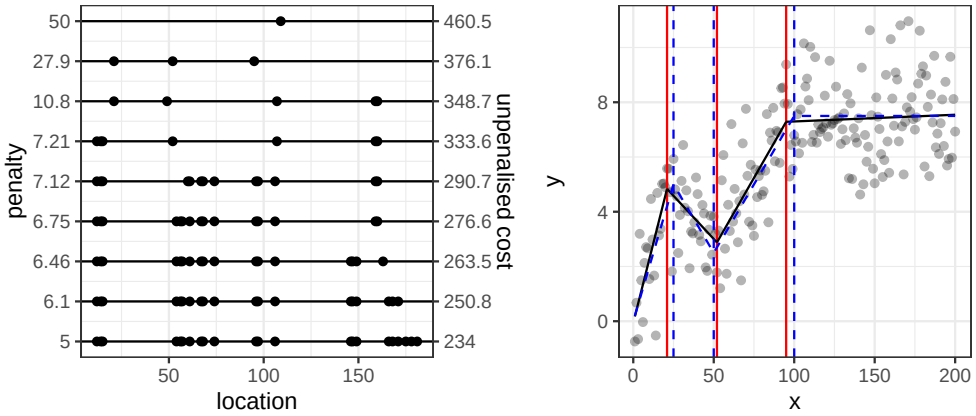

Figure 8: Example plot of output from `cpop.crops` (left), and best segmentation based on calculated `BIC` (right). For the left-hand plot each row shows a segmentation, with points at estimated changepoint location. The left-hand axis shows a penalty value that leads to that segmentation, and the right-hand axis gives the corresponding unpenalized cost. For the right-hand plot we show the true mean and changepoints (blue dashed lines), and estimated mean (black line) and changepoints (red lines).

The arguments of `cpop.crops` are identical to those of `cpop` except that, rather than specifying a single penalty value (`beta`), the range of penalty values to be used is specified by `beta_min` and `beta_max`, which fix the smallest and largest penalty value to be used. The output is an instance of an S4 class that contains details of all segmentations found by minimizing the penalized cost for some penalty value in the interval between `beta_min` and `beta_max`.

To see the use of `cpop.crops`, consider an application where we do not know the standard deviation of the noise. Under our criteria (2), the optimal segmentation with penalty $c^2\beta$ and standard deviation σ_i/c will be the same if we fix β and $\sigma_{1:n}$ but vary $c > 0$. Thus under an assumption that the noise is homogeneous, we can run `cpop` with `sd = 1` but for a range of $\beta \in [2\sigma_-^2 \log n, 2\sigma_+^2 \log n]$, and this will give us the optimal segmentations for $\beta = 2\log n$ as we vary noise standard deviation between σ_- and σ_+.

We simulated data as per Section 3 but with `sd = 1.5`. To run CPOP for a range of $\beta \in [5, 50]$ we use

```
R> res.crops <- cpop.crops(y, x, beta_min = 5, beta_max = 50, sd = 1)
```

For our example $2\log n = 10.6$, so this is equivalent to trying noise standard deviation in the range $[0.69, 2.2]$.

We can plot the location in the changepoints for each segmentation found by CPOP for $\beta \in [5, 50]$.

```
R> plot(res.crops)
```

This is shown in Figure 8, and shows that there are 6 different segmentations found. These are labelled with a penalty value which gives that segmentation (left axis) and the unpenalized cost, i.e., the weighted RSS, for that segmentation and penalty value (right axis).

Details of the segmentations can be obtained using `segmentations(res.crops)`. This gives a matrix with one row for each of the segmentations, and each row contains a value of β that gives that segmentation, the corresponding unpenalized cost, the penalized cost, the number of changepoints, and then the list of ordered changepoints. We can also obtain a list with the output from `cpop` corresponding to each segmentation, with `models(res.crops)`. For example, one approach to choose a segmentation from the output from `cpop.crops` is to find the segmentation that minimizes a cost under a model where we assume the noise variance is unknown (Fryzlewicz 2014),

$$n \log \left(\frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(x_i) \right)^2 \right) + 2K \log n.$$

Here $\hat{f}$ is the estimated mean function and K is the number of changepoints. This can be calculated as follows.

```
R> models <- cpop.crops.models(res.crops)
R> M <- length(models)
R> BIC <- rep(NA, M)
R> ncps <- segmentations(res.crops)[, 4]
R> n <- length(y)
R> for(j in 1:M) {
+    BIC[i] <- n * log(mean((residuals(models[[j]]))^2)) +
+      2 * ncps[i] * log(n)
+ }
```

This uses that the fourth column of the matrix `segmentations(res.crops)` stores the number of changepoints in each segmentation, and that we can calculate $y_i - \hat{f}(x_i)$ using the `residual` function evaluated for the corresponding entry of `cpop.crops.models(res.crops)`. The segmentation which has the smallest value of `BIC` is shown in Figure 8, and shows that this correctly chooses the segmentation with three changes.

