Efficient Analysis of Complex Changepoint Problems

Robert Maidstone

Submitted for the degree of Doctor of Philosophy at

Lancaster University.

November 2016

Abstract

Many time series experience abrupt changes in structure. Detecting where these changes in structure, or changepoints, occur is required for effective modelling of the data. In this thesis we explore the common approaches used for detecting changepoints. We focus in particular on techniques which can be formulated in terms of minimising a cost over segmentations and solved exactly using a class of dynamic programming algorithms. Often implementations of these dynamic programming methods have a computational cost which scales poorly with the length of the time series. Recently pruning ideas have been suggested that can speed up the dynamic programming algorithms, whilst still being guaranteed to be optimal.

In this thesis we extend these methods. First we develop two new algorithms for segmenting piecewise constant data: FPOP and SNIP. We evaluate them against other methods in the literature. We then move on to develop the method OPPL for detecting changes in data subject to fitting a continuous piecewise linear model. We evaluate it against similar methods. We finally extend the OPPL method to deal with penalties that depend on the segment length.

Acknowledgements

Firstly I would like to thank everyone involved with the Statistics and Operational Research Centre for Doctoral Training at Lancaster University, especially the leadership team Jon Tawn, Idris Eckley and Kevin Glazebrook for giving me the opportunity to be part of the unique, stimulating research environment which STOR-i provides. I am also very grateful for the financial support provided through STOR-i by the EPSRC, and hope they continue to fund similar projects in the future.

The work in Chapter 3 is taken from the paper Maidstone et al. (2016) and I would like to thank my collaborators on this work; Guillem Rigaill and Toby Hocking. Discussions on this paper first started during the workshop *Inference for Change-Point and Related Processes* at the Isaac Newton Institute for Mathematical Sciences in Cambridge, and I would like to thank them for their support and hospitality. Further insightful emails with Guillem and Toby (and a productive meeting with Guillem in Lancaster) proved valuable in the development of the paper and Guillem and Toby's coding and analysis of the FPOP algorithm was also invaluable.

Perhaps most importantly, I would like to thank my supervisors Adam Letchford and Paul Fearnhead for all the time and effort they have put into this PhD project. Not only have they taught me a lot about the technical and statistical methods discussed in this thesis, but I feel that I have also learnt valuable research and life skills under their guidance.

The students of STOR-i over the years I have been part of the programme also deserve a mention as they have all impacted my life positively. In particular my intake year Emma, Ben, Dave, Judd, Tom, Hugo, Jeddy and Rhian have provided much friendship and banter as well as the occasional in-depth discussion. Lastly I would like to thank Kaylea for putting up with me and doing a lot of proof reading.

Declaration

I declare that the work in this thesis has been done by myself and has not been submitted elsewhere for the award of any other degree.

Robert Maidstone

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Analysis of time series data is required in numerous applications including finance (Fryzlewicz, 2014), medical statistics (Lavielle, 2005), climatology (Killick et al., 2010) and technology (Lung-Yut-Fong et al., 2012). As data collection and storage methods improve, these data sets are getting increasingly bigger. This has led to the idea of *Big Data* (Bühlmann et al., 2016). Big Data refers to data sets that are so huge that traditional statistical and analytical techniques cannot be applied, due to the exorbitant computational resources (time and/or memory) that would be needed. To analyse such data sets, new models and/or algorithms are required.

One area which requires algorithms to deal with Big Data is changepoint detection in time series. Changepoint models are often fitted to time series to deal with scenarios where the data experiences a sudden change in structure, such as that shown in Figure 1.1. These changes in structure, known as changepoints or breakpoints need to be taken into account if the data is to be modelled effectively.

The structure of this thesis is as follows. In Chapter 2 we will discuss some of the large body of literature on changepoint detection including; detection of a single change, multiple changepoint detection, Bayesian methods, nonparametric methods, multivariate methods and methods for more complicated data structures.

Then in Chapter 3 we will extend the discussion to pruned dynamic programming methods and develop two methods of our own; Functional Pruning Optimal Partitioning (FPOP) and Segment Neighbourhood with Inequality Pruning (SNIP). We evaluate the algorithms against competing methods and apply the FPOP algorithm to detecting changes in DNA copy number data in tumour microarray data.

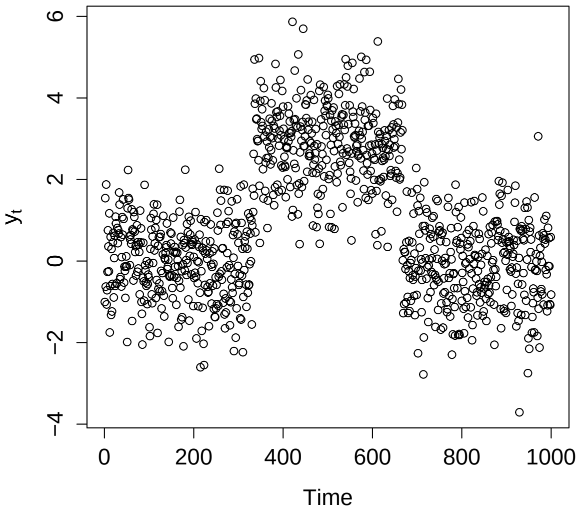


Figure 1.1: Example of data with sudden changes in structure. In this case a change in mean.

Chapter 4 considers the problem of fitting a piecewise linear model to the data where at the changepoints (places where the trend changes) we enforce continuity. Such a model is often of interest, for example when considering changes in velocity. We develop a method, Optimal Partitioning for Piecewise Linear data (OPPL), based on a dynamic programming algorithm with pruning steps added to it. We evaluate the method in terms of both efficiency and accuracy. We then apply the method to detecting changes in the angular velocity of a biological motor.

Chapter 5 offers an extension to the OPPL algorithm, adapting it to work with a penalty function which depends on the lengths of the fitted segments. Again this method is evaluated in terms of the efficiency and the accuracy and compared to using the standard OPPL method with the Bayesian Information Criterion (BIC) penalty value, which depends only on the length of the data set.

Finally, in Chapter 6, we conclude with a discussion of the main contributions of the thesis and potential paths to extend the methods developed in the future.

Chapter 2

Literature Review

The literature on changepoint detection is large and in this chapter we will discuss some of the key contributing papers. For a general overview of changepoint methods we refer the reader to the books Carlstein et al. (1994) and Chen and Gupta (2011), the book chapter Eckley et al. (2011) and the review papers Braun and Müller (1998) and Reeves et al. (2007).

We assume that we have data ordered by time, or some other attribute such as position along a chromosome. We denote this data by $\mathbf{y} = (y_1, \ldots, y_n)$. We are then interested in detecting the existence of changes in the structure of the data. We denote the number of changes as m and the positions of the changes by $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_m)$, labelled such that $\tau_1 < \tau_2 < \ldots \tau_m$. Often this is extended to let $\tau_0 = 0$ and $\tau_{m+1} = n$. In this section we discuss techniques for detecting changes in such data.

Often changepoint detection can be thought of in terms of two concepts. First we define some criterion on the data for model selection. This could be a cost to be minimised, such as the negative log likelihood, or a test statistic, such as the CUSUM. Secondly we require a search algorithm to find the changepoint (and segment parameter) estimates that satisfy the criterion.

In this literature review we will consider examples of both. First in Section 2.1 we consider detecting a single changepoint. These methods often do not require a search algorithm as checking every single changepoint location individually is usually feasible. In Section 2.2 we move on to consider multiple changepoint detection. In these methods checking all solutions is infeasible for large data sets and hence novel search algorithms are required.

2.1 Detecting a Single Changepoint

The detection of single changepoints can be useful for a number of reasons. For example, even when multiple changepoints exist, proving the existence of a single change in the structure can be enough to highlight the flaws in current models which assume no such changes exist.

The problem of detecting a single changepoint is fairly straight forward, and comes down to a model selection problem; whether to choose the model with $m = 0$ or the model with $m = 1$ with a changepoint at some location, τ. Four types of methods will be discussed here; the CUSUM test statistic, a likelihood-ratio based approach, penalised likelihood methods and Bayesian methods.

2.1.1 CUSUM

A simple method for single changepoint detection is to use a test statistic based on the cumulative sums of the data. This test statistic, known as the CUSUM test statistic, was first considered for changepoint detection problems in Page (1954) where a change in mean was considered. The CUSUM for a change in mean of an i.i.d. time series, $y_1, \dots, y_n$, is then defined as

$$U_n(\lfloor nr \rfloor) = n^{-\frac{1}{2}} \sum_{i=1}^{\lfloor nr \rfloor} (y_i - \bar{y}_n),$$

where $r \in [0, 1]$ and $\bar{y}_n$ is the sample mean of the sequence. The idea is if a changepoint occurs then the CUSUM will get large, if no changepoint occurs then the values either side of the mean will cancel each other out and the CUSUM test statistic will remain small. U_n can be shown to converge weakly to a standard Brownian bridge and this convergence result can be used to provide a suitable threshold value to test against.

Similar results can be shown for more general cases and as such the CUSUM provides a simple test for changepoint detection. Changes in regression are considered in Brown et al. (1975), where they also consider a related test statistic based on the sum of the squares of the data. Extensions of the CUSUM to changes in variance were considered in Inclan and Tiao (1994). Detecting a change in an autoregressive models is also considered by a number of authors including Lee et al. (2003) for an AR(1) process, Zhou and Liu (2009) for a change in mean in an infinite variance AR(p) process and Bai (1994) for detecting changes in ARMA models.

2.1.2 Likelihood-Ratio Based Approach

When it comes to model selection between two competing methods, one solution which instantly comes to mind is the likelihood-ratio test, which was first proposed for changepoint models by Hinkley (1970). This is used to compare the fit of two models where one (the null) is nested inside the other (the alternative). This is clearly the case in the situation of detecting a single changepoint, as the model where there is no changepoint is just a special case (where the two segment models are identical) of the model with a single changepoint. Hence we can define the null and alternative hypotheses as follows:

$$H_0 : \text{ No changepoint, } m = 0,$$

$$H_1 : \text{ A single changepoint, } m = 1.$$

H_1 then corresponds to a model with a single changepoint at $\tau \in (2, 3, \ldots, n-1)$.

Assume that a single model with density $p(\mathbf{y}_{1:n}|\boldsymbol{\theta})$ is fitted to the null model and that the two models $p_1(\mathbf{y}_{1:\tau}|\boldsymbol{\theta}_1)$ and $p_2(\mathbf{y}_{\tau:n}|\boldsymbol{\theta}_2)$ are fitted to the alternative. Then to use the likelihood-ratio test the likelihood-ratio test statistic, or equivalently the log-likelihood-ratio test statistic, needs to be calculated. The log-likelihood-ratio test is calculated as follows;

$$\begin{aligned} W(\tau) &= -2 \log \frac{L_{H_0}(\hat{\boldsymbol{\theta}})}{L_{H_1}(\hat{\boldsymbol{\theta}}_1, \hat{\boldsymbol{\theta}}_2)}, \\ &= -2 \log \frac{\prod_{i=1}^{n} p(y_i|\hat{\boldsymbol{\theta}})}{\prod_{i=1}^{\tau} p_1(y_i|\hat{\boldsymbol{\theta}}_1) \prod_{i=\tau}^{n} p_2(y_i|\hat{\boldsymbol{\theta}}_2)}, \\ &= 2 \left[\sum_{i=1}^{\tau} \log p_1(y_i|\hat{\boldsymbol{\theta}}_1) + \sum_{i=\tau+1}^{n} \log p_2(y_i|\hat{\boldsymbol{\theta}}_2) - \sum_{i=1}^{n} \log p(y_i|\hat{\boldsymbol{\theta}}) \right]. \end{aligned}$$

However this only works if τ is known, this is almost certainly not the case. To calculate the likelihood-ratio test statistic for unknown τ, we then need to take the maximum over all possible values of τ

$$W = \max_{\tau} W(\tau).$$

Then the null hypothesis (that there is no changepoint) is rejected in favour of the alternative (there is a single changepoint) if $W > c$ where c is some threshold value.

This threshold value is often taken to be c such that $\alpha = \Pr(W > c|H_0 \text{ is true})$ for some significance level α. The standard approach for the likelihood-ratio test is to assume that W is distributed with a $\chi^2_{d^*}$ distribution (where d^* is the difference in the number of parameters), this is known as Wilks' Theorem (Wilks, 1938). However this result does not hold for the changepoint problem as, due to the parameter τ being discrete, the two models are not considered "nested" under Wilks' assumptions.

Yao and Davis (1986) derive the null distribution for a changepoint model with a change in mean in Normal random variables as converging in distribution to the double exponential extreme value distribution. They also give a more precise finite-sample approximation. Using either of these, c can be chosen as the $(1-\alpha) \times 100\%$ quantile of the distribution.

Examples of the likelihood-ratio test statistic being used to detect single changepoints can be found in a number of papers. Notable examples include Hinkley (1970), where it is used on normally distributed data with a change in mean and Haccou et al. (1987) where the test is applied to exponentially distributed data with a change in the rate parameter (which affects both the mean and variance of the distribution).

2.1.3 Penalised Likelihood

Another way of detecting a single changepoint is to use the penalised likelihood. Fitting a penalised likelihood model is considered in the case of multiple changepoints, however the methodology can be applied to single changepoint detection. The likelihood is used as a measure of fit with an added penalty term to avoid overfitting. The likelihood and penalty term are combined into a single function which is then minimised. If we consider fitting a model $\mathcal{M}$, with p parameters. Denote the parameters by $\boldsymbol{\theta}$ and the likelihood by $L(\boldsymbol{\theta})$. The penalised likelihood is then defined as

$$PL(\mathcal{M}) = -2 \log \max L(\boldsymbol{\theta}) + p\beta,$$

where β is a penalty parameter which often depends on the data length n. The model $\mathcal{M}$ that minimises $PL(\mathcal{M})$ is then the preferred model.

For changepoint detection we search over all models with 0 or 1 changes, the total number of parameters is then either $p =$ "number of parameters for modelling as a single segment" or $p = 1+$ "number parameters for modelling as two segments" respectively.

The penalised likelihood is similar in theory to running a log-likelihood test as both methods will prefer a model with one changepoint if the increase in the log-likelihood is greater than a constant. For the penalised likelihood this constant is related to the penalty term, β. The method is highly dependent on the choice of this penalty term and the choice of this will be discussed further in Section 2.2.2.

2.1.4 Bayesian Methods

If, when considering changepoint models, prior information on the parameters is known (or assumed) then Bayesian techniques can be of use. Priors can be chosen for the individual segment parameters, $(\theta, \theta_1, \theta_2)$, the number of changepoints, $m \in \{0, 1\}$, and the locations of the potential changepoint, τ. Using these, posteriors can be calculated and the optimal model can be selected as the one with the largest posterior probability. A mathematical formulation of this is now given.

Let $p(\boldsymbol{\theta}|\boldsymbol{\psi})$ denote the prior on the parameters $\boldsymbol{\theta}$, where $\boldsymbol{\psi}$ are hyperparameters. Then the segment marginal likelihood can be defined by:

$$Q(s, t|\boldsymbol{\psi}) = \int p(\mathbf{y}_{s:t}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\psi}) \, \mathrm{d}\boldsymbol{\theta}. \tag{2.1}$$

Next assume that a prior probability for the existence of a changepoint is specified, that is $Pr(M = 1)$ is given. Then obviously the probability for there being no changepoints is also given, $Pr(M = 0) = 1 - Pr(M = 1)$. Also assign a prior to the changepoint location, τ, and denote this by $p(\tau)$. Then, for the case in which the $\boldsymbol{\psi}$ are known, the posterior probabilities can be found as follows:

$$\begin{aligned} &Pr(M = 0|\mathbf{y}_{1:n}) \propto Pr(M = 0) Q(1, n|\boldsymbol{\psi}), \\ &Pr(M = 1|\mathbf{y}_{1:n}) \propto Pr(M = 1) p(\tau) Q(1, \tau|\boldsymbol{\psi}) Q(\tau + 1, n|\boldsymbol{\psi}), \quad (\text{for } \tau = 1, \ldots, n). \end{aligned}$$

These values can be calculated in $\mathcal{O}(n)$ time, and they can be easily normalised to give the exact posterior probabilities.

To grade the decisiveness of the evidence in favour of selecting one model over another the *Bayes Factor* can be used. This is defined as the ratio of the posterior odds over the prior odds. Jeffreys (1961) gives a guide on how to use the Bayes Factor.

In some cases the integral in (2.1) is intractable and cannot be computed. One solution to this is to use MCMC to simulate from the marginal posterior distribution. Carlin et al. (1992) give an example of this where they use the Gibbs sampler to detect a single change in a Poisson process. They further apply their method to detecting a single change in the observations from a Markov chain and an simple linear regression.

2.2 Multiple Changepoint Methods

Single changepoint detection methods can often be extended to the multiple changepoint detection case, however the multiple changepoint case is much more computationally challenging. For a data set of length n, rather than having $n-1$ potential positions for a changepoints we have 2^{n-1} sets of possibilities to check if the number of changepoints is unbounded. Due to this added complexity novel search methods are required to efficiently compute solutions.

2.2.1 Binary Segmentation

Binary Segmentation, first introduced in Scott and Knott (1974) (and first used in a stochastic setting in Vostrikova (1981)), is one of the most widely used changepoint detection methods. This is due to its simplicity, fast runtime and relatively good accuracy.

Binary Segmentation takes a recursive approach. The idea is that if a single changepoint can be detected in a time series then the series can be split around this changepoint into two sub-series either side of the change. Then the single changepoint detection method can be applied to each of the sub-series. This can then be iterated until no further changepoints are detected.

Often this is achieved via the calculation of a test statistic, $\Gamma(\mathbf{y}_{1:n})$, that is dependent on the data. This test statistic can be computed over the full data set and then compared to a threshold value, c, to determine if a change occurs. If it does then the change is calculated via an estimator $\hat{\tau}(\mathbf{y}_{1:n})$ and stored. The data is then split around this detected changepoint and the process is repeated on the two subsets of the data $\mathbf{y}_{1:\hat{\tau}(\mathbf{y}_{1:n})}$ and $\mathbf{y}_{\hat{\tau}(\mathbf{y}_{1:n})+1:n}$. This process is repeated until the threshold is not breached by any subset of the data. The stored changepoints are then returned.