As a final example, we performed a similar analysis but with correlated noise. This violates the assumption of IID noise that underpins the default choice of penalty, thus we run `cpop.crops` for a range of penalties.

We simulated data with $n = 500$ data points and 10 equally spaced changepoints.

```
R> n <- 500
R> x <- 1:n
R> mu <- simchangeslope(x, changepoints = 45 * 0:10,
+    change.slope = c(0.15, 0.3 * (-1)^(1:10)), sd = 0)
R> epsilon <- rnorm(n + 2)
R> y <- mu + (epsilon[1:n] + epsilon[2:(n + 1)] + epsilon[3:(n + 2)])/sqrt(3)
```

The noise is MA(3), and we simulate the data by first calculating the mean function, `mu`, and then adding the MA(3) noise.

We could continue as above, and choose between the segmentations by minimizing a penalized cost under an appropriate model for the noise; this type of approach is suggested for change in mean models by Cho and Fryzlewicz (2024). A simpler, albeit more qualitative approach,

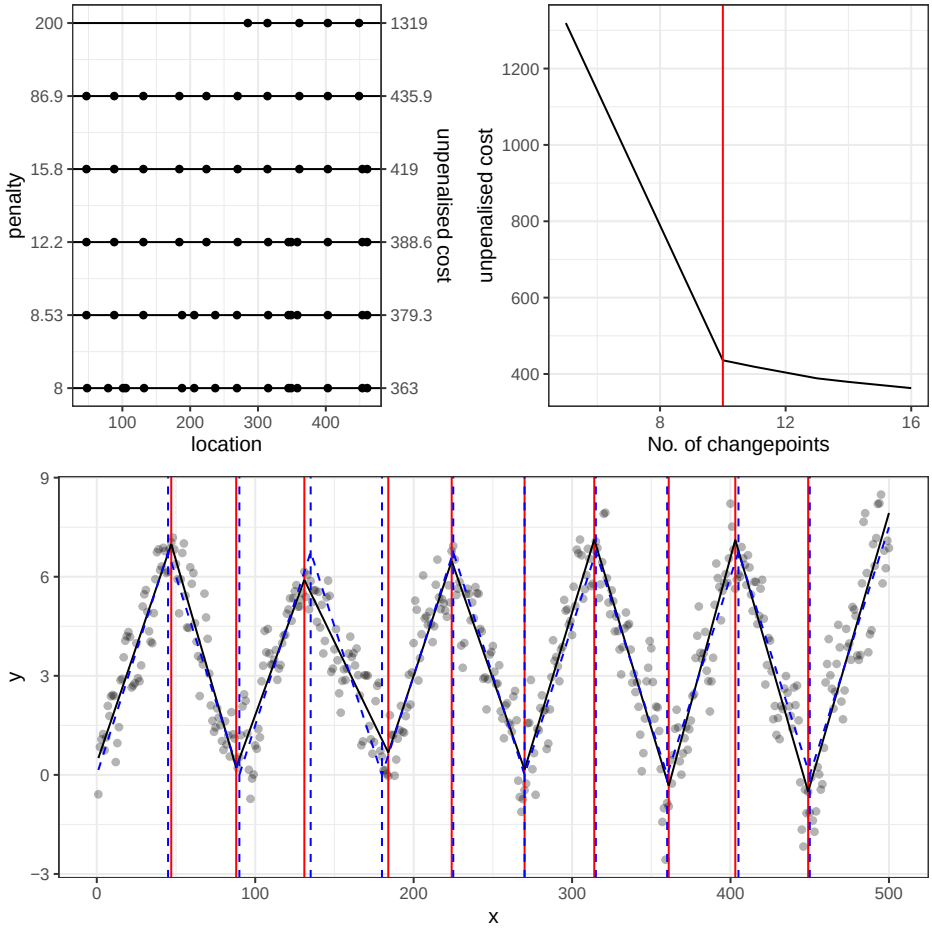

Figure 9: Correlated noise example. Output from `cpop.crops` (top left), unpenalized cost against number of changepoint (top right) and estimate from segmentation corresponding to "elbow" (bottom).

is to plot the residual sum of squares of the segmentation against the number of changepoints (Lebarbier 2005; Baudry, Maugis, and Michel 2012; Fearnhead and Rigaill 2020; Fryzlewicz 2020). This avoids the need to specify a model for the residuals. The idea of this approach is that adding "true" changes should lead to a noticeably larger reduction in the residual sum of squares than adding "spurious" changes. Thus the best segmentation should correspond to an "elbow" in this plot.

```
R> res.crops <- cpop.crops(y, x, beta_min = 8, beta_max = 200, sd = 1)
R> segs <- segmentations(res.crops)
R> p <- ggplot(data = segs, aes(x = m))
R> p <- p + geom_line(aes(y = Qm))
R> p <- p + geom_vline(xintercept = 10, color = "red")
R> p <- p + xlab("No. of changepoints") + ylab("unpenalized cost")
R> plot(p)
```

This runs `cpop.crops` and then uses the fact that the output of `segmentations` includes

columns that give the number of changepoints and the unpenalized cost of each segmentation. These columns are labelled `"m"` and `"Qm"` respectively. The plot gives a clear elbow, see Figure 9, and this corresponds to a correct estimate of the number of changes.

5. Application

We now demonstrate an application of `cpop` on analyzing power spectra of velocity as a function of wavenumber obtained from models of the Atlantic Ocean. The data is available in the **cpop** package and can be loaded with `data("wavenumber_spectra", package = "cpop")`. It contains four spectra, corresponding to two different months (February and August) from two different runs of the model (2000 and 2100) corresponding to present and future scenarios: see Figure 10. The data comes from Richards, Whitt, Brett, Bryan, Feloy, and Long (2021), and is available from Richards, Whitt, and Brett (2020). See Richards *et al.* (2021) for a fuller description of the data.

Interest lies in estimating the rate of decay of the log-spectra against log-wavenumber. We can do this by removing the first three data points (where the spectra is increasing) and then using `cpop` to fit a piecewise-linear curve to the remaining data. We perform an initial run of `cpop` assuming an estimated homogeneous noise variance on the data from August from the 2000 run.

```
R> data("wavenumber_spectra", package = "cpop")
R> x <- log(wavenumber_spectra[-(1:3), 1], base = 10)
R> y <- log(wavenumber_spectra[-(1:3), 4], base = 10)
R> grid <- seq(from = min(x), to = max(x), length = 200)
R> sig2 <- mean(diff(diff(y))^2)/6
R> res <- cpop(y, x, grid, sd = sqrt(sig2),
+    minseglen = 0.09, beta = 2 * log(200))
```

Here we estimate the noise variance, `sig2`, based on the variance of the double difference of the data. For regions where the mean is linear and the data is evenly spaced, taking the double difference will lead to a mean zero process. If the noise is IID with variance σ^2 then the double-differenced process will have a marginal variance that is $6\sigma^2$. Thus our estimate is the empirical mean of the square of the double-difference data divided by 6. The original data is evenly spaced in in terms of wavenumber, but as we take logs, `x` is unevenly spaced: so this estimator will be biased in our setting. However it will give a reasonable ball-park figure for an initial run of `cpop` – the residuals from which can then be used to get a better estimate of the noise variance. We use a evenly spaced grid for possible change-point locations. To avoid the potential for adding multiple changepoints between two observations, we set a minimum segment length of 0.09 (as the largest distance between consecutive `x` values is 0.08).

The output is shown in the top-right plot of Figure 10, and appears to be over-fitting to the early part of the series. This is because the noise variance is heterogeneous, and decreasing with `x`. However, given our initial fit we can use the residuals to estimate the noise variance. The noise for the spectra is expected to be approximately inversely proportional to the wavenumber. By using a Taylor-expansion, we have the variance of the noise for the log-spectra should be approximately the variance of the noise of the spectra divided by the square of the mean of the spectra. As the mean of the spectra is roughly a power of the wavenumber,