An obvious choice for the test statistic is to use the likelihood-ratio test (from Section 2.1.2). With the null hypothesis corresponding to the model with no changepoint and the alternative with a change at time $\tau \in (1, 2, \ldots, n-1)$ then we can define a test statistic Γ on the segment from s to t as follows

$$\Gamma(\mathbf{y}_{s:t}) = 2\left[\max_{\tau}\left(\sum_{i=s}^{\tau}\log p_1(y_i|\hat{\boldsymbol{\theta}}_1) + \sum_{i=\tau+1}^{t}\log p_2(y_i|\hat{\boldsymbol{\theta}}_2)\right) - \sum_{i=s}^{t}\log p(y_i|\hat{\boldsymbol{\theta}})\right].$$

The null hypothesis is rejected in favour of the alternative (i.e. we detect a changepoint) if $\Gamma(\mathbf{y}_{s:t}) > c$ for some threshold value c to be chosen. Often this value is chosen to be such that $\alpha = Pr(\Gamma(\mathbf{y}_{s:t}) > c|H_0 \text{ is true})$ for some significance level α. The changepoint estimator would then be the value of τ that maximises the test statistic. One issue with using the likelihood-ratio test in this way is that at each stage of the algorithm we test for the existence of a changepoint by comparing against the null hypothesis that no such change exists, even though we may already know of the existence of a change. Hyun et al. (2016) discusses the issues with this naive approach further and offers an alternative test statistic that retrospectively conditions on detected changepoints.

Binary Segmentation can also be used with a cost function (often taken as negative the log-likelihood) to be minimised and gives a "greedy" like heuristic where the changepoint is picked as the value which gives the greatest reduction to the cost at each step. The threshold value, c, then becomes equivalent to the penalty value in a penalised cost function approach (see Section 2.1.3).

Algorithm 1: Binary Segmentation

```
Input  : Set of data of the form $\mathbf{y}_{1:n} = (y_1, \ldots, y_n)$,
         A test statistic dependent on the data, $\Gamma(\cdot)$,
         An estimator of changepoint location $\hat{\tau}(\cdot)$,
         A rejection threshold $c$.
Let $\mathcal{CP} = \emptyset$ and $\mathcal{S} = \{[1, n]\}$;
while $\mathcal{S} \neq \emptyset$ do
    Choose an element of $\mathcal{S}$; denote this element as $[s, t]$;
    if $\Gamma(\mathbf{y}_{s:t}) < c$ then
        remove $[s, t]$ from $\mathcal{S}$
    if $\Gamma(\mathbf{y}_{s:t}) \geq c$ then
        remove $[s, t]$ from $\mathcal{S}$;
        calculate $r = \hat{\tau}(\mathbf{y}_{s:t}) + s - 1$, and add $r$ to $\mathcal{CP}$;
        if $r \neq s$ add $[s, r]$ to $\mathcal{S}$;
        if $r \neq t - 1$ add $[r + 1, t]$ to $\mathcal{S}$;
Output: The set of changepoint locations, $\mathcal{CP}$.
```

Binary Segmentation can also be modified to output a user specified number of changes. Rather than comparing against a threshold value, all changes are accepted until the total number exceeds a given value.

For detecting changes in the mean of normal random variables, Venkatraman (1992) give a proof of the consistency of Binary Segmentation for both the number of identified changepoints and the changepoint locations. However, they show that this has sub-optimal rates if m is allowed to increase with the data length. Chen et al. (2011) prove similar results for a fixed number of changepoints. Fryzlewicz (2014) further show that Binary Segmentation is only consistent when the minimum spacing between any two adjacent changes is of order greater than $\mathcal{O}(n^{\frac{3}{4}})$.

Binary Segmentation is an efficient algorithm, running in $\mathcal{O}(n \log n)$. Its efficiency and simplicity means that many adaptations of it exist, manipulating the algorithm to give better accuracy (Olshen et al., 2004; Fryzlewicz, 2014; Kirch and Muhsal, 2014) or to deal with more complex models (Fryzlewicz and Subba Rao, 2014). However, these adaptations often come at the cost of an increased computational time. Below we discuss two of the most promising variations of Binary Segmentation; Circular Binary Segmentation and Wild Binary Segmentation.

Circular Binary Segmentation (CBS)

One issue with Binary Segmentation is that it cannot detect small segments which are buried within larger segments, this is largely due to the fact that BS searches for a single changepoint. Olshen et al. (2004) propose the Circular Binary Segmentation (CBS) method to deal with this, specifically for segmenting DNA copy number data.

Circular Binary Segmentation uses both the likelihood-ratio test for detecting a single change and an adapted version of this test from Levin and Kline (1985) which tests for epidemic changes (adding two changepoints where the first changes the mean to a new value and the second returns to the original mean). At each step in the algorithm they consider the segment being tested upon as having the two ends joined to form a circle; this feature gives the algorithm its name.

Olshen et al. (2004) show that Circular Binary Segmentation performs better than Binary Segmentation in certain scenarios where a narrow segment is buried within a larger one. Willenbrock and Fridlyand (2005) and Lai et al. (2005) both compare CBS against other methods for detecting changes in CGH data and show that CBS performs well on this data, however Lai et al. (2005) also finds that CBS is one of the slowest methods that they tested. This is due to computational cost of the nonparametric method they use to calculate the p-value, resulting in the algorithm growing quadratically with the data length. With this in mind, a faster CBS algorithm is developed in Venkatraman and Olshen (2007) which uses a hybrid approach to calculate the p-value of the test statistic using a Gaussian random field and also adds a stopping rule which limits the number of iterations of the algorithm when there is strong evidence of a change. Theys show that this improves the efficiency of the method with only a negligible loss in accuracy.

Wild Binary Segmentation (WBS)

Another adaptation of Binary Segmentation that has recently been proposed is Wild Binary Segmentation (WBS), first introduced by Fryzlewicz (2014). Wild Binary Segmentation adds a random element to the Binary Segmentation algorithm in a hope to improve the accuracy.

As mentioned previously Binary Segmentation can struggle when detecting more than one changepoint, this is due to the fact that at each step in the algorithm it fits a single changepoint model to a segment which may contain more then one change. To get around this WBS runs their test on multiple intervals with start and end points which are drawn (independently with replacement) uniformly from the set $\{s, \ldots, e\}$ (where s and e are the start and end points respectively of the current segment). The test statistics are then weighted according to the length of the interval and the greatest is tested against the threshold value.

The intuition behind WBS is that if a change exists in a segment then the hope is that one of the drawn intervals will have just this change and no other within it and the change will be detected. Smaller segments are more likely to only contain one changepoint, but have smaller statistical power than longer segments hence the need for both.

One potential issue with WBS is that in addition to choosing the threshold value (c in Algorithm 1) the number of intervals to draw at each iteration also needs to be chosen. This number will have an impact on both the accuracy (as drawing more intervals should lead to a better chance of getting a "good" one) and the efficiency (the more intervals the test statistic is calculated on the slower the method). Fryzlewicz (2014) discuss how to choose both of these to obtain the best results.

Fryzlewicz (2014) compare the WBS method against both Binary Segmentation and PELT (Killick et al. (2012), introduced fully in Section 3.3.2) on simulated data. They simulate the number of changepoints in the data from a Poisson distribution and the jumps at the changepoints from a zero mean Normal distribution, to this true signal they then add Gaussian noise (with variance 1). They compare the methods by computing the distance between the true number of changepoints and the number of changepoints estimated by the methods. For this study they show an increased performance of WBS over both Binary Segmentation and PELT, however for PELT this is down to poor penalty choice as we will show later (Section 3.7.4). Due to having to compute the test statistics for multiple intervals at each iteration of the algorithm the computational time of WBS will be greater than that of BS.

2.2.2 Dynamic Programming Based Methods

Many multiple changepoint detection methods aim to solve an optimisation problem, often minimising a cost function defined on the data. To solve such an optimisation problem exactly, dynamic programming can be used. Often this is used as the optimisation problem in question is computationally expensive to compute by complete enumeration and hence a more efficient algorithm is required. One example of such a problem can be obtained by extending the penalised likelihood method from Section 2.1.3 to multiple changepoint detection. By comparing the penalised likelihood for all possible models, the optimum can be chosen, however as there are 2^n such possible models (assuming that the segment parameters can be found analytically) this is infeasible for long data sets. Dynamic programming algorithms for penalised likelihood methods are given in Jackson et al. (2005) and Killick et al. (2012). Other criteria can have similar forms; for example maximum *a posteriori* (MAP) estimation in Bayesian statistics (which is discussed in Section 2.2.3) and Minimum Description Length (MDL, Rissanen (1989)).

One approach often used in dynamic programming based methods for changepoint detection is to fit a cost function to each segment and then minimise the sum over the segments. This cost function can be, for example, the negative log likelihood or the residual sum of squares. The minimisation can be formulated in one of two ways; by fixing the number of changepoints or by adding a penalty term.

If the number of changepoints is fixed then we obtain the following minimisation problem, for a given K,

$$\min_{\boldsymbol{\tau}} \left[\sum_{j=0}^{K} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) \right], \tag{2.2}$$

for a given cost function, $\mathcal{C}(\cdot)$. This is known as the *constrained minimisation problem.*

Conversely, we can minimise over the number of segments as well. If we do this then a penalty term will have to added with each new segment to avoid overfitting and ending up in the trivial case of fitting a changepoint at each time point. This leads to the following

$$\min_{k,\boldsymbol{\tau}} \left[\sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \beta \right] - \beta, \tag{2.3}$$

for a given cost function, $\mathcal{C}(\cdot)$ and a penalty value $\beta > 0$. This is known as the *penalised minimisation problem*. If the cost function used is the negative log-likelihood then the penalised minimisation problem is equivalent to the penalised likelihood method.

Both the constrained and penalised minimisation problem are computationally demanding to be solved exactly by complete enumeration, thus efficient algorithms have been developed. To solve the constrained minimisation problem Auger and Lawrence (1989) developed the Segment Neighbourhood Search (SNS) algorithm. For the penalised minimisation problem Jackson et al. (2005) developed the method Optimal Partitioning (OP). Both SNS and OP are computationally slow when compared to fast heuristics such as Binary Segmentation, with this in mind a couple of authors have developed methods to prune the search space of the algorithms for a more efficient algorithm. pDPA (Rigaill, 2015) is a pruned version of SNS whilst PELT (Killick et al., 2012) provides a pruned version of OP. SNS, OP, pDPA and PELT are all discussed in greater detail in Chapter 3 and two new pruned dynamic programming methods, FPOP and SNIP, are developed.

Choice of Penalty in the Penalised Minimisation Problem

Solving the penalised minimisation problem in (2.3) relies on the choice of penalty value β to stop the method overfitting the data. Choosing this penalty can significantly affect the accuracy of the methods such as Optimal Partitioning and PELT. In general this penalty can be difficult to choose, however there exists some approaches to find a good value.

Akaike's Information Criterion (AIC), from Akaike (1974), is one of the simplest forms of penalty term. This is due to the fact that the penalty term is constant with respect to the data length. In equation (2.3) for a cost function of twice the negative log likelihood it is usually taken as $\beta = 2$. Intuitively the AIC aims to pick the model which is closest to the unknown true model, however this often leads to it overfitting the data. An example of the AIC being used on Gaussian changepoint models with a change in mean can be seen in Ninomiya (2005).

The Bayesian Information Criterion (BIC), also called the Schwarz Information Criterion (SIC), was first introduced in Schwarz (1978). It uses a penalty term of $\beta = \log n$ (for a cost function of twice the log likelihood), as opposed to the AIC's constant penalty. Intuitively the BIC is an estimate of the posterior probability of a model being true, it is often preferred over the AIC however it can be susceptible to underfitting the data. Yao (1988) gives a framework for using the BIC to detect changes of mean in a sequence of data which is normally distributed with constant variance, they further go on to establish weak consistency for estimating the number and position of the changepoints.

Despite the widespread use of the AIC and BIC, other penalty functions can be used in their place. Lavielle (2005) gives a more generalised description of using a penalised likelihood function to detect changepoints as part of a dynamic programming method. However it should be noted that despite the general approach considered, the authors end up recommending either the use of the AIC or the BIC.

Changepoints for a Range of PenaltieS (CROPS)

An alternative approach is to solve equation (2.3) for a range of penalties and then choose the outputted segmentation that gives desired characteristics (whether that be close to the true number of changes or a low mean squared error) however for real data, where the true signal is unknown, choosing between these solutions can be difficult.

A recent paper by Haynes et al. (2015) looks at ways to do this efficiently. They concentrate primarily on using PELT, however their procedure can be applied to Optimal Partitioning and other methods that exactly minimise the penalised minimisation problem (including FPOP introduced in Chapter 3).

Firstly they describe a way of only running the method for specific values of β such that given a range for β every possible segmentation is outputted, this method is known as CROPS (Changepoints for a Range of PenaltieS). Given a range for the penalty parameter $\beta \in [\beta_{min}, \beta_{max}]$, it works by first running PELT on the each of the two ends of the interval to get two segmentations. If the number of changepoints in the two segments are the same then they are equal and no other possible segmentation exist over the interval. If the number of changepoints differ by one then the two segmentations are the only two that exist over the interval and, by extrapolating the costs as the penalty changes, the intersection can be found which is the value that provides the boundary from choosing one segmentation to the other. If the number of changepoints differs by more than one then the intersection can be found and then PELT is needed to be run on that value, the process is then repeated for the two new intervals created.

Further Haynes et al. (2015) show that given the range $(\beta_{min}, \beta_{max})$ the maximum number of times

PELT is run is $m(\beta_{min}) - m(\beta_{max}) + 2$ times, where $m(\beta)$ is the number of changepoints outputted from running PELT with penalty β. For PELT itself they also offer a way of reusing the calculations from previous runs to further increase the efficiency.

For an interval $[\beta_{min}, \beta_{max}]$, CROPS efficiently outputs all optimal segmentations for the penalised minimisation problem (2.3). A resulting segmentation with m changepoints is also the solution to the constrained minimisation problem (2.2), with $K = m$. However, some values of the total number of changepoints, m, will never be optimal regardless of the penalty value used and hence will not be outputted by CROPS.

CROPS gives a way of efficiently running PELT to output all segmentations over an interval, however the penalty choice problem remains. Lavielle (2005) offer a way to choose this once the segmentations are found. They suggest plotting the unpenalised cost against the number of changes found and then choosing segmentations that lie on the elbow of the plot, that is the point where the decrease in cost due to detecting further changes noticeably changes. An example of this is given in Figure 2.1, where the elbow is shown by the red point. This elbow point can be found by eye or Lavielle (2005) give a formula for calculating it exactly which relies on picking the greatest number of changes such that the gradient is less than a chosen threshold (Lavielle (2005) propose a threshold of $S = -0.75$ based on numerical experiments).

Another method for determining the penalty is given in Rigaill et al. (2013). Here training data is used to choose a good penalty. This can be used in conjuncture with CROPS to limit the amount of testing that is done on the training data to the segmentations outputted by the CROP algorithm, however it requires the training data to be correctly annotated with the true changepoints.

2.2.3 Bayesian Methods

Often Bayesian methods are used for multiple changepoint detection. They offer the opportunity to include prior knowledge of how the changes and the data within the segments are distributed into the model. Bayesian methods for changepoint detection have been used on data from a number of areas such as genomics (Fearnhead and Liu, 2007; Rigaill et al., 2012; Caron et al., 2012), coal mining (Green, 1995; Fearnhead, 2006), geology (Fearnhead, 2006) and speech segmentation (Fearnhead, 2005).

When considering Bayesian methods it seems natural to want to include priors on the two objects of

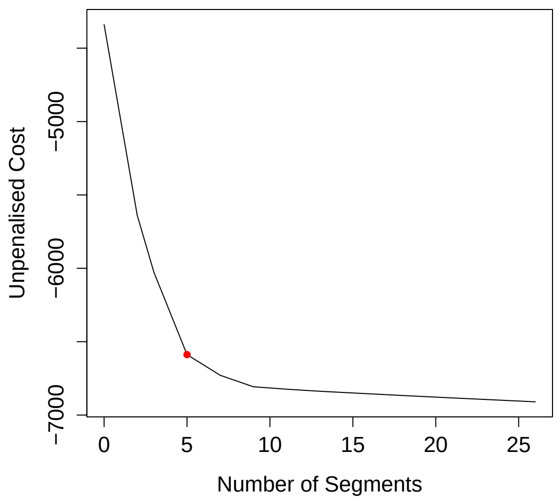


Figure 2.1: Plot of the unpenalised cost against the number of segments found, Lavielle (2005) suggests choosing the elbow of the plot (red point) as a good choice of segmentation.

interest; namely the number of changepoints m and the vector of positions of these changepoints, $\boldsymbol{\tau}$. This would be done by first introducing a prior $p(m)$ on the number of changepoints, and then introducing a prior $p(\boldsymbol{\tau}|m)$ on the locations given the number of changepoints.

However there is another alternative. The information given by m and $\boldsymbol{\tau}$ can also be stored by the segment lengths. Let $s_i = \tau_i - \tau_{i-1}$ be the length of the ith segment, then a prior can be given to s_i (for $i = 1, \ldots, m+1$). Not only does this have computational advantages over the first method, but it also doesn't have to be adapted dependent on the time period over which the considered time series is observed. With this in mind, we briefly outline this approach below.

Letting $f(\cdot|\boldsymbol{\psi})$ be the probability mass function for the length of a segment, and for notational purposes we also define the survivor function as $S(t|\boldsymbol{\psi}) = \sum_{i=t}^{\infty} f(i|\boldsymbol{\psi})$. Then the joint prior for m and $\boldsymbol{\tau}$ can be inferred as:

$$p(m, \boldsymbol{\tau}_{1:m}|\boldsymbol{\psi}) = \left(\prod_{i=1}^{m} f(s_i|\boldsymbol{\psi})\right) S(\tau_{m+1} - \tau_m|\boldsymbol{\psi}). \tag{2.4}$$

A common choice for the distribution of the segment lengths is the geometric, under this distribution the inferred distribution for the joint prior for m and $\boldsymbol{\tau}$ is:

$$p(m, \boldsymbol{\tau}_{1:m}) = \lambda^m (1-\lambda)^{n-m-1}, \tag{2.5}$$

where λ is the parameter of the geometric distribution.

The segment marginal likelihood (defined in equation (2.1)) can then be used to calculate the marginal posterior as was done in Section 2.1.4:

$$p(m, \boldsymbol{\tau}_{1:m}|\boldsymbol{\psi}, \mathbf{y}_{1:n}) = \left(\prod_{i=1}^{m} f(s_i|\boldsymbol{\psi})Q(\tau_{i-1}+1, \tau_i|\boldsymbol{\psi})\right) S(\tau_{m+1}-\tau_m|\boldsymbol{\psi})Q(\tau_m+1, \tau_{m+1}|\boldsymbol{\psi}). \quad (2.6)$$

As these are all known quantities (assuming that the segment marginal distributions can be calculated) then the marginal posterior can be simulated from using MCMC.

Markov Chain Monte Carlo

Using the segment marginal likelihood a MCMC algorithm can be used to determine the changepoint numbers, positions and the segment parameters by repeatedly drawing samples from approximations to the posteriors. Stephens (1994) achieve this by extending the methodology of Carlin et al. (1992) to the multiple changepoint model. They describe how an exact Bayesian solution can be obtained if the priors and likelihood function are conjugate, however this is not always the case. For more intractable posteriors they suggest using the Gibbs sampler where draws are taken from the full conditional distributions (conditional on all the remaining variables).