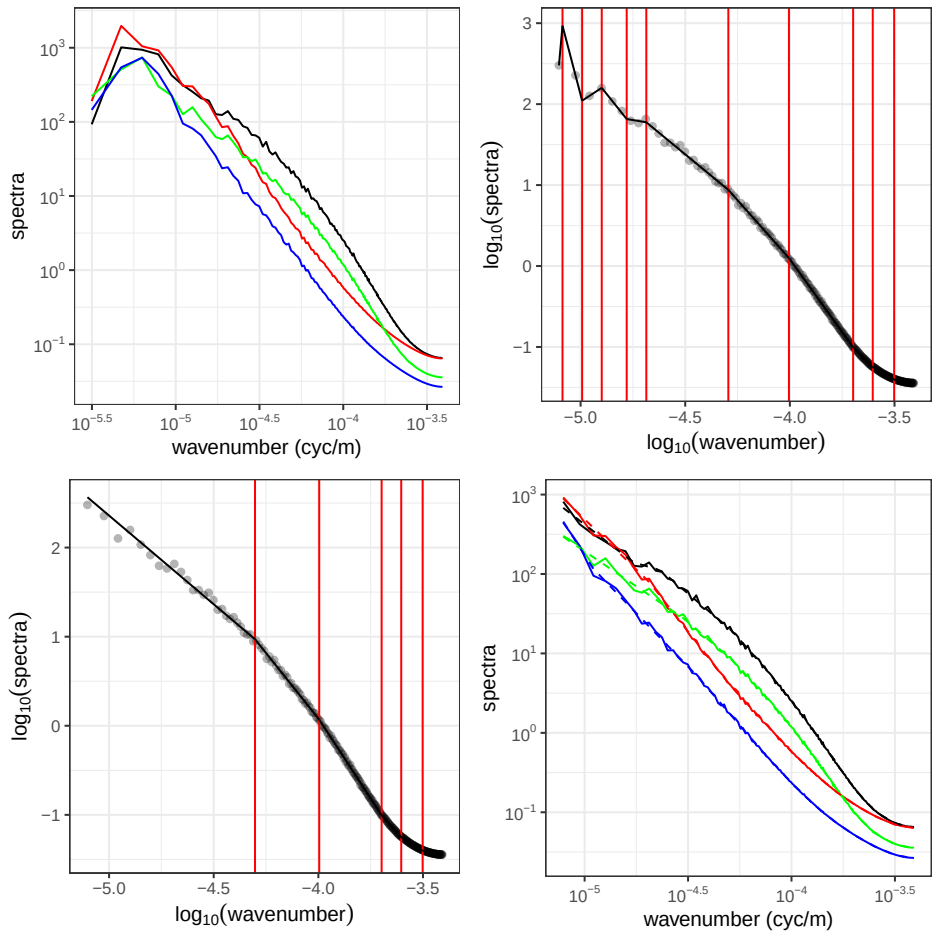

Figure 10: Application of `cpop` to `wavenumber_spectra` data. Log-log plot of raw data (top left) of horizontal wavenumber spectra of velocity for two months and two runs of an ocean model: February 2000 (black), August 2000 (red), February 2100 (green) and August 2100 (blue). Output from `cpop` applied to analyse the decay of spectra from August 2000, with y equal to log spectra and x equal to log wave number, with estimated homogeneous noise variance (top right) and estimated heterogeneous noise variance (bottom left). Log-log plot of fitted spectra for all four series (bottom right) with original data in full-lines and estimate in dashed lines.

this suggests using a model for the variance, σ_x^2 say, depending on x as $\log \sigma_x^2 = a + bx$: so that the variance is proportional to some power of the wavenumber. We can estimate the parameters of this model by maximizing the log-likelihood of Gaussian model for the residuals with this form for the variance.

```
R> r2 <- residuals(res)^2
R> loglik <- function(par) {
+    return(length(r2) * par[1] + par[2] * sum(x) +
+      sum(r2/(exp(par[1] + par[2] * x))))
+  }
R> est.hat <- optim(c(0, 0), loglik)
```

```
R> sig2 <- exp(est.hat$par[1] + est.hat$par[2] * x)
R> res2 <- cpop(y, x, grid, sd = sqrt(sig2),
+    minseglen = 0.09, beta = 2 * log(200))
```

Here we have calculated the maximum likelihood estimates by using `optim` to minimize minus the log-likelihood. The resulting output from `cpop` is shown in the bottom left plot of Figure 10. The first two changes could represent real regime transitions relating to the inviscid fluid physics that one would see in the real ocean (see Figure 6a of Callies and Ferrari 2013), while the three changes for the largest values of `x` may relate to a breakdown in the numerical ocean model near the highest wavenumber of the ocean model grid (Soufflet, Marchesiello, Lemarié, Jouanno, Capet, Debreu, and Benshila 2016). The estimates of the spectra for all four series, obtained by repeating this approach, is also shown in Figure 10. For this application, the residuals from the fitted model appear to be uncorrelated and sub-Gaussian, so using the fit based on the default penalty choice is reasonable. Though one could also explore segmentations for other penalty choices using `cpop.crops` as in the previous section.

Acknowledgments

We would like to thanks Jessica Luo and Dan Whitt for access to and help with the wavenumber spectra data. Paul Fearnhead acknowledges funding from EPSRC grant EP/N031938/1.

References

Aminikhanghahi S, Cook DJ (2017). "A Survey of Methods for Time Series Change Point Detection." *Knowledge and Information Systems*, **51**(2), 339–367. doi:10.1007/s10115-016-0987-z.

Anastasiou A, Chen Y, Cho H, Fryzlewicz P (2022). ***breakfast**: Methods for Fast Multiple Change-Point Detection and Estimation*. R package version 2.3, URL https://CRAN.R-project.org/package=breakfast.

Anastasiou A, Fryzlewicz P (2022). "Detecting Multiple Generalized Change-Points by Isolating Single Ones." *Metrika*, **85**(2), 141–174. doi:10.1007/s00184-021-00821-6.

Andreou E, Ghysels E (2002). "Detecting Multiple Breaks in Financial Market Volatility Dynamics." *Journal of Applied Econometrics*, **17**(5), 579–600. doi:10.1002/jae.684.

Arnold TB, Tibshirani RJ (2022). ***genlasso**: Path Algorithm for Generalized Lasso Problems*. R package version 1.6.1, URL https://github.com/glmgen/genlasso.

Auger IE, Lawrence CE (1989). "Algorithms for the Optimal Identification of Segment Neighborhoods." *Bulletin of Mathematical Biology*, **51**(1), 39–54. doi:10.1016/s0092-8240(89)80047-3.

Baranowski R, Chen Y, Fryzlewicz P (2019). "Narrowest-Over-Threshold Detection of Multiple Change-Points and Change-Point-like Features." *Journal of the Royal Statistical Society B*, **81**, 649–672. doi:10.1111/rssb.12322.

Baranowski R, Chen Y, Fryzlewicz P (2023). ***not**: Narrowest-Over-Threshold Change-Point Detection*. R package version 1.5, URL https://CRAN.R-project.org/package=not.

Baudry JP, Maugis C, Michel B (2012). "Slope Heuristics: Overview and Implementation." *Statistics and Computing*, **22**(2), 455–470. doi:10.1007/s11222-011-9236-1.

Callies J, Ferrari R (2013). "Interpreting Energy and Tracer Spectra of Upper-Ocean Turbulence in the Submesoscale Range (1–200 km)." *Journal of Physical Oceanography*, **43**(11), 2456–2474. doi:10.1175/jpo-d-13-063.1.

Cho H, Fryzlewicz P (2024). "Multiple Change Point Detection under Serial Dependence: Wild Contrast Maximisation and Gappy Schwarz Algorithm." *Journal of Time Series Analysis*, **45**(3), 479–494. doi:10.1111/jtsa.12722.

Erdman C, Emerson JW (2008). "**bcp**: An R Package for Performing a Bayesian Analysis of Change Point Problems." *Journal of Statistical Software*, **23**, 1–13. doi:10.18637/jss.v023.i03.

Fearnhead P, Liu Z (2011). "Efficient Bayesian Analysis of Multiple Changepoint Models with Dependence across Segments." *Statistics and Computing*, **21**, 217–229. doi:10.1007/s11222-009-9163-6.

Fearnhead P, Maidstone R, Letchford A (2019). "Detecting Changes in Slope with an L_0 Penalty." *Journal of Computational and Graphical Statistics*, **28**(2), 265–275. doi:10.1080/10618600.2018.1512868.

Fearnhead P, Rigaill G (2020). "Relating and Comparing Methods for Detecting Changes in Mean." *Stat*, **9**(1), e291. doi:10.1002/sta4.291.

Frick K, Munk A, Sieling H (2014). "Multiscale Change Point Inference." *Journal of the Royal Statistical Society B*, **76**(3), 495–580. doi:10.1111/rssb.12047.

Fryzlewicz P (2014). "Wild Binary Segmentation for Multiple Change-Point Detection." *The Annals of Statistics*, **42**(6), 2243–2281. doi:10.1214/14-aos1245.

Fryzlewicz P (2020). "Detecting Possibly Frequent Change-Points: Wild Binary Segmentation 2 and Steepest-Drop Model Selection." *Journal of the Korean Statistical Society*, **49**(4), 1027–1070. doi:10.1007/s42952-020-00060-x.

Grose D, Fearnhead P (2024). ***cpop**: Detection of Multiple Changes in Slope in Univariate Time-Series*. R package version 1.0.7, URL https://CRAN.R-project.org/package=cpop.

Grundy T, Killick R, Mihaylov G (2020). "High-Dimensional Changepoint Detection via a Geometrically Inspired Mapping." *Statistics and Computing*, **30**(4), 1155–1166. doi:10.1007/s11222-020-09940-y.