When the conditional distributions are not known or unable to be sampled from other methods are needed. One approach is to use a Metropolis-Hastings algorithm, where draws are taken from a proposal distribution which approximates the posterior and accepted with some probability related to the conditionals. Lavielle and Lebarbier (2001) give details of such an approach. A reversible jump Metropolis-Hastings algorithm is detailed in Green (1995) which allows simulation even if the number of parameters is not known (such as in the case of unknown number of changepoints). Lavielle and Lebarbier (2001) extend the Metropolis-Hasting by combining it with a Gibbs sampling step to provide a hybrid approach without the need for a reversible jump step. They show that this converges much faster than the reversible jump algorithm of Green (1995).

Chib (1998) discusses some of the problems with a Metropolis Hastings based algorithm; namely the choice of the proposal distribution and the computational complexity of the methods for long time series with many changes. They suggest an alternative MCMC approach where the changepoint model is parameterised as a hidden Markov model. MCMC techniques from the hidden Markov model literature can then be applied, for example applying the MCMC algorithm from Chib (1996). Hidden Markov Models are discussed in more detail later on in this section.

MCMC techniques in general are computationally expensive and also often come into difficulties when dealing with the convergence of the algorithms.

Direct Simulation from the Posterior

As mentioned above, MCMC techniques aim to get around the problem of being unable to compute the true posterior distribution in Bayesian changepoint models. One issue that remains is choosing a proposal that is sufficiently close to the posterior. Alternative methods in the literature offer a way of simulating directly from the true posterior, despite not being able to compute it. This avoids the problems associated with diagnosing the convergence that occur with MCMC.

Barry and Hartigan (1992) introduce product partition models (PPMs) which work under the assumption that the observations can be divided into independent adjacent segments and hence the likelihood conditioned on the segmentation can be written as

$$\begin{aligned} p(\mathbf{y}|m,\tau_1,\ldots,\tau_m) &= \prod_{i=0}^{m}\left[\int\left(\prod_{j=\tau_i+1}^{\tau_{i+1}} p(y_t|\theta_i)\right)p(\theta_i|\boldsymbol{\psi})\,\mathrm{d}\theta_i\right], \qquad (2.7)\\ &= \prod_{i=0}^{m} Q(\tau_i+1,\tau_{i+1}|\boldsymbol{\psi}) \end{aligned}$$

where θ_i are the parameters associated with the segment i and $p(\theta_i|\boldsymbol{\psi})$ is the prior on these parameters with hyperparameters $\boldsymbol{\psi}$. The function $Q(\tau_i+1,\tau_{i+1}|\boldsymbol{\psi})$ is the segment marginal likelihood, as defined in equation (2.1).

By integrating out θ_i and summing over τ_i for $i=1,\ldots,m$ (and θ_0) in equation (2.7) the marginal likelihood conditional on the number of changepoints can be obtained. The sum over τ_i for $i=1,\ldots,m$ is computationally challenging, however Rigaill et al. (2012) introduce a dynamic programming algorithm which performs this in $\mathcal{O}(mn^2)$. Rigaill et al. (2012) also derive exact formulas for other quantities of interest such as the posterior probability of a changepoint occurring at time t.

A similar dynamic programming approach is used in Fearnhead (2006), which is a generalised case of Liu and Lawrence (1999). However, whilst before the segmentation was chosen conditional on the number of changepoints, here the number of changepoints is allowed to vary. Firstly the probabilities of observing the data after a change conditional on the location of that change are computed. Then the changepoints can be recursively simulated from these probabilities. Using these simulations a Viterbi algorithm (Viterbi, 1967) can be used to find the MAP (maximum *a posteriori*) estimate of the changepoint positions. Fearnhead (2006) also show that the computational cost of the method is better than using an MCMC approach, being linear in the length of the data. Further work in Fearnhead and Liu

(2007) and Caron et al. (2012) extends this approach to provide exact inference in an online setting and in Fearnhead (2005) the method is extended to cater for fitting a linear regression model on the segments.

The Bayesian methods discussed here utilise dynamic programming algorithms to perform the integration. They generally have computational complexitity of the same order as their non-Bayesian counterparts, however with a much larger constant. Further, pruning techniques such as those seen in Rigaill (2015) and Killick et al. (2012) (and discussed further in Chapter 3) have not been developed for the Bayesian case.

Hidden Markov Models

A specific type of Bayesian model often used for changepoint problems is hidden Markov models (HMMs) and their application has been studied by many authors. For a general overview of hidden Markov models themselves we direct the reader to Cappé et al. (2009). For changepoint analysis with HMMs, Luong et al. (2004) give a good introduction.

A typical HMM is defined by the joint probability distribution

$$Pr(y_{1:n}, S_{1:n}) = Pr(S_1)Pr(y_1|S_1)\prod_{i=2}^{n} Pr(S_i|S_{i-1})Pr(y_i|S_i), \tag{2.8}$$

where $S_{1:n}$ is the vector of hidden variables.

In HMM terminology the observations $y_{1:n}$ are the *observed states* and the variables $S_{1:n}$ the *hidden states*. The general topology is then that the observed states depend on the hidden state at that time point and the hidden states depend on the hidden state at the previous time point. This is shown graphically in Figure 2.2 for $n = 5$. By choosing a prior on the states of the variables, we wish to compute the posterior probabilities of the hidden variables.

Luong et al. (2004) discuss two approaches to fitting HMMs to changepoint problems; firstly level-based where the hidden state S_i relates to the distribution of the observation y_i, and secondly segment-based where the hidden state S_i relates to the segment index of the observation y_i. The main difference between the two approaches is that level-based allows non-adjacent segments to be modelled with the same distribution whereas segment based would model them separately.

Many methods for inference in HMMs exist in the literature, some of which we have discussed before. Rabiner (1989) and Durbin et al. (1998) suggest using a forward-backward algorithm similar to dynamic

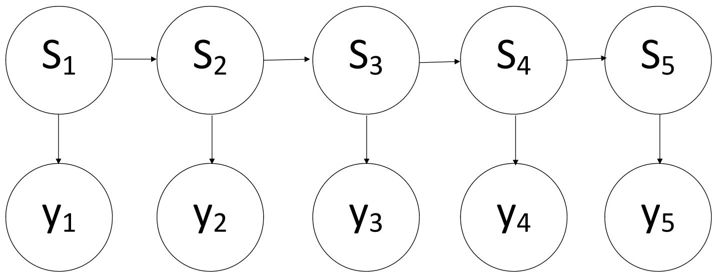


Figure 2.2: HMM topology with $n = 5$. S_i are the hidden states and y_i are the observed states, for $i = 1, \dots, 5$.

programming or MAP. Expectation-maximisation (E-M) algorithms have also been used, for example by Chamroukhi et al. (2011). Varying MCMC techniques have been used to fit HMMs, including Gibbs sampling (Chib, 1998), reversible jump MCMC (Rueda and Díaz-Uriarte, 2007) and sequential MCMC (Nam et al., 2012). Further, sampling via particle filters is used to fit a HMM in Fearnhead and Clifford (2003) and finite Markov chain embedding used in Aston et al. (2012).

2.2.4 Fused LASSO

The fused LASSO is another technique often used for changepoint detection. Originally proposed in the signal processing literature under the name "total variation denoising" in Rudin et al. (1992), it was reproposed as the fused LASSO in Tibshirani et al. (2005). Examples of the use of the fused LASSO include EEG data (Chan et al., 2013), genomics (Tibshirani et al., 2005; Hoefling, 2009) and climate data (Shen et al., 2014).

For data $\mathbf{y} = y_1, \dots, y_n$ the fused LASSO estimate is defined by

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^n}{\arg\min} \frac{1}{2} \sum_{i=1}^{n} (y_i - \theta_i)^2 + \lambda \sum_{i=1}^{n-1} |\theta_i - \theta_{i+1}|, \tag{2.9}$$

where $\lambda \geq 0$ is a tuning parameter to be chosen.

As can be seen from equation (2.9) the fused LASSO is similar in nature to the penalised minimisation problem defined in equation (2.3) with a residual sum of squares cost, but with a non-linear penalty term. The penalty term in equation (2.9) penalises large changes in the parameter value θ. This often leads to the method choosing to fit many small jumps rather than one large one which can be an issue.

Many papers develop the theory behind the fused LASSO further. For example Dalalyan et al. (2014) look at the l_2 error rates of the fused LASSO method and Harchaoui and Lévy-Leduc (2010) and Rojas and Wahlberg (2014) look at the changepoint recovery properties of the method. Extensions of the fused LASSO have also been developed. Trend Filtering (Kim et al., 2009; Tibshirani, 2014) extends the fused LASSO to deal with changes in trend, penalising second order (rather than first order) differences. We discuss Trend Filtering further in Chapter 4 where we compare it against our method OPPL. The graph fused LASSO (Tibshirani et al., 2005; Hoefling, 2009) extends the fused LASSO for the problem of changepoint detection on a graph.

2.2.5 Simultaneous Multiscale Change-Point Inference

In the Royal Statistical Society read paper Frick et al. (2014), the authors propose the Simultaneous Multiscale Changepoint Estimator (SMUCE) method. The SMUCE method detects an unknown number of changepoints in the parameter of a distribution from the one-dimensional exponential family. It involves minimising the number of changepoints over a set of underlying parameter functions that, for a multiscale test statistic, do not exceed a threshold. They then compute the maximum likelihood estimator, constrained on this meeting this criteria.

They let $\vartheta : [0,1) \to \Theta \subseteq \mathbb{R}$ be a right-continuous step function representing the changing parameters

$$\vartheta(t) = \sum_{k=0}^{K} \theta_k \mathbb{I}_{[\tau_k, \tau_{k+1})}(t). \tag{2.10}$$

Then a multiscale statistic can be defined for a function of this type, by maximising a test statistic over all possible segments of the data that do not cross one of the changepoints in the function $\vartheta(t)$.

$$T_n(Y, \vartheta) = \max_{\substack{1 \leq i < j \leq n \\ \vartheta(t)=\theta_0 \text{ for } t \in [i/n, j/n]}} \left(\sqrt{2T_i^j(Y, \theta_0)} - \sqrt{2 \log \left(\frac{n \exp(1)}{j - i + 1} \right)} \right). \tag{2.11}$$

Here T_i^j is the local likelihood-ratio statistic for testing the null hypothesis that θ_0 is the true parameter against the alternative that it is not on the (rescaled time) interval $[i/n, j/n]$. Formally this is defined as

$$\begin{aligned} T_i^j(Y, \theta_0) &= \log \left(\frac{\sup_{\theta \in \Theta} \prod_{l=i}^{j} f_\theta(Y_l)}{\prod_{l=i}^{j} f_{\theta_0}(Y_l)} \right), \\ &= \sup_{\theta \in \Theta} \left(\sum_{l=i}^{j} (\theta Y_l - \psi(\theta)) \right) - \sum_{l=i}^{j} (\theta_0 Y_l - \psi(\theta_0)). \end{aligned}$$

Hence, T_n evaluates the maximum of the local likelihood-ratio statistics on the discrete intervals $[i/n, j/n]$ on which ϑ is constant, with parameter value $\theta = \theta_{i,j}$. Note that the 'multiscale' description comes from the power of the method to consider segments of different lengths equally by normalising them using the second term in (2.11).

Then on the space, S, of all step functions defined by (2.10) the following minimisation problem is solved for some threshold value q

$$\inf_{\vartheta \in S} \#J(\vartheta) \quad \text{s.t.} \quad T_n(Y, \vartheta) \leq q \tag{2.12}$$

where $J(\vartheta)$ is the number of changepoints in the step function ϑ.

If we denote the number of changes in the solution of (2.12) for a given threshold q as $\hat{K}(q)$, then we can find the set of functions that have $\hat{K}(q)$ changes and whose multiscale statistic lies under the threshold

$$\mathcal{C}(q) = \{\vartheta \in S : \#J(\vartheta) = \hat{K}(q), \text{ and } T_n(Y, \vartheta) \leq q\}.$$

This set is referred to as a *confidence set* for the true step function ϑ. The *Simultaneous Multiscale Changepoint Estimator (SMUCE)* is then defined to be the 'constrained maximum likelihood estimator' within the confidence set $\mathcal{C}(q)$

$$\hat{\vartheta}(q) = \arg\max_{\vartheta \in \mathcal{C}(q)} \sum_{i=1}^{n} \log(f_{\vartheta(i/n)}(Y_i)).$$

This estimator is computed using a dynamic programming framework, which is possible due to the method's consideration of the local likelihoods of the constant parts of ϑ, rather than the full likelihood of the entire sequence. To keep the computational cost to a minimum the intervals searched over are restricted to only intervals of dyadic length, further the value $\hat{\vartheta}(q)$ can be re-written as a penalised cost function and pruning steps incorporated into the implementation (similar to PELT). With these tricks to improve efficiency the computational time can be reduced to $\mathcal{O}(n)$ in the worst case.

Adaptations of SMUCE have been developed in a couple of papers. Futschik et al. (2014) develop the SMUCE method to deal specifically with changepoint detection in Bernoulli and Binomial distributed data. Their method, which they denote B-SMUCE, has specific applications to biological sequences. Pein et al. (2015) introduce the H-SMUCE method. H-SMUCE deals with heterogeneous data, allowing the variance to change at the changepoints as well as the mean.

2.3 Nonparametric Methods

Most of the methods described above make some assumptions on the underlying distribution of the data, for example many assume that the noise is Gaussian. Often such assumptions cannot be justified and a nonparametric changepoint detection method is required.

Nonparametric methods for single changepoint detection have been developed by a number of authors. Pettitt (1979) (and more recently in Hawkins and Deng (2010)) suggest the use of the Mann-Whitney test statistic to detect a single change in the location of the distribution. For detecting a more general class of distributional change Ross and Adams (2012) suggest using either the Kolmogorov-Smirnov or Cramer-von-Mises tests which are both based on the empirical distribution function. Kernel density estimations can also be used for nonparametric detection of a single change (Baron, 2000), however these can be computationally expensive to compute.

Nonparametric methods for detecting multiple changepoints have been looked at less in the literature, and single changepoint detection methods do not easily extend to the multiple changepoint case (Zou et al., 2014). One way in which single nonparametric changepoint detection methods can be extended to the multiple changepoint problem is if they can be formulated as test statistic. Then Binary Segmentation (or alternatively Circular Binary Segmentation or Wild Binary Segmentation) can be used to heuristically detect multiple changepoints, the process for doing this was outlined in Section 2.2.1.

The "E-Divisive" method of Matteson and James (2014) is a non-parametric approach which uses a Binary Segmentation based algorithm. Their method defines a cost function on the data which aims to maximise a Euclidean distance between the two sub-segments at each iteration. Alternatively, in a later paper (James and Matteson, 2015) they fit the same cost function using a dynamic programming approach applying a probablistic pruning step to improve the efficiency of the algorithm.

Alternatively, Ross and Adams (2012) suggest an extension to their single changepoint methods for sequential data, treating the problem as a single changepoint problem that resets every time a new change is detected. Another method is proposed by Lee (1996) and is based on weighted empirical measures over a window, however this is shown to have unsatisfactory results.

A nonparametric likelihood function based on the empirical distribution is used for multiple changepoint detection by Zou et al. (2014). They let $F_i(t)$ be the unknown CDF for the ith segment and $\hat{F}_i(t)$ be the empirical CDF defined as follows

$$\hat{F}_i(t) = \frac{1}{\tau_i - \tau_{i-1}} \left(\sum_{j=\tau_{i-1}+1}^{\tau_i} \mathbb{I}_{\{y_j < t\}} + \frac{1}{2}\mathbb{I}_{\{y_j = t\}} \right).$$

Then based on this, a nonparametric likelihood can be defined

$$\mathcal{L}_{np}(\mathbf{y}_{\tau_{i-1}+1:\tau_i}|t) = (\tau_i - \tau_{i-1}) \left[\hat{F}_i(t) \log \hat{F}_i(t) + (1 - \hat{F}_i(t)) \log(1 - \hat{F}_i(t)) \right].$$

Zou et al. (2014) then use the negative of this as a cost function, which they then minimise using the Segment Neighbourhood Search algorithm (Auger and Lawrence, 1989).

This method performs well, but is computationally expensive ($\mathcal{O}(mn^2 + n^3)$, where m is the number of changepoints and n is the length of the data). This cost comes from adding the costs of precomputing the segment costs ($\mathcal{O}(n^3)$) and running the SNS algorithm ($\mathcal{O}(mn^2)$). Haynes et al. (2016) extend the method to increase the efficiency in both of these areas. They simplify the segment cost by approximation to enable the precomputation of all the segments in $\mathcal{O}(n^2 \log n)$. They also use the more efficient PELT algorithm (Killick et al., 2012) to do the minimisation in expected $\mathcal{O}(n)$, when the number of changepoints is linear in the data length. Their resulting algorithm, nonparametric PELT (NP-PELT), then runs much faster than the original, with an overall expected cost of $\mathcal{O}(n + n^2 \log n)$, and is shown to still perform with reasonable accuracy despite the approximation of the segment costs.

2.4 Fitting More Complex Models

The methods considered so far have all dealt with univariate time series with i.i.d. data in the segements. More complicated data structures often require more specialist algorithms than those presented so far. In this section we will consider two areas in which such specialist algorithms have been developed. First we will look at methods for detecting changes in autoregressive changes before we look at methods for changepoint detection in multivariate time series.

2.4.1 Autoregressive Processes

One area in which changepoint methods have been applied and warrants discussion is detecting changes in autoregressive (AR) processes. Autoregressive processes are random processes which incorporate dependence between values at neighbouring time points and have been used to model speech signal (Davis et al., 2006), EEG (Chan et al., 2013) and earthquake (Adak, 1998) data as well as in other areas.

An autoregressive process of order p, denoted as AR(p), is defined as

$$Y_t = \psi_0 + \sum_{i=1}^{p} \psi_i Y_{t-i} + \sigma^2 Z_t,$$

where $\psi_0, \ldots, \psi_p, \sigma$ are the parameters of the model and Z_t is white noise. We then aim to detect an abrupt change in one of the parameters $\psi_0, \ldots, \psi_p, \sigma$. Note that a change in the parameters of a AR(0) process is a change in either ψ_0 or σ (or both) and corresponds to a change in mean or variance (or both).

Detecting changes in autoregressive processes and the related moving average models, ARMA and ARIMA models have been looked at by a number of authors. For detecting a single change in autoregressive models Basseville and Benveniste (1983) compare the two models (with or without a change) by the use of a test statistic. They suggest using either the log-likelihood ratio test or the Kullback-Leibler divergence (a non symmetric measure of the difference between two probability distributions).

For multiple changepoint detection in AR parameters Hamilton (1990) use an expectation-maximisation (E-M) algorithm. They model the changes in the parameters as outcomes of a discrete valued Markov process. A more general approach for detecting changes in stationary time series is developed and applied to autoregressive processes in Adak (1998). The approach taken here is to estimate the time dependent spectrum use a binary tree to perform the segmentation, to speed the method up they suggest only searching for segments of dyadic length. Another general approach is outlined in Shao and Zhang (2010) where a self normalisation based Kolmogorov-Smirnov test is proposed. This test statistic is asymptotically distribution free and is applied to detecting changes in second order structures and more specifically in an AR(1) process.

The LASSO (Section 2.2.4) is extended for AR processes in Chan et al. (2013). They show that whilst the LASSO overestimates the number of true changepoints, a subset of the estimated set can be selected by a second step selection procedure. They then show that this subset correctly estimates the number and location of the changes with probability tending to one.

The PELT algorithm (discussed in Section 2.2.2) is applied to autoregressive processes in Killick et al. (2012). They look for changes in the autocovariance of AR processes and compare against the approach of Davis et al. (2006) (discussed below). Changes in the autocovariance of AR processes is also considered in Korkas and Fryzlewicz (2015), where they use the WBS algorithm and detect changes using wavelet based non-stationary time series techniques.

ARCH (Autoregressive Conditionally Heteroscedastic) processes are processes where the conditional variance evolves according to an autoregressive process. Changes in ARCH processes are considered by Fryzlewicz and Subba Rao (2014) where they build on the Binary Segmentation algorithm to find changepoints in the FTSE 100 index. Their method, BASTA (Binary Segmentation for Transformed

Autoregressive Conditional Heteroscedasticity), first transforms the data to decorrelate it and lighten the tails, and then performs Binary Segmentation to find the changes.

Further work on detecting changepoints in ARCH models has been widely looked at in the econometrics literature where they are often referred to as “switching ARCH models”. This is first looked at in Cai (1994) where a switching ARCH model is fitted by combining an ARCH process with a switching regime Markov process. Further work on this can be found in, for example, Haas et al. (2004) and Pelletier (2006) and the references therein.

Detecting Changes in AR Models using the Minimum Description Length

One algorithm of note for detecting changes in autoregressive processes is presented in Davis et al. (2006) and minimises a cost based on the minimum description length.

The minimum description length (MDL) principle was first introduced in Rissanen (1989). The idea of MDL is to choose the model which stores the data in the least possible space, or with the smallest code length. Techniques for calculating the MDL for various models by approximating the code length are given by Rissanen (1989).

Using the techniques from Rissanen (1989), Davis et al. (2006) calculate the MDL for a changepoint model where each segment is modelled by an autoregressive process of unknown order. This cost is as follows

$$MDL(m, \tau_1, \ldots, \tau_m, p_1, \ldots, p_{m+1}) = \log m + (m+1)\log n + \sum_{j=1}^{m+1} \left[\log p_j + \frac{p_j + 2}{2} \log s_j + \frac{s_j}{2} \log(2\pi\hat{\sigma}_j^2) \right],$$

where $p_1, \ldots, p_{m+1}$ are the orders of the AR processes for each segment, $s_j = \tau_j - \tau_{j-1}$ the length of segment j (for $j = 1, \ldots, m+1$) and $\hat{\sigma}_j^2$ is the Yule-Walker estimate of σ_j^2.

The best fitting model is then chosen as the model which minimises the MDL. As the search space is large, Davis et al. (2006) suggest performing the minimisation via a genetic algorithm. Genetic algorithms are heuristics and hence do not solve the minimisation problem exactly. Despite this, Davis et al. (2006) show that the method performs well in their simulations however it should be noted that they only compare to other heuristic methods.

2.4.2 Multivariate Data

Most of the methods listed prior to this and the methods developed in this thesis deal with univariate data sets. In this section we look at methods which deal with detecting changepoints in multivariate data sets. Detecting changes in multivariate data is of use in a number of areas including genetics (Zhang et al., 2010; Jeng et al., 2013; Bardwell and Fearnhead, 2016), finance (Cho and Fryzlewicz, 2015), geology (Srivastava and Worsley, 1986), EEG brain wave data (Ombao et al., 2005) and network analysis (Lung-Yut-Fong et al., 2012).

Multivariate changepoints can be categorised into two types; fully multivariate and subset multivariate. Fully multivariate changepoints are changes that are simultaneous across all variables, whereas subset multivariate changepoints will only occur in a subset of the variables. Both types of changepoint can be of interest. For example, in financial stock market data, a fully multivariate change may indicate a stock market crash, whereas a subset multivariate change might indicate an event occurring only in a particular industrial sector.

Fully multivariate changepoint methods

Traditionally, multivariate changepoint methods detect only fully multivariate changes. Detecting fully multivariate changes can be performed by applying univariate methods. The issue with this is that any dependence between the parameters is lost, and often this is undesirable. Therefore many papers in the literature develop methods specifically for fully multivariate changepoint detection. For detecting a single fully multivariate change the earliest contribution can be found in Srivastava and Worsley (1986) where they consider a change in the mean of multivariate Normal observations. Other parametric methods for detecting a single change can be found in Horváth and Hušková (2012) and Batsidis et al. (2013), a nonparametric method is given in Aue et al. (2009).

For multiple fully multivariate changepoint detection Srivastava and Worsley (1986) and Aue et al. (2009) both extend their methods using a Binary Segmentation algorithm. Another method based on Binary Segmentation is the “E-Divisive” method of Matteson and James (2014) which was discussed in Section 2.3, but can also be applied to multivariate time series.

Lavielle and Teyssière (2006) offer a dynamic programming based algorithm similar to the Segment Neighbourhood Search method (Auger and Lawrence, 1989). They use a penalised cost function proportional to the negative log-likelihood, where the penalty is adaptively chosen. Similar methods are described in Maboudou and Hawkins (2009), where the BIC penalty is used making the method less flexible, and James and Matteson (2015) which was discussed in Section 2.3, but also extends to multivariate changepoint detection. A further dynamic programming fully multivariate changepoint method is given by Lung-Yut-Fong et al. (2012), where a nonparametric rank statistic is used.

A Bayesian approach for fully multivariate changepoint detection in tree structured graphical models is given in Schwaller and Robin (2016). Here Bayesian computation techniques for detecting univariate changepoints (Rigaill et al., 2012) and dealing with multivariate time series under a tree structured model (Schwaller et al., 2015) are combined to provide an efficient algorithm.

A method for the detection of fully multivariate changes in auto and cross correlation is given in Ombao et al. (2005). Their approach, SLEX (smooth localised complex exponentials), minimises a penalised cost function over a collection of dyadic length bases. A LASSO based approach can also be used and this is detailed in Vert and Bleakley (2010), where the model fitted is penalised by the total variation rather than the number of changepoints.

Subset multivariate changepoint methods

As discussed before, often multivariate changepoints will only exist in a subset of the variables and methods are needed to detect these.

Binary Segmentation based methods for detecting subset multivariate changepoints have been developed by a number of authors. Zhang et al. (2010) outline a method which models the data as a multivariate Normal with a diagonal covariance matrix with a test statistic based on the log-likelihood. Their method detects changes which affect a large number of variables (known as “common” changes). An extension of this method occurs in Siegmund et al. (2011) where focus is shifted to detecting changes which only affects a small number of variables (known as “rare” changes). Jeng et al. (2013) offer a similar method which is able to detect both rare and common changes. An online adaptation of the method in Siegmund et al. (2011) to detect rare changes is given in Xie and Siegmund (2013).

Maboudou-Tchao and Hawkins (2013) provide another approach, where they first run the dynamic programming method from their earlier paper (Maboudou and Hawkins, 2009) to obtain the fully multivariate changepoints and then perform variable specific hypothesis tests on each estimated changepoint to determine which variables it affects. A further dynamic programming based algorithm is given in Pickering (2015). Their method, Subset Multivariate Optimal Partitioning (SMOP), minimises a penalised cost function exactly, outputting both the changepoints and the subset of affected variables. Pickering (2015) also attempt to apply the pruning techniques of Killick et al. (2012), however they find that doing so is not practically viable.

A Bayesian approach can also be taken. Bardwell and Fearnhead (2016) extend the method of Fearnhead (2005) (discussed in Section 2.2.3) to the subset multivariate changepoint model. Their method, the Bayesian Abnormal Region Detector (BARD) is then used to detect segments in CNV data which differ from a baseline.

Finally, methods for detecting subset multivariate changepoints have also been developed for changes in second order. Cho and Fryzlewicz (2015) develop the Sparsified Binary Segmentation method for both changes in auto and cross covariance and Preuss et al. (2015) develop a nonparametric method for detecting changes in auto covariance.

Chapter 3

On Optimal Multiple Changepoint Algorithms for Large Data

3.1 Introduction

Often time series data experiences multiple abrupt changes in structure which need to be taken into account if the data is to be modelled effectively. These changes, known as changepoints, or breakpoints, cause the data to be split into segments which can then be modelled separately. Detecting changepoints, both accurately and efficiently, is required in a number of applications including bioinformatics (Picard et al., 2011), financial data (Fryzlewicz, 2014), climate data (Killick et al., 2012; Reeves et al., 2007), EEG data (Lavielle, 2005), Oceanography (Killick et al., 2010) and the analysis of speech signals (Davis et al., 2006).

As increasingly large data sets are obtained in modern applications, there is a need for statistical methods for detecting changepoints that are not only accurate but also are computationally efficient. A motivating application area where computational efficiency is important is in detecting copy number variation (Olshen et al., 2004; Zhang et al., 2010). For example, in Section 3.7 we look at detecting changes in DNA copy number in tumour microarray data. Accurate detection of regions in which this copy number is amplified or reduced from a baseline level is crucial as these regions can relate to tumorous cells and their detection is important for classifying tumour progression and type. The data analysis in Section 3.7 involves detecting changepoints in thousands of time series, many of which have hundreds of thousands of data points. Other applications of detecting copy number variation can involve analysing data sets which are orders of magnitude larger still.

There are a wide-range of approaches to detecting changepoints, see for example Frick et al. (2014) and Aue and Horvth (2013) and the references therein. We focus on one important class of approaches (e.g. Braun et al., 2000; Davis et al., 2006; Zhang and Siegmund, 2007) that can be formulated in terms of defining a cost function for a segmentation. They then either minimise a penalised version of this cost (e.g. Yao, 1988; Lee, 1995), which we call the penalised minimisation problem; or minimise the cost under a constraint on the number of changepoints (e.g. Yao and Au, 1989; Braun and Müller, 1998), which we call the constrained minimisation problem. If the cost function depends on the data through a sum of segment-specific costs then the minimisation can be done exactly using dynamic programming (Auger and Lawrence, 1989; Jackson et al., 2005). However these dynamic programming methods have a cost that increases at least quadratically with the amount of data, and is prohibitive for large-data applications.

Alternatively, much faster algorithms exist that provide approximate solutions to the minimisation problem. The most widely used of these approximate techniques is Binary Segmentation (Scott and Knott, 1974). This takes a recursive approach, adding changepoints one at a time, with a new changepoint added in the position that would lead to the largest reduction in cost given the location of previous changepoints. Due to its simplicity, Binary Segmentation is computationally efficient, being roughly linear in the amount of data, however it only provides an approximate solution and can lead to poor estimation of the number and position of changepoints (Killick et al., 2012). Variations of Binary Segmentation, such as Circular Binary Segmentation (Olshen et al., 2004) and Wild Binary Segmentation (Fryzlewicz, 2014), can offer more accurate solutions for slight decreases in the computational efficiency.

An alternative approach is to look at ways of speeding up the dynamic programming algorithms. Recent work has shown this is possible via pruning of the solution space. Killick et al. (2012) present a technique for doing this which we shall refer to as *inequality based pruning*. This forms the basis of their method PELT which can be used to solve the penalised minimisation problem. Rigaill (2015) develop a different pruning technique, *functional pruning*, and this is used in their pDPA method which can be used to solve the constrained minimisation problem. Both PELT and pDPA are optimal algorithms, in the sense that they find the true optimum of the minimisation problem they are trying to solve.

However the pruning approaches they take are very different, and work well in different scenarios. PELT is most efficient in applications where the number of changepoints is large, and pDPA when there are few changepoints.

The focus of this chapter is on these pruning techniques, with the aim of trying to combine ideas from PELT and pDPA. This leads to two new algorithms, Functional Pruning Optimal Partitioning (FPOP) and Segment Neighbourhood with Inequality Pruning (SNIP). SNIP uses inequality based pruning to solve the constrained minimisation problem providing an alternative to pDPA which offers greater versatility, especially in the case of multivariate data. FPOP uses functional pruning to solve the penalised minimisation problem efficiently. We show that FPOP always prunes more than PELT. Empirical results suggest that FPOP is efficient for large data sets regardless of the number of changepoints, and we observe that FPOP has a computational cost that is, in some scenarios, even competitive with Binary Segmentation.

The structure of the chapter is as follows. We introduce the constrained and penalised optimisation problems for segmenting data in the next section. We then review the existing dynamic programming methods and pruning approaches for solving the penalised optimisation problem in Section 3.3 and for solving the constrained optimisation problem in Section 3.4. The new algorithms, FPOP and SNIP, are developed in Section 3.5, and compared empirically and theoretically with existing pruning methods in Section 3.6. We then evaluate FPOP empirically on both simulated and CNV data in Section 3.7. The chapter ends with a discussion.

3.2 Model Definition

Assume we have data ordered by time, though the same ideas extend trivially to data ordered by any other attribute such as position along a chromosome. Denote the data by $\mathbf{y} = (y_1, \dots, y_n)$. We will use the notation that, for $t \geq s$, the set of observations from time s to time t is $\mathbf{y}_{s:t} = (y_s, ..., y_t)$. If we assume that there are k changepoints in the data, this will correspond to the data being split into $k+1$ distinct segments. We let the location of the jth changepoint be τ_j for $j = 1, \dots, k$, and set $\tau_0 = 0$ and $\tau_{k+1} = n$. The jth segment will consist of data points $y_{\tau_{j-1}+1}, \dots, y_{\tau_j}$. We let $\tau = (\tau_0, \dots, \tau_{k+1})$ be the set of changepoints.

The statistical problem we are considering is how to infer both the number of changepoints and their locations. The specific details of any approach will depend on the type of change, such as change in mean, variance or distribution, that we wish to detect. However a general framework that encompasses many changepoint detection methods is to introduce a cost function for each segment. The cost of a segmentation can then be defined in terms of the sum of the costs across the segments, and we can infer segmentations through minimising the segmentation cost.

Throughout we will let $\mathcal{C}(\mathbf{y}_{s+1:t})$, for $s < t$, denote the cost for a segment consisting of data points $y_{s+1}, \dots, y_t$. The cost of a segmentation, $\tau_1, \dots, \tau_k$ is then

$$\sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}). \tag{3.1}$$

The form of this cost function will depend on the type of change we are wanting to detect. One generic approach to defining these segments is to introduce a model for the data within a segment, and then to let the cost be minus the maximum log-likelihood for the data in that segment. If our model assumes that the data is independent and identically distributed with segment-specific parameter μ then

$$\mathcal{C}(\mathbf{y}_{s+1:t}) = \min_{\mu} \sum_{i=s+1}^{t} -\log(p(y_i|\mu)). \tag{3.2}$$

In this formulation we are detecting changes in the value of the parameter, μ, across segments.

For example if μ is the mean in Normally distributed data, with known variance σ^2, then the cost for a segment would simply be

$$\mathcal{C}(\mathbf{y}_{s+1:t}) = \frac{1}{2\sigma^2} \sum_{i=s+1}^{t} \left(y_i - \frac{1}{t-s} \sum_{j=s+1}^{t} y_j \right)^2, \tag{3.3}$$

which is just a quadratic error loss. We have removed a term that does not depend on the data and is linear in segment length, as this term does not affect the solution to the segmentation problem. The cost for a segment can also include a term that depends on the length of segment. Such a cost appears within a minimum description length criteria (Davis et al., 2006), where the cost for a segment $\mathbf{y}_{s+1:t}$ would also include a $\log(t-s)$ term.

3.2.1 Segmenting Data using Penalised and Constrained Optimisation

If we know the number of changepoints in the data, k, then we can infer their location through minimising (3.1) over all segmentations with k changepoints. Normally however k is unknown, and thus has to be estimated. A common approach is to define

$$C_{k,n} = \min_{\boldsymbol{\tau}} \left[\sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) \right], \tag{3.4}$$

the minimum cost of segmenting data $\mathbf{y}_{1:n}$ with k changepoints. As k increases we have more flexibility in our model for the data. If the segment cost is such that the addition of a changepoint will decrease the cost (such as for the negative log-likelihood) then $C_{k,n}$ will be monotonically decreasing in k and estimating the number of changepoints by minimising $C_{k,n}$ is not possible. One solution is to solve (3.4) for a fixed value of k which is either assumed to be known or chosen separately. We call this problem the *constrained minimisation problem.*

If k is not known, then a common approach is to calculate $C_{k,n}$ and the corresponding segmentations for a range of values, $k = 0, 1, \ldots, K$, where K is some chosen maximum number. We can then estimate the number of changepoints by minimising $C_{k,n} + f(k, n)$ over k for some suitable penalty function $f(k, n)$.

Choosing a good value for $f(k, n)$ is still very much an open problem. The most common choices of $f(k, n)$, for example SIC (Schwarz, 1978) and AIC (Akaike, 1974) are linear in k, however these are only consistent in specific cases and rely on assumptions made about the data generating process which in practice is generally unknown. Recent work in Haynes et al. (2015) looks at picking penalty functions in greater detail, offering ranges of penalties that give good solutions. This is discussed in grater detail in Section 2.2.2.

If the penalty function is linear in k, with $f(k, n) = \beta k$ for some $\beta > 0$ (which may depend on n), then we can directly find the number of changepoints and corresponding segmentation by noting that

$$\begin{aligned} \min_{k} \left[C_{k,n} + \beta k \right] &= \min_{k,\boldsymbol{\tau}} \left[\sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) \right] + \beta k \\ &= \min_{k,\boldsymbol{\tau}} \left[\sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \beta \right] - \beta. \end{aligned} \tag{3.5}$$

We call the minimisation problem in (3.5) the *penalised minimisation problem.*

In both the constrained and penalised cases we need to solve a minimisation problem to find the optimal segmentation under our criteria. There are dynamic programming algorithms for solving each of these minimisation problems. For the constrained case this is achieved using the Segment Neighbourhood Search algorithm (see Section 3.4.1), whilst for the penalised case this can be achieved using the Optimal Partitioning algorithm (see Section 3.3.1).

Solving the constrained case offers a way to get segmentations for $k = 0, 1, \ldots, K$ changepoints, and thus gives insight into how the segmentation varies with the number of segments. However, a big advantage of the penalised case is that it incorporates model selection into the problem itself, and therefore it is often computationally more efficient when dealing with an unknown value of k. In the following we will use the terminology optimal segmentation to define segmentations that are the solution to either the penalised or constrained minimisation problem, with the context making it clear as to which minimisation problem it relates to.