Haynes K, Eckley IA, Fearnhead P (2017a). "Computationally Efficient Changepoint Detection for a Range of Penalties." *Journal of Computational and Graphical Statistics*, **26**(1), 134–143. doi:10.1080/10618600.2015.1116445.

Haynes K, Fearnhead P, Eckley IA (2017b). "A Computationally Efficient Nonparametric Approach for Changepoint Detection." *Statistics and Computing*, **27**(5), 1293–1305. doi:10.1007/s11222-016-9687-5.

Jackson B, Scargle JD, Barnes D, Arabhi S, Alt A, Gioumousis P, Gwin E, Sangtrakulcharoen P, Tan L, Tsai TT (2005). "An Algorithm for Optimal Partitioning of Data on an Interval." *IEEE Signal Processing Letters*, **12**(2), 105–108. doi:10.1109/lsp.2001.838216.

James NA, Matteson DS (2015). "**ecp**: An R Package for Nonparametric Multiple Change Point Analysis of Multivariate Data." *Journal of Statistical Software*, **62**(7), 1–25. doi:10.18637/jss.v062.i07.

Jewell SW, Hocking TD, Fearnhead P, Witten DM (2020). "Fast Nonconvex Deconvolution of Calcium Imaging Data." *Biostatistics*, **21**(4), 709–726. doi:10.1093/biostatistics/kxy083.

Killick R, Eckley I (2014). "**changepoint**: An R Package for Changepoint Analysis." *Journal of Statistical Software*, **58**(3), 1–19. doi:10.18637/jss.v058.i03.

Killick R, Fearnhead P, Eckley IA (2012). "Optimal Detection of Changepoints with a Linear Computational Cost." *Journal of the American Statistical Association*, **107**(500), 1590–1598. doi:10.1080/01621459.2012.737745.

Kim SJ, Koh K, Boyd S, Gorinevsky D (2009). "ℓ_1 Trend Filtering." *SIAM Review*, **51**(2), 339–360. doi:10.1137/070690274.

Kovács S, Li H, Bühlmann P, Munk A (2023). "Seeded Binary Segmentation: A General Methodology for Fast and Optimal Change Point Detection." *Biometrika*, **110**(1), 249–256. doi:10.1093/biomet/asac052.

Lebarbier É (2005). "Detecting Multiple Change-Points in the Mean of Gaussian Process by Model Selection." *Signal Processing*, **85**(4), 717–736. doi:10.1016/j.sigpro.2004.11.012.

Li H, Munk A, Sieling H (2016). "FDR-Control in Multiscale Change-Point Segmentation." *Electronic Journal of Statistics*, **10**(1), 918–959. doi:10.1214/16-ejs1131.

Maidstone R, Hocking T, Rigaill G, Fearnhead P (2017). "On Optimal Multiple Changepoint Algorithms for Large Data." *Statistics and Computing*, **27**(2), 519–533. doi:10.1007/s11222-016-9636-3.

Matteson DS, James NA (2014). "A Nonparametric Approach for Multiple Change Point Analysis of Multivariate Data." *Journal of the American Statistical Association*, **109**(505), 334–345. doi:10.1080/01621459.2013.849605.

Meier A, Kirch C, Cho H (2021). "**mosum**: A Package for Moving Sums in Change-Point Analysis." *Journal of Statistical Software*, **97**(8), 1–42. doi:10.18637/jss.v097.i08.

Niu YS, Zhang H (2012). "The Screening and Ranking Algorithm to Detect DNA Copy Number Variations." *The Annals of Applied Statistics*, **6**(3), 1306–1326. doi:10.1214/12-aoas539.

Pein F, Hotz T, Sieling H (2023). ***stepR**: Multiscale Change-Point Inference*. R package version 2.1-9, URL https://CRAN.R-project.org/package=stepR.

Pein F, Sieling H, Munk A (2017). "Heterogeneous Change Point Inference." *Journal of the Royal Statistical Society B*, **79**(4), 1207–1227. doi:10.1111/rssb.12202.

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

Reeves J, Chen J, Wang XL, Lund R, Lu QQ (2007). "A Review and Comparison of Changepoint Detection Techniques for Climate Data." *Journal of Applied Meteorology and Climatology*, **46**(6), 900–915. doi:10.1175/jam2493.1.

Richards KJ, Whitt DB, Brett G (2020). *Climate Change Impact on Submesoscale ROMS Data*. doi:10.5281/zenodo.4615129.

Richards KJ, Whitt DB, Brett G, Bryan FO, Feloy K, Long MC (2021). "The Impact of Climate Change on Ocean Submesoscale Activity." *Journal of Geophysical Research: Oceans*, **126**(5), e2020JC016750. doi:10.1029/2020jc016750.