3.2.2 Conditions for Pruning

The focus of this chapter is on methods for speeding up these dynamic programming algorithms using pruning methods. The pruning methods can be applied under one of two conditions on the segment costs:

C1 The cost function satisfies

$$\mathcal{C}(\mathbf{y}_{s+1:t}) = \min_{\mu} \sum_{i=s+1}^{t} \gamma(y_i, \mu),$$

for some function $\gamma(\cdot, \cdot)$, with parameter μ.

C2 There exists a constant κ such that for all $s < t < T$,

$$\mathcal{C}(\mathbf{y}_{s+1:t}) + \mathcal{C}(\mathbf{y}_{t+1:T}) + \kappa \leq \mathcal{C}(\mathbf{y}_{s+1:T}).$$

Condition C1 will be used by functional pruning (which is discussed in Sections 3.4.2 and 3.5.1). Condition C2 will be used by the inequality based pruning (Section 3.3.2 and 3.5.2).

Note that C1 is a stronger condition than C2. If C1 holds then C2 also holds with $\kappa = 0$ and this is true for many practical cost functions. For example it is easily seen that for the negative log-likelihood (3.2) C1 holds with $\gamma(y_i, \mu) = -\log(p(y_i|\mu))$ and C2 holds with $\kappa = 0$. By comparison, segment costs that are the sum of (3.2) and a term that depends non-linearly on the length of the segment will obey C2 but not C1.

3.3 Solving the Penalised Optimisation Problem

We first consider solving the penalised optimisation problem (3.5) using a dynamic programming approach. The initial algorithm, Optimal Partitioning (Jackson et al., 2005), will be discussed first before mentioning how pruning can be used to reduce the computational cost.

3.3.1 Optimal Partitioning

Consider segmenting the data $\mathbf{y}_{1:t}$. Denote $F(t)$ to be the minimum value of the penalised cost (3.5) for segmenting such data, with $F(0) = -\beta$. The idea of Optimal Partitioning is to split the minimisation over segmentations into the minimisation over the position of the last changepoint, and then the minimisation over the earlier changepoints. We can then use the fact that the minimisation over the earlier changepoints will give us the value $F(\tau^*)$ for some $\tau^* < t$

$$\begin{aligned}
F(t) &= \min_{\boldsymbol{\tau},k} \sum_{j=0}^{k} \left[\mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \beta\right] - \beta, \\
&= \min_{\boldsymbol{\tau},k} \left\{ \sum_{j=0}^{k-1} \left[\mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \beta\right] + \mathcal{C}(\mathbf{y}_{\tau_k+1:t}) + \beta \right\} - \beta, \\
&= \min_{\tau^*} \left\{ \min_{\boldsymbol{\tau},k'} \sum_{j=0}^{k'} \left[\mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \beta\right] - \beta + \mathcal{C}(\mathbf{y}_{\tau^*+1:t}) + \beta \right\}, \\
&= \min_{\tau^*} \left\{ F(\tau^*) + \mathcal{C}(\mathbf{y}_{\tau^*+1:t}) + \beta \right\}.
\end{aligned}$$

Here note that $\tau_{k+1} = t$ at the second step and $\tau_{k'+1} = \tau_k = \tau^*$ and $k' = k - 1$ at the third step.

Hence we obtain a simple recursion for the $F(t)$ values

$$F(t) = \min_{0 \le \tau < t} \left[F(\tau) + \mathcal{C}(\mathbf{y}_{\tau+1:t}) + \beta\right]. \tag{3.6}$$

The segmentations themselves can be recovered by first taking the arguments which minimise (3.6)

$$\tau_t^* = \operatorname*{arg\,min}_{0 \le \tau < t} \left[F(\tau) + \mathcal{C}(\mathbf{y}_{\tau+1:t}) + \beta\right], \tag{3.7}$$

which give the optimal location of the last changepoint in the segmentation of $y_{1:t}$.

If we denote the vector of ordered changepoints in the optimal segmentation of $y_{1:t}$ by $cp(t)$, with $cp(0) = \emptyset$, then the optimal changepoints up to a time t can be calculated recursively

$$cp(t) = (cp(\tau_t^*), \tau_t^*).$$

As equation (3.6) is calculated for time steps $t = 1, 2, \dots, n$ and each time step involves a minimisation over $\tau = 0, 1, \dots, t-1$ the computation takes $\mathcal{O}(n^2)$ time.

3.3.2 PELT

One way to increase the efficiency of Optimal Partitioning is discussed in Killick et al. (2012) where they introduce the PELT (Pruned Exact Linear Time) algorithm. PELT works by limiting the set of potential previous changepoints (i.e. the set over which τ is chosen in the minimisation in equation 3.6). They show that if condition C2 holds for some κ, and if

$$F(s) + \mathcal{C}(\mathbf{y}_{(s+1:t)}) + \kappa > F(t), \tag{3.8}$$

then at any future time $T > t$, s can never be the optimal location of the most recent changepoint prior to T.

This means that at every time step t the left hand side of equation (3.8) can be calculated for all potential values of the last changepoint. If the inequality holds for any individual s then that s can be discounted as a potential last changepoint for all future times. Thus the update rules (3.6) and (3.7) can be restricted to a reduced set of potential last changepoints, τ, to consider. This set, which we shall denote as R_t, can be updated simply by

$$R_{t+1} = \{\tau \in \{R_t \cup \{t\}\} : F(\tau) + \mathcal{C}(\mathbf{y}_{(\tau+1):t}) + \kappa \leq F(t)\}. \tag{3.9}$$

This pruning technique, which we shall refer to as *inequality based pruning*, forms the basis of the PELT method.

Since at each time step in the PELT algorithm the minimisation is being run over fewer values it is expected that this method will be more efficient than the basic Optimal Partitioning algorithm. In Killick et al. (2012) it is shown to be at least as efficient as Optimal Partitioning, with PELT's computational cost being bounded above by $\mathcal{O}(n^2)$. Under certain conditions the expected computational cost can be shown to be bounded by Ln for some constant $L < \infty$. These conditions are given fully in Killick et al. (2012), the most important of which is that the expected number of changepoints in the data increases linearly with the length of the data, n. This intuitively makes sense as the more changepoints in the data set the more points which can be pruned, hence the reduction in computationally time.

3.4 Solving the Constrained Optimisation Problem

We now consider applications of dynamic programming to solve the constrained optimisation problem (3.4). These methods assume a maximum number of changepoints that are to be considered, K, and then solve the constrained optimisation problem for all values of $k = 1, 2, \dots, K$. We first describe the initial algorithm, Segment Neighbourhood Search (Auger and Lawrence, 1989), and then an approach that uses pruning.

3.4.1 Segment Neighbourhood Search

Take the constrained case (3.4) which segments the data up to t, for $t \geq k+1$, into $k+1$ segments (using k changepoints), and denote the minimum value of the cost by $C_{k,t}$. The idea of Segment Neighbourhood Search is to derive a relationship between $C_{k,t}$ and $C_{k-1,s}$ for $s < t$:

$$\begin{aligned} C_{k,t} &= \min_{\boldsymbol{\tau}} \sum_{j=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}), \\ &= \min_{\tau_k} \left[\min_{\boldsymbol{\tau}_{1:k-1}} \sum_{j=0}^{k-1} \mathcal{C}(\mathbf{y}_{\tau_j+1:\tau_{j+1}}) + \mathcal{C}(\mathbf{y}_{\tau_k+1:\tau_{k+1}}) \right], \\ &= \min_{\tau_k} \left[C_{k-1,\tau_k} + \mathcal{C}(\mathbf{y}_{\tau_k+1:\tau_{k+1}}) \right]. \end{aligned}$$

Thus the following recursion is obtained:

$$C_{k,t} = \min_{\tau \in \{k,\dots,t-1\}} \left[C_{k-1,\tau} + \mathcal{C}(\mathbf{y}_{\tau+1:t}) \right]. \tag{3.10}$$

If this is run for all values of t up to n and for $k = 2, \dots, K$, then the exact segmentations with $1, \dots, K$ segments can be acquired.

To extract the exact segmentation we first let $\tau_l^*(t)$ denote the optimal position of the last changepoint if we segment data $\mathbf{y}_{1:t}$ using l changepoints. This can be calculated as

$$\tau_l^*(t) = \arg\min_{\tau \in \{l,\dots,t-1\}} \left[C_{l-1,\tau} + \mathcal{C}(\mathbf{y}_{\tau+1:t}) \right].$$

Then if we let $(\tau_1^k, \dots, \tau_k^k)$ be the set of changepoints in the segmentation of $\mathbf{y}_{1:n}$ into $k+1$ segments, we have $\tau_k^k = \tau_k^*(n)$. Furthermore we can calculate the other changepoint positions recursively for $l = k-1, \dots, 1$ using

$$\tau_l^k(n) = \tau_l^*(\tau_{l+1}^k).$$

For a fixed value of k equation (3.10) is computed for $t \in 1, \ldots, n$. Then for each t the minimisation is done for $\tau = 1, \ldots, t-1$. This means that $\mathcal{O}(n^2)$ calculations are needed. However, to also identify the optimal number of changepoints this then needs to be done for $k \in 1, \ldots, K$ so the total computational cost in time can be seen to be $\mathcal{O}(Kn^2)$.

3.4.2 Pruned Segment Neighbourhood Search

Rigaill (2015) has developed techniques to increase the efficiency of Segment Neighbourhood Search using functional pruning. These form the basis of a method called pruned Dynamic Programming Algorithm (pDPA). A more generic implementation of this method is presented in Cleynen et al. (2014). Here we describe how this algorithm can be used to calculate the $C_{k,t}$ values. Once these are calculated, the exact segmentation can be extracted as in Segment Neighbourhood Search.

Assuming condition C1, the segment cost function can be split into the component parts $\gamma(y_i, \mu)$, which depend on the parameter μ. We can then define new cost functions, $Cost^{\tau}_{k,t}(\mu)$, as the minimal cost of segmenting data $y_{1:t}$ into k segments, with a most recent changepoint at τ, and where the segment after τ is conditioned to have parameter μ. Thus for $\tau \leq t-1$,

$$Cost^{\tau}_{k,t}(\mu) = C_{k-1,\tau} + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu), \tag{3.11}$$

and $Cost^{t}_{k,t}(\mu) = C_{k-1,t}$.

These functions, which are stored for each candidate changepoint, can then be updated at each new time step as for $\tau \leq t-1$

$$Cost^{\tau}_{k,t}(\mu) = Cost^{\tau}_{k,t-1}(\mu) + \gamma(y_t, \mu). \tag{3.12}$$

By taking the minimum of $Cost^{\tau}_{k,t}(\mu)$ over μ, the individual terms of the right hand side of equation (3.10) can be recovered. Therefore, by further minimising over τ, the minimum cost $C_{k,t}$ can be returned

$$\begin{aligned}
\min_{\tau} \min_{\mu} Cost^{\tau}_{k,t}(\mu) &= \min_{\tau} \min_{\mu} \left[C_{k-1,\tau} + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu) \right], \\
&= \min_{\tau} \left[C_{k-1,\tau} + \min_{\mu} \sum_{i=\tau+1}^{t} \gamma(y_i, \mu) \right], \\
&= \min_{\tau} \left[C_{k-1,\tau} + \mathcal{C}(\mathbf{y}_{\tau+1:t}) \right], \\
&= C_{k,t}.
\end{aligned}$$

By interchanging the order of minimisation the values of the potential last changepoint, τ, can be pruned whilst allowing for changes in μ. First we define the function $Cost^{*}_{k,t}(\mu)$ as follows

$$Cost^{*}_{k,t}(\mu) = \min_{\tau} Cost^{\tau}_{k,t}(\mu).$$

We can now get a recursion for $Cost^{*}_{k,t}(\mu)$ by splitting the minimisation over the most recent changepoint τ into the two cases $\tau \leq t-1$ and $\tau = t$:

$$\begin{aligned} Cost^{*}_{k,t}(\mu) &= \min\left\{\min_{\tau\leq t-1} Cost^{\tau}_{k,t}(\mu)\ ,\ Cost^{t}_{k,t}(\mu)\right\} \\ &= \min\left\{\min_{\tau\leq t-1} Cost^{\tau}_{k,t-1}(\mu) + \gamma(y_t,\mu)\ ,\ C_{k-1,t}\right\}, \end{aligned}$$

which gives

$$Cost^{*}_{k,t}(\mu) = \min\left\{Cost^{*}_{k,t-1}(\mu) + \gamma(y_t,\mu)\ ,\ C_{k-1,t}\right\}.$$

The idea of pDPA is to use this recursion for $Cost^{*}_{k,t}(\mu)$. We can then use the fact that $C_{k,t} = \min_{\mu} Cost^{*}_{k,t}(\mu)$ to calculate the $C_{k,t}$ values. In order to do this we need to be able to represent this function of μ in an efficient way. This can be done if μ is a scalar, because for any value of μ, $Cost^{*}_{k,t}(\mu)$ is equal to the value of $Cost^{\tau}_{k,t}(\mu)$ for some value of τ. Thus we can partition the possible values of μ into intervals, with each interval corresponding to a value for τ for which $Cost^{*}_{k,t}(\mu) = Cost^{\tau}_{k,t}(\mu)$.

To make the idea concrete, an example of $Cost^{*}_{k,t}(\mu)$ is given in Figure 3.1 for a change in mean using the cost function given in (3.3). As each $\gamma(y_i,\mu)$ is quadratic in μ then the sum of these, $Cost^{\tau}_{k,t}(\mu)$, is also a quadratic function in this case. In this example there are 8 intervals of μ corresponding to 7 different values of τ for which $Cost^{*}_{k,t}(\mu) = Cost^{\tau}_{k,t}(\mu)$. The pDPA algorithm needs to just store the 7 different $Cost^{\tau}_{k,t}(\mu)$ functions, and the corresponding sets.

Formally speaking we define the set of intervals for which $Cost^{*}_{k,t}(\mu) = Cost^{\tau}_{k,t}(\mu)$ as $Set^{\tau}_{k,t}$. The recursion for $Cost^{*}_{k,t}(\mu)$ can be used to induce a recursion for these sets. First define:

$$I^{\tau}_{k,t} = \{\mu : Cost^{\tau}_{k,t}(\mu) \leq C_{k-1,t}\}. \tag{3.13}$$

Then, for $\tau \leq t-1$ we have

$$\begin{aligned} Set^{\tau}_{k,t} &= \left\{\mu : Cost^{\tau}_{k,t}(\mu) = Cost^{*}_{k,t}(\mu)\right\} \\ &= \left\{\mu : Cost^{\tau}_{k,t-1}(\mu) + \gamma(y_t,\mu) = \min\left\{Cost^{*}_{k,t-1}(\mu) + \gamma(y_t,\mu), C_{k-1,t}\right\}\right\}. \end{aligned}$$

Remembering that $Cost^{\tau}_{k,t-1}(\mu)+\gamma(y_t,\mu) \geq Cost^{*}_{k,t-1}(\mu)+\gamma(y_t,\mu)$, we have that for μ to be in $Set^{\tau}_{k,t}$ we need that $Cost^{\tau}_{k,t-1}(\mu) = Cost^{*}_{k,t-1}(\mu)$, and that $Cost^{\tau}_{k,t-1}(\mu)+\gamma(y_t,\mu) \leq C_{k-1,t}$. The former condition

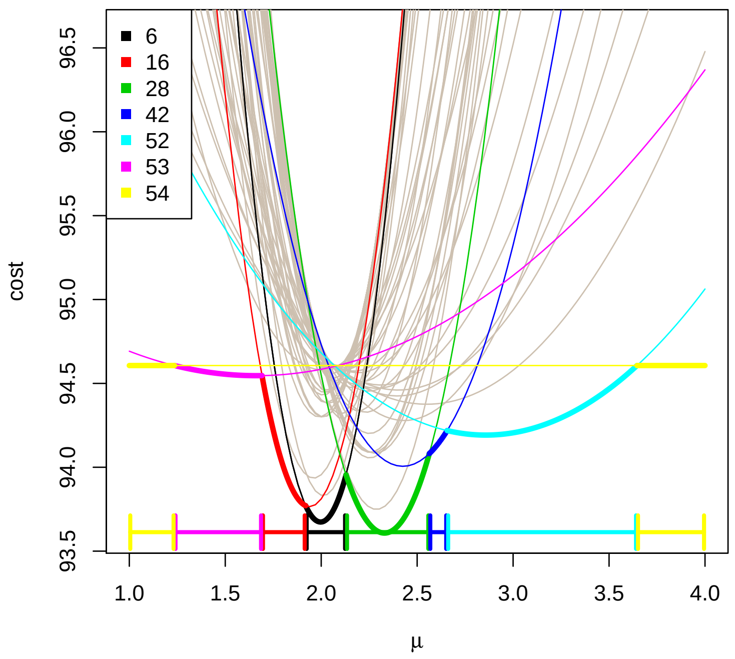


Figure 3.1: Cost functions, $Cost_{k,\tau}(\mu, t)$ for $\tau = 0, \ldots, 54$ and $t = 54$ and the corresponding $C_k^*(\mu, t)$ (in bold) for a change in mean using the negative normal log-likelihood cost function (3.3). Coloured lines correspond to $Cost_{k,\tau}(\mu, t)$ that contribute to $C_k^*(\mu, t)$, with the coloured horizontal lines showing the intervals of μ for which each value of τ is such that $Cost_{k,\tau}(\mu, t) = C_k^*(\mu, t)$. Faded lines correspond to candidates which have previously been pruned, and do not contribute to $C_k^*(\mu, t)$.

corresponds to μ being in $Set_{k,t-1}^{\tau}$ and the second that μ is in $I_{k,t}^{\tau}$. So for $\tau \leq t - 1$

$$Set_{k,t}^{\tau} = Set_{k,t-1}^{\tau} \cap I_{k,t}^{\tau}.$$

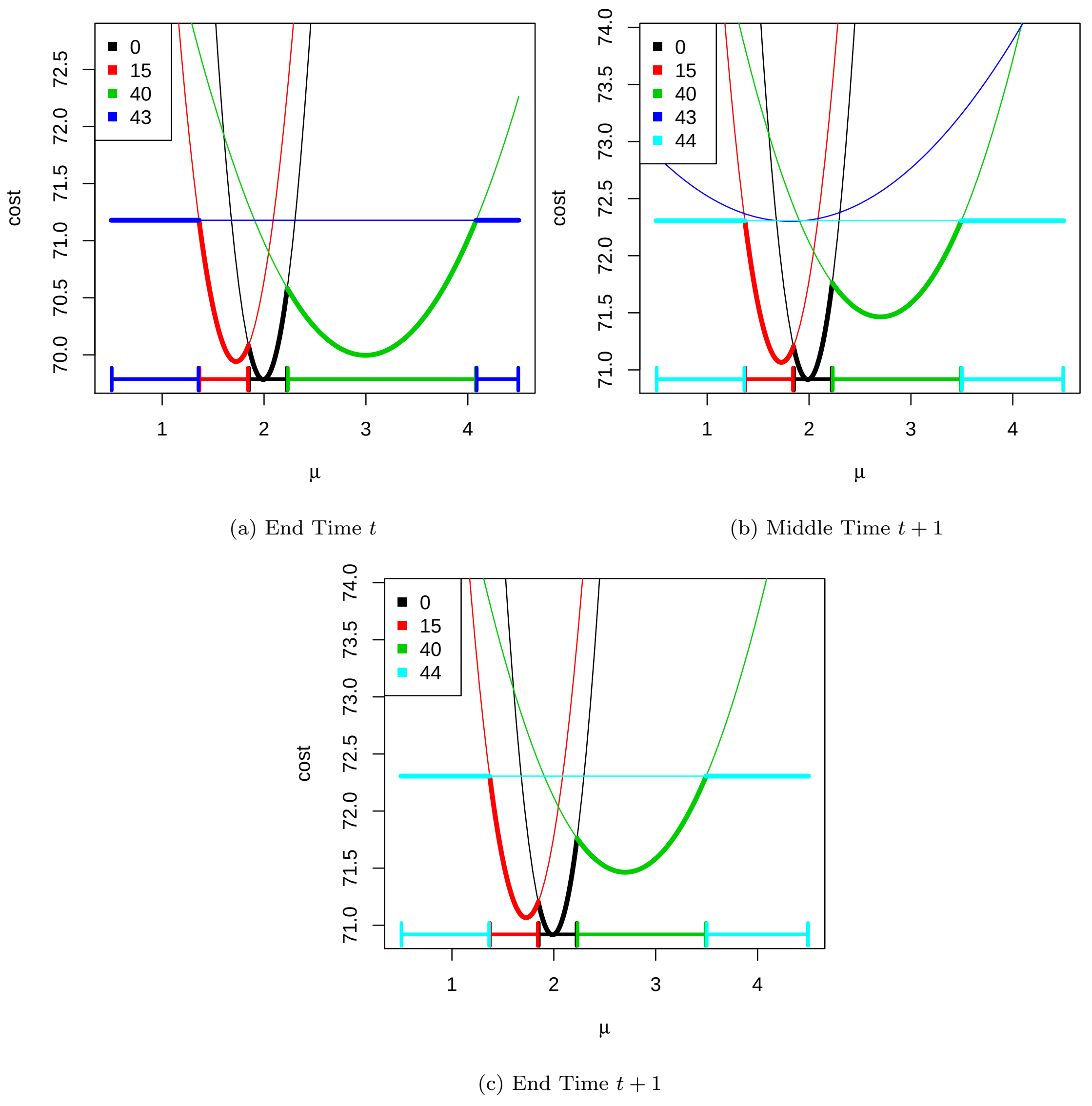
If this $Set_{k,t}^{\tau} = \emptyset$ then the value τ can be pruned, as $Set_{k,T}^{\tau} = \emptyset$ for all $T > t$.