Romano G, Eckley IA, Fearnhead P, Rigaill G (2023). "Fast Online Changepoint Detection via Functional Pruning CUSUM Statistics." *Journal of Machine Learning Research*, **24**(81), 1–36. URL http://jmlr.org/papers/v24/21-1230.html.

Ross GJ (2015). "Parametric and Nonparametric Sequential Change Detection in R: The **cpm** Package." *Journal of Statistical Software*, **66**(3), 1–20. doi:10.18637/jss.v066.i03.

Runge V, Hocking TD, Romano G, Afghah F, Fearnhead P, Rigaill G (2023). "**gfpop**: An R Package for Univariate Graph-Constrained Change-Point Detection." *Journal of Statistical Software*, **106**(6), 1–39. doi:10.18637/jss.v106.i06.

Scott AJ, Knott M (1974). "A Cluster Analysis Method for Grouping Means in the Analysis of Variance." *Biometrics*, **30**(3), 507–512. doi:10.2307/2529204.

Shi X, Gallagher C, Lund R, Killick R (2022). "A Comparison of Single and Multiple Changepoint Techniques for Time Series Data." *Computational Statistics & Data Analysis*, **170**, 107433. doi:10.1016/j.csda.2022.107433.

Soufflet Y, Marchesiello P, Lemarié F, Jouanno J, Capet X, Debreu L, Benshila R (2016). "On Effective Resolution in Ocean Models." *Ocean Modelling*, **98**, 36–50. doi:10.1016/j.ocemod.2015.12.004.

Tibshirani RJ (2014). "Adaptive Piecewise Polynomial Estimation via Trend Filtering." *The Annals of Statistics*, **42**(1), 285–323. doi:10.1214/13-aos1189.

Truong C, Oudre L, Vayatis N (2018). "**ruptures**: Change Point Detection in Python." *arXiv 1801.00826*, arXiv.org E-Print Archive. doi:10.48550/arxiv.1801.00826.

Truong C, Oudre L, Vayatis N (2020). "Selective Review of Offline Change Point Detection Methods." *Signal Processing*, **167**, 107299. doi:10.1016/j.sigpro.2019.107299.

Wang T, Samworth RJ (2018). "High Dimensional Change Point Estimation via Sparse Projection." *Journal of the Royal Statistical Society B*, **80**(1), 57–83. doi:10.1111/rssb.12243.

Xu H, Padilla O, Wang D, Li M (2022). ***changepoints**: A Collection of Change-Point Detection Methods*. R package version 1.1.0, URL https://CRAN.R-project.org/package=changepoints.

Yu Y, Chatterjee S, Xu H (2022). "Localising Change Points in Piecewise Polynomials of General Degrees." *Electronic Journal of Statistics*, **16**(1), 1855–1890. doi:10.1214/21-ejs1963.

A. Details of the recursion

Here we describe how to calculate the inner minimization

$$\min_{\alpha'} \left[F_k(\alpha') + \mathcal{C}_{l-1,l}(\alpha', \alpha) + \beta \right]$$

in the dynamic programming recursion – and how to do this so that the computational cost does not increase with the number of observations since the putative most recent changepoint.

The function $F_k(\alpha')$ will be defined as the minimum of a set of quadratics. Denote this $q_i^{(k)}(\alpha')$ for $i = 1, \ldots, M_k$. Then we wish to solve

$$\min_{\alpha'} \left[\min_{i \in 1:M_k} \left\{ q_i^{(k)}(\alpha') + \mathcal{C}_{k,l}(\alpha', \alpha) \right\} + \beta \right] = \min_{i \in 1:M_k} \left\{ \min_{\alpha'} \left[q_i^{(k)}(\alpha') + \mathcal{C}_{k,l}(\alpha', \alpha) + \beta \right] \right\}.$$

So we only need to be able to calculate $\min_{\alpha'}[q(\alpha') + \mathcal{C}_{l-1,l}(\alpha', \alpha)]$, for any known quadratic $q(\alpha')$. In the following, we will denote the co-coefficients of $q(\alpha')$ by a, b and c, so

$$q(\alpha') = a + b\alpha' + c\alpha'^2.$$

Our approach will be to (i) calculate the co-coefficients of $\mathcal{C}_{k,l}(\alpha', \alpha)$ in constant time, through the use of summary statistics; (ii) calculate the co-coefficients of the sum $q_i^{(k)}(\alpha') + \mathcal{C}_{k,l}(\alpha', \alpha) + \beta$; (iii) calculate the co-coefficients of the quadratic in α after we minimize with respect to α'. Steps (ii) and (iii) are trivial, but we give details below for completeness.