If we denote the range of values μ can take to be D, then we further have that

$$Set_{k,t}^{t} = D \backslash \left[\bigcup_{\tau} I_{k,t}^{\tau} \right],$$

where t can be pruned straight away if $Set_{k,t}^{t} = \emptyset$.

An example of the pDPA recursion is given in Figure 3.2 for a change in mean using the negative normal log-likelihood cost function (3.3). Figure 3.2(a) shows $Cost_{k,t}^{*}(\mu)$ (the bold line). In this example there are 5 intervals of μ corresponding to 4 different values of τ for which $Cost_{k,t}^{*}(\mu) = Cost_{k,t}^{\tau}(\mu)$. When we analyse the next data point, we update each of these four $Cost_{k,t}^{\tau}(\mu)$ functions, using $Cost_{k,t+1}^{\tau}(\mu) = Cost_{k,t}^{\tau}(\mu) + \gamma(y_{t+1}, \mu)$, and introduce a new curve corresponding to a change-point at time $t + 1$,



(a) End Time t

(b) Middle Time $t + 1$

(c) End Time $t + 1$

Figure 3.2: Example of the pDPA algorithm over two time-steps. On each plot we show individual $Cost^{\tau}_{k,t}(\mu)$ functions that are stored, together with the intervals (along the bottom) for which each candidate last changepoint is optimal. In bold is the value of $Cost^{*}_{k,t}(\mu)$. For this example $t = 43$ and we are detecting a change in mean (see Section 3.2). (a) 4 candidates are optimal for some interval of μ, however at $t = 44$ (b), when the candidate functions are updated and the new candidate is added, then the candidate $\tau = 43$ is no longer optimal for any μ and hence can be pruned (c).

$Cost^{t+1}_{k,t+1}(\mu) = C_{k-1,t+1}$ (Figure 3.2(b)). We can then prune the functions which are no longer optimal for any μ values, and in this case we remove one such function (Figure 3.2(c)).

pDPA can be shown to be bounded in time by $\mathcal{O}(Kn^2)$. Rigaill (2015) further analyse the time complexity of pDPA and show it empirically to be $\mathcal{O}(Kn \log n)$, further indications towards this will be presented in Section 3.7. However pDPA has a computational overhead relative to Segment Neighbourhood Search, as it requires calculating and storing the $Cost^{\tau}_{k,t}(\mu)$ functions and the corresponding sets $Set^{\tau}_{k,t}$. Currently implementations of pDPA have only been possible for models with scalar segment parameters μ, due to the difficulty of calculating the sets in higher dimensions. Being able to efficiently store and update the $Cost^{\tau}_{k,t}(\mu)$ has also restricted applications primarily to models where $\gamma(y, \mu)$ corresponds to the log-likelihood of an exponential family. However this still includes a wide-range of changepoint applications, including that of detecting CNVs that we consider in Section 3.7. The cost of updating the sets depends heavily on whether the updates (3.13) can be calculated analytically, or whether they require the use of numerical methods.

3.5 New Changepoint Algorithms

Two natural ways of extending the two methods introduced above will be examined in this section. These are, respectively, to apply functional pruning (Section 3.4.2) to Optimal Partitioning, and to apply inequality based pruning (Section 3.3.2) to Segment Neighbourhood Search. These lead to two new algorithms, which we call Functional Pruning Optimal Partitioning (FPOP) and Segment Neighbourhood with Inequality Pruning (SNIP).

3.5.1 Functional Pruning Optimal Partitioning

Functional Pruning Optimal Partitioning (FPOP) provides a version of Optimal Partitioning (Jackson et al., 2005) which utilises functional pruning to increase the efficiency. As will be discussed in Section 3.6 and shown in Section 3.7, FPOP provides an alternative to PELT which is more efficient in certain scenarios. The approach used by FPOP is similar to the approach for pDPA in Section 3.4.2, however the theory is slightly simpler here as there is no longer the need to condition on the number of changepoints.

We assume condition C1 holds, that the cost function, $\mathcal{C}(\mathbf{y}_{\tau+1:t})$, can be split into component parts $\gamma(y_i, \mu)$ which depend on the parameter μ. Cost functions $Cost^{\tau}_{t}$ can then be defined as the minimal cost of the data up to time t, conditional on the last changepoint being at τ and the last segment having parameter μ. Thus for $\tau \leq t-1$

$$Cost_t^{\tau}(\mu) = F(\tau) + \beta + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu), \tag{3.14}$$

and $Cost_t^t(\mu) = F(t) + \beta$.

These functions, which only need to be stored for each candidate changepoint, can then be recursively updated at each time step, $\tau \leq t-1$

$$Cost_t^{\tau}(\mu) = Cost_{t-1}^{\tau}(\mu) + \gamma(y_t, \mu). \tag{3.15}$$

Given the cost functions $Cost_t^{\tau}(\mu)$ the minimal cost $F(t)$ can be returned by minimising over both τ and μ:

$$\begin{aligned}
\min_{\tau}\min_{\mu} Cost_t^{\tau}(\mu) &= \min_{\tau}\min_{\mu}\left[F(\tau) + \beta + \sum_{i=\tau+1}^{t}\gamma(y_i,\mu)\right], \\
&= \min_{\tau}\left[F(\tau) + \beta + \min_{\mu}\sum_{i=\tau+1}^{t}\gamma(y_i,\mu)\right], \\
&= \min_{\tau}\left[F(\tau) + \beta + \mathcal{C}(\mathbf{y}_{\tau+1:t})\right], \\
&= F(t).
\end{aligned}$$

As before, by interchanging the order of minimisation, the values of the potential last changepoint, τ, can be pruned whilst allowing for a varying μ. Firstly we will define the function $Cost_t^*(\mu)$, the minimal cost of segmenting data $y_{1:t}$ conditional on the last segment having parameter μ:

$$Cost_t^*(\mu) = \min_{\tau} Cost_t^{\tau}(\mu).$$

Note that if a potential last changepoint τ_1 doesn't form part of the piecewise function $Cost_t^*(\mu)$ for a time t (i.e. there doesn't exist μ such that $Cost_t^*(\mu) = Cost_t^{\tau_1}(\mu)$), then this implies that for any given μ we can find τ_2 such that $Cost_t^{\tau_2}(\mu) < Cost_t^{\tau_1}(\mu)$ and further, from the recursion given in (3.15), $Cost_T^{\tau_2}(\mu) < Cost_T^{\tau_1}(\mu)$ for all $T > t$. Hence if τ_1 doesn't form part of the piecewise function $Cost_t^*(\mu)$ at time t then it can be pruned from all future time steps.

We will update these functions recursively over time, and use $F(t) = \min_{\mu} Cost_t^*(\mu)$ to then obtain the solution of the penalised minimisation problem. The recursions for $Cost_t^*(\mu)$ are obtained by splitting the minimisation over τ into $\tau \leq t-1$ and $\tau = t$

$$\begin{aligned} Cost_t^*(\mu) &= \min\left\{ \min_{\tau \leq t-1} Cost_t^\tau(\mu)\ ,\ Cost_t^t(\mu) \right\}, \\ &= \min\left\{ \min_{\tau \leq t-1} Cost_{t-1}^\tau(\mu) + \gamma(y_t,\mu)\ ,\ Cost_t^t(\mu) \right\}, \end{aligned}$$

which then gives:

$$Cost_t^*(\mu) = \min\{Cost_{t-1}^*(\mu) + \gamma(y_t,\mu)\ ,\ F(t) + \beta\}.$$

To implement this recursion we need to be able to efficiently store and update $Cost_t^*(\mu)$. As before we do this by partitioning the space of possible μ values, D, into sets where each set corresponds to a value τ for which $Cost_t^*(\mu) = Cost_t^\tau(\mu)$. We then need to be able to update these sets, and store $Cost_t^\tau(\mu)$ just for each τ for which the corresponding set is non-empty.

This can be achieved by first defining

$$I_t^\tau = \{\mu : Cost_t^\tau(\mu) \leq F(t) + \beta\}. \tag{3.16}$$

Then, for $\tau \leq t-1$, we define

$$\begin{aligned} Set_t^\tau &= \{\mu : Cost_t^\tau(\mu) = Cost_t^*(\mu)\} \\ &= \{\mu : Cost_{t-1}^\tau(\mu) + \gamma(y_t,\mu) = \min\{Cost_{t-1}^*(\mu) + \gamma(y_t,\mu)\ ,\ F(t) + \beta\}\} \end{aligned}$$

Remembering that $Cost_{t-1}^\tau(\mu) + \gamma(y_t,\mu) \geq Cost_{t-1}^*(\mu) + \gamma(y_t,\mu)$; we have that for μ to be in Set_t^τ we need that $Cost_{t-1}^\tau(\mu) = Cost_{t-1}^*(\mu)$, and that $Cost_{t-1}^\tau(\mu) + \gamma(y_t,\mu) \leq F(t) + \beta$. The former condition corresponds to μ being in Set_{t-1}^τ and the second that μ is in I_t^τ, so for $\tau \leq t-1$

$$Set_t^\tau = Set_{t-1}^\tau \cap I_t^\tau.$$

If $Set_t^\tau = \emptyset$ then the value τ can be pruned, as then $Set_T^\tau = \emptyset$ for all $T > t$.

If we denote the range of values μ can take to be D, then we further have that

$$Set_t^t = D \backslash \left[\bigcup_\tau I_t^\tau \right],$$

where t can be pruned straight away if $Set_t^t = \emptyset$.

This updating of the candidate functions and sets is illustrated in Figure 3.3 where the *Cost* functions and *Set* intervals are displayed across two time steps. In this example a change in mean has been considered, using the negative normal log-likelihood cost function (3.3). As each $\gamma(y_i,\mu)$ is quadratic in μ then the sum of these, $Cost^{\tau}_{k,t}(\mu)$, is also a quadratic function in this case. The bold line on Figure 3.3(a) corresponds to the function $Cost^{*}_{t}(\mu)$ and is made up of 7 pieces which relate to 6 candidate last changepoints. As the next time point is analysed the six $Cost^{\tau}_{t}(\mu)$ functions are updated using the formula $Cost^{\tau}_{t+1}(\mu) = Cost^{\tau}_{t}(\mu)+\gamma(y_{t+1},\mu)$ and a new function, $Cost^{t+1}_{t+1}(\mu) = F(t+1)+\beta$, is introduced corresponding to placing a changepoint at time $t+1$ (Figure 3.3(b)). The functions which are no longer optimal for any values of μ (i.e. do not form any part of $Cost^{*}_{t+1}(\mu)$) can then be pruned, and one such function is removed in Figure 3.3(c).

Once again we denote the set of potential last changes to consider as R_t and then restrict the update rules (3.6) and (3.7) to $\tau \in R_t$. This set can then be recursively updated at each time step

$$R_{t+1} = \{\tau \in \{R_t \cup \{t\}\} : Set^{\tau}_{t} \neq \emptyset\}. \tag{3.17}$$

These steps can then be applied directly to an Optimal Partitioning algorithm to form the FPOP method and the full pseudocode for this is presented in Algorithm 2.

3.5.2 Segment Neighbourhood with Inequality Pruning

In a similar vein to Section 3.5.1, Segment Neighbourhood Search can also benefit from using pruning methods. In Section 3.4.2 the method pDPA was discussed as a fast pruned version of Segment Neighbourhood Search. In this section a new method, Segment Neighbourhood with Inequality Pruning (SNIP), will be introduced. This takes the Segment Neighbourhood Search algorithm and uses inequality based pruning to increase the speed.

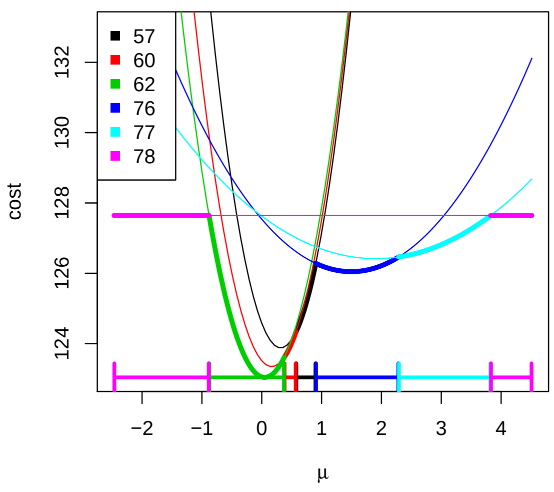
Under condition (C2) the following result can be proved for Segment Neighbourhood Search and this will enable points to be pruned from the candidate changepoint set.

Theorem 1 *Assume that there exists a constant, κ, such that condition C2 holds. If, for any $k \geq 1$ and $s < t$*

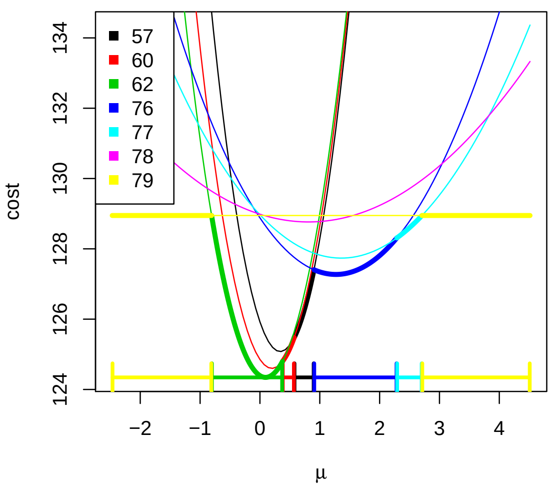
$$C_{k-1,s} + \mathcal{C}(\mathbf{y}_{s+1:t}) + \kappa > C_{k-1,t} \tag{3.18}$$

then at any future time $T > t$, s cannot be the position of the last changepoint in the exact segmentation of $y_{1:T}$ with k changepoints.

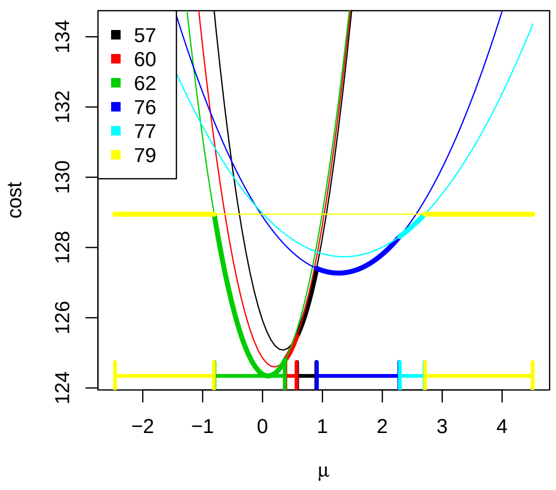
Proof. The idea of the proof is to show that a segmentation of $y_{1:T}$ into k segments with the last changepoint at t will be better than one with the last changepoint at s for all $T > t$.



(a) End Time t



(b) Middle Time $t+1$



(c) End Time $t+1$

Figure 3.3: Candidate functions over two time steps, the intervals shown along the bottom correspond to the intervals of μ for which each candidate last changepoint is optimal. When $t = 78$ (a) 6 candidates are optimal for some interval of μ, however at $t = 79$ (b), when the candidate functions are updated and the new candidate is added, then candidate $\tau = 78$ is no longer optimal for any μ and hence can be pruned (c).

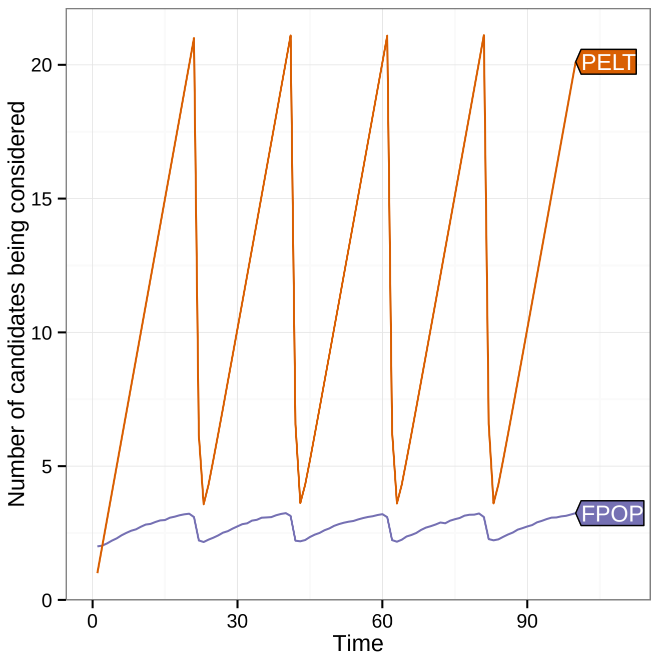


Figure 3.4: Comparison of the number of candidate changepoints stored over time by FPOP and PELT. Averaged over 1000 data sets with changepoints at $t = 20, 40, 60$ and 80.

Assume that (3.18) is true. Now for any $t < T \leq n$

$$C_{k-1,s} + \mathcal{C}(\mathbf{y}_{s+1:t}) + \kappa > C_{k-1,t},$$

$$C_{k-1,s} + \mathcal{C}(\mathbf{y}_{s+1:t}) + \kappa + \mathcal{C}(\mathbf{y}_{t+1:T}) > C_{k-1,t} + \mathcal{C}(\mathbf{y}_{t+1:T}),$$

$$C_{k-1,s} + \mathcal{C}(\mathbf{y}_{s+1,T}) > C_{k-1,t} + \mathcal{C}(\mathbf{y}_{t+1,T}), \qquad \text{(by C2)}.$$

Therefore for any $T > t$ the cost $C_{k-1,s} + \mathcal{C}(\mathbf{y}_{s+1,T}) > C_{k,T}$ and hence s cannot be the optimal location of the last changepoint when segmenting $\mathbf{y}_{1:T}$ with k changepoints. ■

Theorem 1 implies that the update rule (3.10) can be restricted to a reduced set over τ of potential last changes to consider without losing the exactness of Segment Neighbourhood Search. This set, which we shall denote as $R_{k,t}$, can be updated simply by

$$R_{k,t+1} = \{v \in \{R_{k,t} \cup \{t\}\} : C_{k-1,v} + \mathcal{C}(\mathbf{y}_{v+1,t}) + \kappa < C_{k-1,t}\}. \qquad (3.19)$$

This new algorithm, SNIP, is described fully in Algorithm 3.

3.6 Comparisons Between Pruning Methods

Functional and inequality based pruning both offer increases in the efficiency in solving both the penalised and constrained problems, however their use depends on the assumptions which can be made on the cost

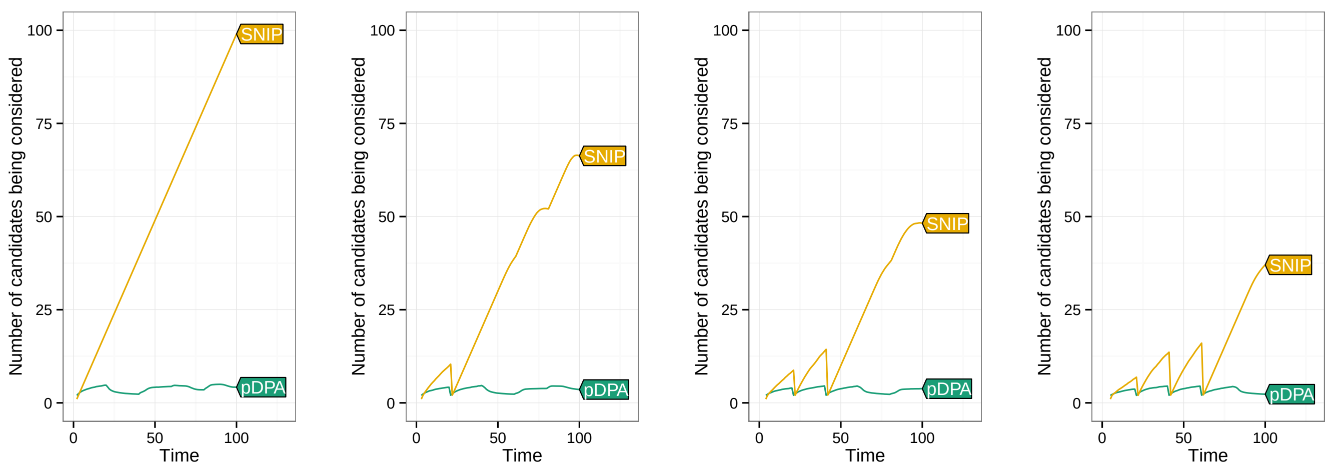


Figure 3.5: Comparison of the number of candidate changepoints stored over time by pDPA and SNIP at multiple values of k in the algorithms (going from left to right $k = 2, 3, 4, 5$). Averaged over 1000 data sets with changepoints at $t = 20, 40, 60$ and 80.

function. Inequality based pruning is dependent on the assumption C2, while functional pruning requires the slightly stronger condition C1.

Functional pruning also requires a larger computational overhead than inequality based pruning. This arises due to the potential difficulties in calculating Set_t^{τ} for all τ at a given timepoint t. If this calculation can be done efficiently (ie. for a univariate parameter from a model in the exponential family, where the intervals can be calculated analytically) then the algorithm (such as FPOP or pDPA) will be efficient too. In particular, this is infeasible (at least using current approaches) for multi-dimensional parameters, as in this case the intervals Set_t^{τ} are also multi-dimensional.

If we consider models for which both pruning methods can be implemented, we can compare the extent to which the methods prune. This will give some insight into when the different pruning methods would be expected to work well. To explore this in Figures 3.4 and 3.5 we look at the amount of candidates stored by functional and inequality based pruning in each of the two optimisation problems.

As Figure 3.4 illustrates, PELT prunes very rarely; only when evidence of a change is particularly high. In contrast, FPOP prunes more frequently keeping the candidate set small throughout. Figure 3.5 shows similar results for the constrained problem. While pDPA constantly prunes, SNIP only prunes sporadically. In addition SNIP fails to prune much at all for low values of k.

Figures 3.4 and 3.5 give strong empirical evidence that functional pruning prunes more points than the inequality based method. In fact it can be shown that any point pruned by inequality based pruning will also be pruned at the same time step by functional pruning. This result holds for both the penalised and constrained case and is stated formally in Theorem 2.

Theorem 2 *Let $\mathcal{C}(\cdot)$ be a cost function that satisfies condition C1, and consider solving either the constrained or penalised optimisation problem using dynamic programming and either inequality or functional pruning.*

Any point pruned by inequality based pruning at time t will also have been pruned by functional pruning at the same time.

Proof. We prove this for pruning of Optimal Partitioning, with the ideas extending directly to the pruning of the Segment Neighbourhood algorithm.

For a cost function which can be decomposed into pointwise costs, it's clear that condition C2 holds when $\kappa = 0$ and hence inequality based pruning can be used. Recall that the point τ (where $\tau < t$, the current time point) is pruned by inequality based pruning in the penalised case if

$$F(\tau) + \mathcal{C}(\mathbf{y}_{\tau+1:t}) \geq F(t),$$

Then, by letting $\hat{\mu}_\tau$ be the value of μ such that $Cost_t^\tau(\mu)$ is minimised, this is equivalent to

$$Cost_t^\tau(\hat{\mu}_\tau) - \beta \geq F(t),$$

Which can be generalised for all μ to

$$Cost_t^\tau(\mu) \geq F(t) + \beta.$$

Therefore equation (3.16) holds for no value of μ and hence $I_t^\tau = \emptyset$ and furthermore $Set_{t+1}^\tau = Set_t^\tau \cap I_t^\tau = \emptyset$ meaning that τ is pruned under functional pruning. ■

3.7 Empirical evaluation of FPOP

As explained in Section 3.6 functional pruning leads to a better pruning in the following sense: any point pruned by inequality based pruning will also be pruned by functional pruning. However, functional pruning is computationally more demanding than inequality based pruning. We thus decided to empirically compare the performance of FPOP to PELT (Killick et al., 2012), pDPA (Rigaill, 2015), Binary Segmentation (BinSeg), Wild Binary Segmentation (WBS) (Fryzlewicz, 2014) and SMUCE (Frick et al., 2014). Due to the number of candidates stored by the SNIP algorithm, especially for low values of k (as shown in Figure 3.5), it is inefficient and for this reason we exclude it from our simulations.

PELT and pDPA have been discussed in Sections 3.3.2 and 3.4.2 respectively. Binary Segmentation (Scott and Knott, 1974) involves the entire data being scanned for a single changepoint and then splitting into two segments around this change. The process is then repeated on these two segments. This recursion is repeated until a certain criterion is satisfied. Wild Binary Segmentation (Fryzlewicz, 2014) takes this method further, taking a randomly drawn number of subsamples from the data and searching these subsamples for a changepoint. As before the data is then split around the changepoint and the process repeated on the two created segments. Lastly SMUCE (Simultaneous Multiscale Changepoint Inference) (Frick et al., 2014) uses a multiscale test at level α and estimates a step function that minimises the number of changepoints whilst lying in the acceptance region of this test. Binary Segmentation and Wild Binary Segmentation are discussed in more detail in Section 2.2.1 and SMUCE in Section 2.2.5.

To do the analysis, we implement FPOP for the quadratic loss (3.3) in C++, the code for this can be found in the opfp project repository on R-Forge:
`https://r-forge.r-project.org/R/?group_id=1851`. We assess the runtimes of FPOP on both real microarray data as well as synthetic data. All algorithms were implemented in C++.

The individual code for all the algorithms considered here is freely avaliable online. For SMUCE we use the code from the `stepR` package, for WBS we use the code from the `wbs` package, for PELT we use the code from the `changepoint` package, for pDPA we use the code from the `cghseg` package and lastly the code for Binary Segmentation and FPOP can be found as part of the `fpop` package. All of these packages are avaliable on CRAN with the exception of `fpop` which can be found on the R-Forge repository given above.

The FPOP algorithm uses, on average, 280MB of RAM for a data set of length 10^7.

3.7.1 Speed benchmark: 4467 chromosomes from tumour microarrays

Hocking et al. (2014) proposed to benchmark the speed of segmentation algorithms on a database of 4467 problems of size varying from $n = 25$ to 153662 data points. These data come from different microarrays data sets (Affymetrix, Nimblegen, BAC/PAC) and different tumour types (leukaemia, lymphoma, neuroblastoma, medulloblastoma).

We compared FPOP to several other segmentation algorithms: pDPA (Rigaill, 2015), PELT (Killick et al., 2012), Binary Segmentation (BinSeg), Wild Binary Segmentation (WBS; Fryzlewicz, 2014), and SMUCE (Frick et al., 2014). We ran pDPA and BinSeg with a maximum number of changes $K = 52$, WBS and SMUCE with default settings, and PELT and FPOP with the SIC penalty. The SIC is used as it is a natural choice with no prior information and also it performs well in terms of the accuracy of the segmentation (this will be shown later in Section 3.7.4).

We used the R `microbenchmark` package to measure the execution time on each of the 4467 segmentation problems. The R source code for these timings is in `benchmark/systemtime.arrays.R` in the opfp project repository on R-Forge: `https://r-forge.r-project.org/R/?group_id=1851`

Figure 3.6 shows that the speed of FPOP is comparable to BinSeg, and faster than the other algorithms. As expected, it is clear that the asymptotic behavior of FPOP is similar to pDPA for a large number of data points to segments. Note that for analysing a single data set, WBS could be more easily implemented in parallelised computing environment that the other methods. If done so this would lead to some reduction in it computational cost per data set. For analysing multiple data sets, as here, all methods are trivially parallelisable through analysing each data set on a different CPU.

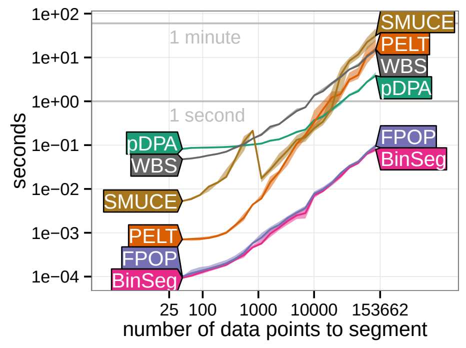
3.7.2 Speed benchmark: simulated data with different number of changes

The speed of PELT, BinSeg and pDPA depends on the underlying number of changes. For pDPA and BinSeg the relationship is clear; to cope with a larger number of changes, one needs to increase the maximum number of changes K. For a signal of fixed size n, the time complexity is expected to be $\mathcal{O}(\log K)$ for BinSeg and $\mathcal{O}(K)$ for pDPA (Rigaill, 2015).

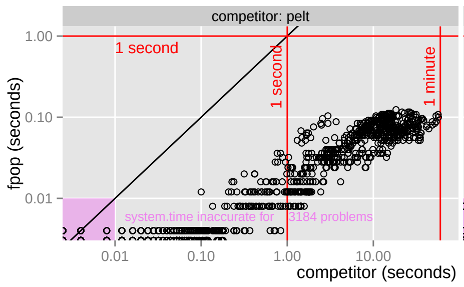
For PELT the expected time complexity is not as clear, but pruning should be more efficient if there are many changepoints. Hence for a signal of fixed size n, we expect the runtime of PELT to decrease with the underlying number of changes.

Based on Section 3.6, we expect FPOP to be faster than PELT and pDPA. Thus it seems reasonable to expect FPOP to faster for the whole range of K. This is what we empirically check in this section.

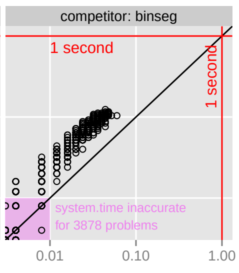
To do that we simulated a Gaussian signal with $n = 2 \times 10^5$ data points, and varied the number of changes K. We then repeat the same experiment for signals with $n = 10^7$ and timed FPOP and BinSeg only. The R source code for these timings is in `benchmark/systemtime.simulation.R` in the



(a)



(b)



(c)

Figure 3.6: Timings on the tumour microarray benchmark. (a) Runtimes as a function of the length n of the profile (median line and quartile error band). (b) Runtimes of PELT and FPOP for the same profiles. (c) Runtimes of BinSeg and FPOP for the same profiles. (Pink regions on plots (b) and (c) indicate the regions for which the `R` function `system.time` is inaccurate.)

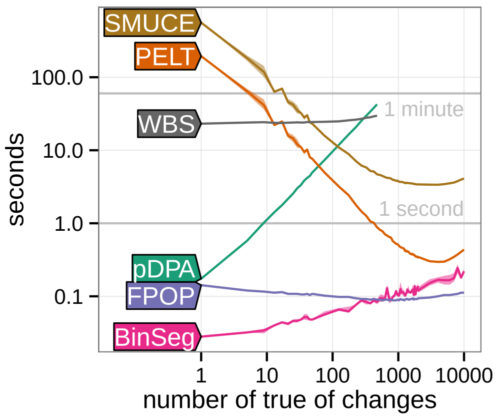
opfp project repository on R-Forge: `https://r-forge.r-project.org/R/?group_id=1851`.

It can be seen in Figure 3.7 that FPOP is always faster than pDPA, PELT, WBS, and SMUCE. Interestingly for both $n = 2 \times 10^5$ and $n = 10^7$, FPOP is faster than BinSeg for a true number of changepoints larger than $K = 500$. This is due to the computational time of FPOP remaining roughly constant as n increases, whereas BinSeg loses efficiency the more changepoints that are added.

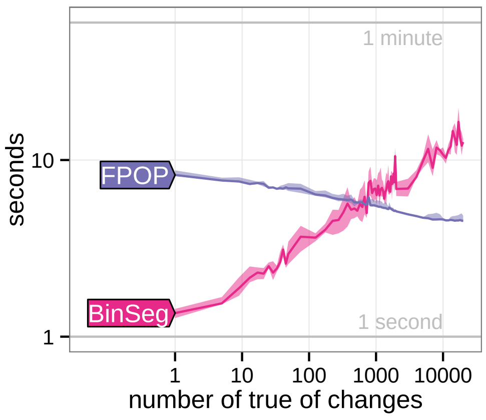
3.7.3 Accuracy benchmark: the neuroblastoma data set

Hocking et al. (2013) proposed the neuroblastoma tumour microarray data set for benchmarking changepoint detection accuracy of segmentation models. These data consist of annotated region labels defined by expert doctors when they visually inspected scatterplots of the data. There are 2845 negative labels where there should be no changes (a false positive occurs if an algorithm predicts a change), and 573 positive labels where there should be at least one change (a false negative occurs if an algorithm predicts no changes). There are 575 copy number microarrays, and a total of 3418 labeled chromosomes (separate segmentation problems).

Let m be the number of segmentation problems in the training set, let $n_1, \dots, n_m$ be the number of data points to segment in each problem, and let $\mathbf{y}^1 \in \mathbb{R}^{n_1}, \dots, \mathbf{y}^m \in \mathbb{R}^{n_m}$ be the vectors of noisy data to segment. Both PELT and pDPA have been applied to this benchmark by first defining a penalty value



(a)



(b)

Figure 3.7: Runtimes in simulated data sets with a variable number of true changepoints (median line and quartile error band). (a) All algorithms in data of size $n = 2 \times 10^5$. (b) BinSeg and FPOP in data of size $n = 10^7$.

of $\beta = \lambda n_i$ in (3.5) for all problems $i \in \{1, \dots, m\}$, and then choosing the constant $\lambda \in \{10^{-8}, \dots, 10^1\}$ that minimises the number of incorrect labels in the training set. To apply this model selection criterion to WBS and SMUCE, we first computed a sequence of models with up to $K = 20$ segments (for WBS we used the `changepoints.sbs` function, and for SMUCE we varied the `q` parameter).

First, we computed training error ROC curves by considering the entire database as a training set, and computing false positive and true positive rates for each penalty λ parameter (Figure 3.8(a)). The ROC curves suggest that FPOP, PELT, pDPA, and BinSeg have the best detection accuracy, followed by SMUCE, and then WBS.

Second, we performed cross-validation to estimate the test error of each algorithm. We divided the labeled segmentation problems into six folds. For each fold we designate it as a test set, and use the other five folds as a training set. For each algorithm we used grid search to choose the penalty λ parameter which had the minimum number of incorrect labels in the training set. We then count the number of incorrect labels on the test set. In agreement with the ROC curves, FPOP/pDPA/PELT/BinSeg had the smallest test error (2.2%), followed by SMUCE (2.43%), and then WBS (3.87%). Using a paired one-sided t_5-test, FPOP had significantly less test error than WBS ($p = 0.005$) but not SMUCE ($p = 0.061$).

3.7.4 Accuracy on the WBS simulation benchmark

We assessed the performance of FPOP using the simulation benchmark proposed in the WBS paper (Fryzlewicz, 2014) page 29. In that paper 5 scenarios are considered. We considered an additional scenario from a further paper on SMUCE (Futschik et al., 2014) corresponding to Scenario 2 of WBS with a standard deviation of 0.2 rather than 0.3. We call this Scenario 2'. Full details of all 6 scenarios are given in Appendix A.1.