To simplify the exposition in the following we will use the convention that expressions that are of the form $0/0$ are equal to 0.

Define the following summary statistics for the data. These can be calculated prior to solving the dynamic programming recursion and enable the simple and quick calculation of $\mathcal{C}_{k,l}(\alpha', \alpha)$. These summary statistics are defined relative to the grid points – so a summary statistic with sub-script k will be based on all the data points for which $x_i \leq g_k$, and we define n_k to be the largest of observation such that $x_{n_k} \leq g_k$.

$$S_k^{(Y)} = \sum_{i=1}^{n_k} \frac{y_i}{\sigma_i^2}, \quad S_k^{(YY)} = \sum_{i=1}^{n_k} \frac{y_i^2}{\sigma_i^2}, \quad S_k = \sum_{i=1}^{n_k} \frac{1}{\sigma_i^2}$$

$$S_k^{(X)} = \sum_{i=1}^{n_k} \frac{x_i}{\sigma_i^2}, \quad S_k^{(XX)} = \sum_{i=1}^{n_k} \frac{x_i^2}{\sigma_i^2}, \quad S_k^{(XY)} = \sum_{i=1}^{n_k} \frac{x_i y_i}{\sigma_i^2}.$$

All summary statistics with sub-script 0, or that involve an empty sum, such that $n_k = 0$, are defined to be 0.

If we then define the co-coefficients of $\mathcal{C}_{k,l}(\alpha', \alpha)$, so that

$$\mathcal{C}_{k,l}(\alpha', \alpha) = A\alpha^2 + B\alpha\alpha' + C\alpha + D + E\alpha' + F\alpha'^2,$$

then tedious algebra gives that these coefficients are defined in terms of the summary statistics as

$$A = \frac{S_k^{(XX)} - S_l^{(XX)}}{(g_k - g_l)^2} - 2g_l \frac{S_k^{(X)} - S_l^{(X)}}{(g_k - g_l)^2} + g_l^2 \frac{S_k - S_l}{(g_k - g_l)^2},$$

$$
\begin{aligned}
B &= 2(g_k + g_l)\frac{S_k^{(X)} - S_l^{(X)}}{(g_k - g_l)^2} - 2\frac{S_k^{(XX)} - S_l^{(XX)}}{(g_k - g_l)^2} - 2g_k g_l \frac{S_k - S_l}{(g_k - g_l)^2}, \\
C &= 2g_l \frac{S_k^{(Y)} - S_l^{(Y)}}{g_k - g_l} - 2\frac{S_k^{(XY)} - S_l^{(XY)}}{g_k - g_l}, \\
D &= S_k^{(YY)} - S_l^{(YY)}, \\
E &= 2\frac{S_k^{(XY)} - S_l^{(XY)}}{g_k - g_l} - 2g_k \frac{S_k^{(Y)} - S_l^{(Y)}}{g_k - g_l}, \\
F &= \frac{S_k^{(XX)} - S_l^{(XX)}}{(g_k - g_l)^2} - 2g_k \frac{S_k^{(X)} - S_l^{(X)}}{(g_k - g_l)^2} + g_k^2 \frac{S_k - S_l}{(g_k - g_l)^2}.
\end{aligned}
$$

Adding $q_i^{(k)}(\alpha') + \beta$ to $\mathcal{C}_{k,l}(\alpha', \alpha)$ just changes the coefficients of powers of α' – that is D increases by $a + \beta$, E increases by b and F increases by c. Minimizing the resulting quadratic with-respect to α' gives a quadratic of the form $a' + b'\alpha + c'\alpha^2$ where

$$
\begin{aligned}
a' &= D + a + \beta - \frac{(E + b)^2}{4(F + c)}, \\
b' &= C - \frac{(E + b)B}{2(F + c)}, \\
c' &= A - \frac{B^2}{4(F + c)}.
\end{aligned}
$$

Affiliation:

Paul Fearnhead, Daniel Grose
Department of Mathematics and Statistics
Lancaster University
Lancaster, LA1 4YF, United Kingdom
E-mail: p.fearnhead@lancaster.ac.uk, dan.grose@lancaster.ac.uk

Journal of Statistical Software https://www.jstatsoft.org/
published by the Foundation for Open Access Statistics https://www.foastat.org/
May 2024, Volume 109, Issue 7 *Submitted:* 2022-11-22
doi:10.18637/jss.v109.i07 *Accepted:* 2023-10-06