We first compared FPOP with the BIC ($\beta = 2\log(n)$), WBS with the sSIC (strengthened Schwarz Information Criterion as defined in Fryzlewicz (2014)) and SMUCE with $\alpha = 0.45$ (used in Futschik et al. (2014) for scenario 2') in terms of mean squared error (MSE). For FPOP we first standardised the signal using the MAD (Median Absolute Deviation) estimate as was done for PELT in Fryzlewicz (2014).

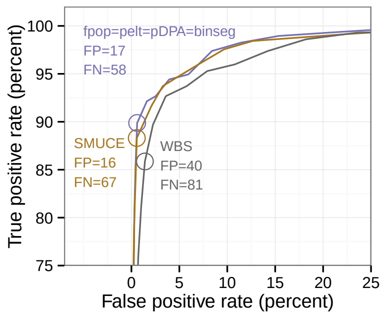
Using 2000 replications per scenario we tested the hypotheses

- H_0: the average MSE difference between WBS and FPOP is lower or equal to 0.
- H_1: the average MSE difference between WBS and FPOP is larger than 0.

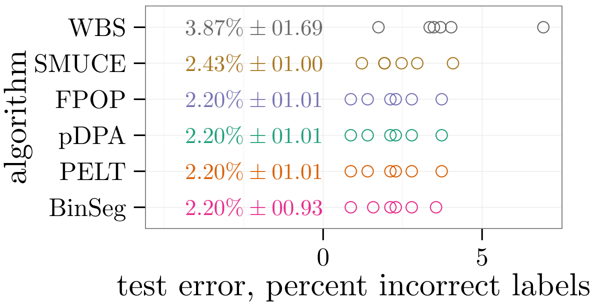
using a paired t-test and paired Wilcoxon test. H_0 is clearly rejected (p-value $< 10^{-16}$) in 4 scenarios out of the 6 (1, 2, 2' and 5). We did the same thing with SMUCE and we found that H_0 is rejected in 4 scenarios (1, 2, 4 and 5). The R code of this comparison is available on R-Forge.

More generally, we compared WBS with the sSIC, mBIC and BIC penalty, SMUCE with $\alpha = 0.35$, 0.45 and 0.55 and FPOP with $\beta = \log(n)$, $2\log(n)$ and $3\log(n)$. For each scenario we made 500 replications. We assessed the ability to recover the true number of changes $\hat{K}$, computed the mean squared error (MSE) and breakpoint error (BkpEr) from the `breakpointError` R package and counted the number of exactly recovered breakpoints (exact TP). With $\beta = 2\log(n)$ or $3\log(n)$ FPOP gets better results, in terms of MSE, $\hat{K}$, exact TP and BkpEr, than SMUCE and WBS in Scenarios 1 and 5. WBS is better than FPOP and SMUCE in Scenario 4. In Scenarios 2 and 3 WBS and FPOP are comparable (WBS is better in terms of BkpEr and worst in terms of MSE). In Scenario 2' FPOP and SMUCE are comparable. The average of each approach is given in Appendix A.2, and the R code is available on R-Forge in the "benchmark wbs" directory.

We performed similar analysis on our speed benchmark (Figure 3.7(a)) and found that FPOP is



(a)



(b)

Figure 3.8: Accuracy results on the neuroblastoma data set. (a) Training error ROC curves computed by varying the penalty λ on the entire data set. Circles and text indicate the penalty λ which minimized the number of incorrect labels (FP: false positive, FN: false negative). (b) Test error (circles: 6 test folds; text: mean and standard deviation).

competitive or better than WBS and SMUCE in terms of MSE, BkpEr, exact TP and $\hat{K}$. Results are shown in Appendix A.3. The R codes are also available on R-forge.

3.8 Discussion

We have introduced two new algorithms for detecting changepoints, FPOP and SNIP. A natural question is which of these, and the existing algorithms, pDPA and PELT, should be used in which applications. There are two stages to answering this question. The first is whether to detect changepoints through solving the constrained or the penalised optimisation problem, and the second is whether to use functional or inequality based pruning.

The advantage of solving the constrained optimisation problem is that this gives exact segmentations for a range of numbers of changepoints. The disadvantage is that solving it is slower than solving the penalised optimisation problem, particularly if there are many changepoints. In interactive situations where you wish to explore segmentations of the data, then solving the constrained problem is to be preferred (Hocking et al., 2014). However in non-interactive scenarios when the penalty parameter is known in advance, it will be faster to solve the penalised problem to recover the single segmentation of interest. Further, recent work in Haynes et al. (2015) explores a way of outputting multiple segmentations (corresponding to various penalty values) for the penalised problem.

The decision as to which pruning method to use is purely one of computational efficiency. We have shown that functional pruning always prunes more than inequality based pruning, and empirically have seen that this difference can be large, particularly if there are few changepoints. However functional pruning can be applied less widely. Not only does it require a stronger condition on the cost functions, but currently its implementation has been restricted to detecting changes in a univariate parameter from a model in the exponential family. Even for situations where functional pruning can be applied, its computational overhead per non-pruned candidate is higher.

Our experience suggests that you should prefer functional pruning in the situations where it can be applied. For example FPOP was always faster than PELT for detecting a change in mean in the empirical studies we conducted, the difference in speed is particularly large in situations where there are few changepoints. Furthermore we observed FPOP's computational speed was robust to changes in the number of changepoints to be detected, and was even competitive with, and sometimes faster than, Binary Segmentation.

Algorithm 2: Functional Pruning Optimal Partitioning (FPOP)

```
Input  : Set of data of the form $\mathbf{y}_{1:n} = (y_1, \ldots, y_n)$,
         A measure of fit $\gamma(\cdot,\cdot)$ dependent on the data and the mean,
         A penalty $\beta$ which does not depend on the number or location of the changepoints.
Let $n$ =length of data, and set $F(0) = -\beta$, $cp(0) = 0$;
then let $R_1 = \{0\}$;
and set $D =$ the range of $\mu$;
$Set_0^0 = D$;
$Cost_0^0(\mu) = F(0) + \beta = 0$;
for $t = 1, \ldots, n$ do
    for $\tau \in R_t$ do
        $Cost_t^\tau(\mu) = Cost_{t-1}^\tau(\mu) + \gamma(y_t, \mu)$;
    Calculate $F(t) = \min_{\tau \in R_t}(\min_{\mu \in Set_t^\tau}[Cost_t^\tau(\mu)])$;
    Let $\tau_t = \arg\min_{\tau \in R_t}(\min_{\mu \in Set_t^\tau}[Cost_t^\tau(\mu)])$;
    Set $cp(t) = (cp(\tau_t), \tau_t)$;
    $Cost_t^t(\mu) = F(t) + \beta$;
    $Set_t^t = D$;
    for $\tau \in R_t$ do
        $I_t^\tau = \{\mu : Cost_t^\tau(\mu) \leq F(t) + \beta\}$;
        $Set_t^\tau = Set_{t-1}^\tau \cap I_t^\tau$;
        $Set_t^t = Set_t^t \backslash I_t^\tau$;
    $R_{t+1} = \{\tau \in \{R_t \cup \{t\}\} : Set_t^\tau \neq \emptyset\}$;
Output: The changepoints recorded in $cp(n)$.
```

```
Algorithm 3: Segment Neighbourhood with Inequality Pruning (SNIP)
Input  : Set of data of the form $\mathbf{y}_{1:n} = (y_1, \dots, y_n)$,
         A measure of fit $\mathcal{C}(\cdot)$ dependent on the data (needs to be minimised),
         An integer, $K$, specifying the maximum number of changepoints to find,
         A constant $\kappa$ that satisfies: $\mathcal{C}(\mathbf{y}_{s+1:t}) + \mathcal{C}(\mathbf{y}_{t+1:T}) + \kappa \leq \mathcal{C}(\mathbf{y}_{s+1:T})$.
Let $n$ =length of data;
Set $C_{0,t} = \mathcal{C}(\mathbf{y}_{1:t})$, for all $t \in \{1, \dots, n\}$;
for $k = 1, \dots, K$ do
    Set $R_{k,k+1} = \{k\}$. for $t = k+1, \dots, n$ do
        Calculate $C_{k,t} = \min_{v \in R_{k,t}} (C_{k-1,v} + \mathcal{C}(\mathbf{y}_{v+1:t}))$;
        Set $R_{k,t+1} = \{v \in \{R_{k,t} \cup \{t\}\} : C_{k-1,v} + \mathcal{C}(\mathbf{y}_{v+1,t}) + \kappa < C_{k-1,t}\}$;
    Set $\tau_{k,1} = \arg\min_{v \in R_{k,n}} (C_{k-1,v} + \mathcal{C}(\mathbf{y}_{v+1,n}))$;
    for $i = 2, \dots, k$ do
        Let $\tau_{k,i} = \arg\min_{v \in R_{k-i,\tau_{k,i-1}}} (C_{k-1,v} + \mathcal{C}(\mathbf{y}_{v+1,\tau_{k,i-1}}))$;
Output: For $k = 0, \dots, K$: the total measure of fit, $C_{k,n}$, for $k$ changepoints and the location
        of the changepoints for that fit, $\boldsymbol{\tau}_{k,(1:k)}$.
```

Chapter 4

Optimal Changepoint Detection for Piecewise Linear Data

4.1 Introduction

In Chapter 3 we considered changepoint detection methods which can be applied, for example, to detect abrupt changes in mean or variance. These changepoint problems had the property that the model for the data after a changepoint did not depend in any way on the data prior to the changepoint. So for the change in mean problem, the segment mean after a changepoint is not constrained by or dependent on the mean for the segment prior to that changepoint. This property was central to the derivation of the dynamic programming recursions that we used. In this chapter we consider a problem where this property does not hold.

We consider the problem of fitting a piecewise linear function to data, subject to the constraint that the function is continuous. An example of such a fit to data is given in Figure 4.1. For this problem the changepoints correspond to points in time where the slope of the linear function changes. The model for the data before the change is dependent on the model for the data after the change, so we will need to develop new methods to deal with this.

There are a wide-range of applications where such a function has been fit to the data, for example cancer rates (Kim et al., 2009), climate data (Tomé and Miranda, 2004) and acoustic data (Horner and Beauchamp, 1996). In particular we are motivated by the problem of modelling how the rotation of a

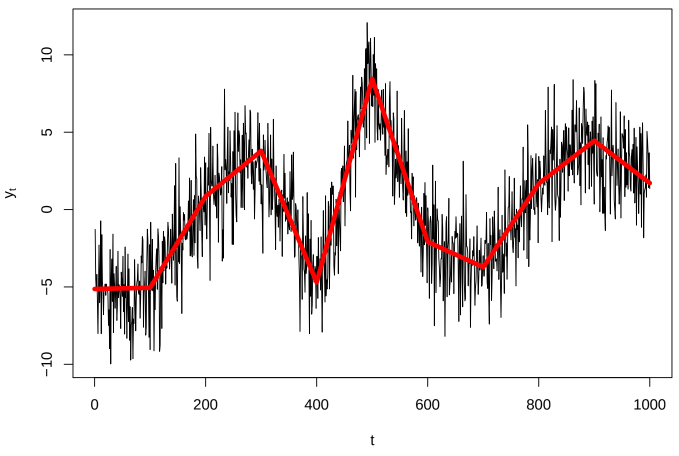


Figure 4.1: Example of piecewise linear data from the model given in (4.2).

bacterial flagellar motor (BFM) changes over time as discussed in (Sowa and Berry, 2008). The rotation of the BFM is such that it has periods with little movement interspersed by time periods where the angle changes rapidly. A subsection of such a data set is shown in Figure 4.2. It is natural to model this data using a changepoint model, with a constant angular velocity model within segments. This would correspond to fitting a linear function within each segment. However the context of the problem enforces that we would want the resulting piecewise linear function to be continuous. Previous analysis of the rotation of the BFMs has ignored this continuity (Weinmann and Storath, 2015; Sowa et al., 2005) because of the difficulty of finding the best continuous piecewise linear function that minimises a square error loss. In fact Weinmann and Storath (2015) claim finding such a best fitting function is an NP-hard problem.

A few papers consider the problem of fitting a continuous piecewise linear function. A single change-point in a piecewise linear function is considered in Julious (2001). Here a F-test is considered to detect a change in the model parameters, however the author notes that this only conforms to its expected parametric distribution when the changepoint location is known. To deal with this issue the authors suggests using a nonparametric bootstrap to assess the changepoint models.

Multiple changepoint detection in piecewise linear models in cancer rate data is looked at in Kim et al. (2000). They fit a piecewise linear model searching the solution space using a grid search method. This works by forcing continuity at the changepoints, but only searching over a discrete uniform set of points both for the location of the change and the value at it. This makes the method only approximate

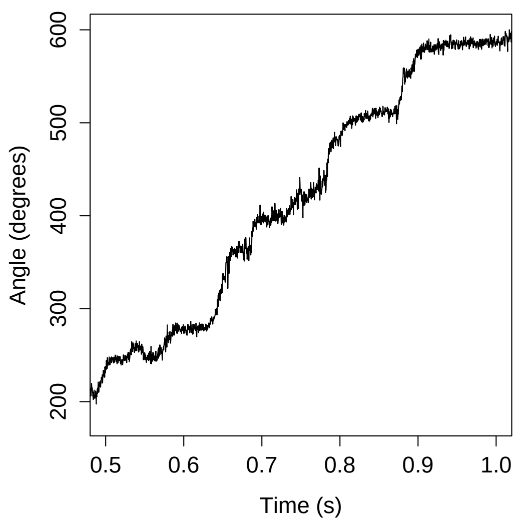


Figure 4.2: Subsection of data showing the rotation of a bacterial flagellar motor over time (Sowa et al., 2005).

as they do not allow changes at all data points. A similar method is used in Goldberg et al. (2014), however they suggest adapting the grid used, making it non-uniform to reflect the particular data set (data on petrol consumption in cars in their paper).

Piecewise linear models with unknown changepoints are fitted to climate data in Tomé and Miranda (2004). They apply their method to three data sets; wave height data, maximum temperature data and precipitation level data. The minimisation problem is solved optimally using an exhaustive search, constraining on the minimum distance between changepoints and the minimum trend change. This is very expensive computationally, but the authors only allow for a small number of changepoints. Even so the method is only feasible for small data sets, and Tomé and Miranda (2004) only apply the method to data with around 200 data points.

A further approach for fitting a piecewise linear model with continuity at the changepoints is used in Horner and Beauchamp (1996) on acoustic data. They consider multivariate data corresponding to different harmonics of the instrument allowing the existence of shared changepoints across the signals. They fit the piecewise linear function using a genetic algorithm to solve the minimisation problem, however this does not necessarily find the best fit.

The l_1 Trend Filtering method, introduced by Kim et al. (2009), can be used to fit a continuous piecewise linear function with unknown changepoints. Trend filtering aims to minimise the sum of squares, whilst adding a penalty based on the l_1 norm to penalise variations in the estimated trend. This is looked at further in Tibshirani (2014). The minimisation can then be solved using an interior point method (Kim et al., 2009) or via a dual path algorithm (Arnold and Tibshirani, 2016). We show in Section 4.4.2 that this is a computationally efficient method that performs well at estimating the underlying function, but performs poorly in terms of estimating changepoints. It has a tendency to add multiple changepoints in regions where there is just a single change.

Current methods in the literature for detecting changes when fitting a continuous piecewise linear function are either heuristic (Kim et al., 2000, 2009; Goldberg et al., 2014; Horner and Beauchamp, 1996) or computationally time consuming (Tomé and Miranda, 2004). In this chapter we focus on developing a new technique which extends the current dynamic programming based approach to deal with fitting a piecewise linear model with continuity at the changepoints. This leads to the introduction of a new algorithm, Optimal Partitioning for Piecewise Linear data (OPPL), which is both exact and, surprisingly, scales well computationally for large data sets.

4.2 Model Definition

We assume that we have data ordered by time and denote this by $\mathbf{y} = (y_1, \ldots, y_n)$. We will also use the notation that for $t \geq s$ the set of observations from time s to time t is $\mathbf{y}_{s:t} = (y_s, \ldots, y_t)$. If we assume that there are m changepoints in the data, this will correspond to the data being split into $m+1$ distinct segments. We let the location of the jth changepoint be τ_j for $j = 1, \ldots, m$, and set $\tau_0 = 0$ and $\tau_{m+1} = n$. The jth segment will consist of data points $y_{\tau_{j-1}+1}, \ldots, y_{\tau_j}$. We let $\tau = (\tau_0, \ldots, \tau_{m+1})$ be the set of ordered changepoints.

We consider the case of fitting a continuous piecewise linear function to the data. An example of such a fit is given in Figure 4.1. For such a problem, changepoints will correspond to points in time where the slope of the function changes. We introduce segment-specific intercept and slope parameters respectively a_i and b_i for segment $i = 0, \ldots, m$. With the assumption of continuity requiring that the fitted value at the beginning of one segment is the same as the fitted value at the end of the previous one

$$\begin{aligned} Y_t &= a_i + b_i t + Z_t, \qquad \text{for } t = \tau_i, \ldots, \tau_{i+1}, \\ \text{s.t.} \quad a_i + b_i\tau_i &= a_{i-1} + b_{i-1}\tau_i, \end{aligned} \tag{4.1}$$

for fixed (but unknown) changepoints $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_m)$, unknown parameters a_i, b_i (for $i = 0, \ldots, m$) and zero mean noise, Z_t.

For our methods it is more convenient to write the dependence across segments as being conditional on a single parameter from each segment (rather than two as in the formulation above). With this in mind, in this chapter we will re-paramaterise (4.1) in terms of the fitted values at the changepoints, $\phi_{\tau_i} = a_i + b_i\tau_i$ for $i = 0, \ldots, m+1$. The model for each segment is then characterised by two parameters, namely the fitted value at each end of the segment. So for the $(i+1)$th segment, our model depends on ϕ_{τ_i} and $\phi_{\tau_{i+1}}$, with

$$Y_t = \phi_{\tau_i} + \tfrac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i}(t - \tau_i) + Z_t, \quad \text{for } t = \tau_i + 1, \ldots, \tau_{i+1}, \tag{4.2}$$

for unknown parameters ϕ_{τ_i} (fitted values at τ_i), for $i = 0, \ldots, m$.

As discussed in previous chapters, one way of detecting the unknown changepoint locations is to define a suitable cost function, that depends on the data and the changepoint parameters. Typically this cost function is defined per segment and summed over all segments, to get a measure of cost for the entire data set. We then minimise this cost using an appropriate algorithm (which could be either exact or heuristic in nature). Throughout we will let $\mathcal{C}(\mathbf{y}_{s+1:t}, \phi_s, \phi_t)$, for $s \leq t$, denote the cost for a segment consisting of data points $y_{s+1}, \ldots, y_t$, where ϕ_s and ϕ_t are the parameters associated with the fitted values at the start and end of the segment respectively. A natural choice is the square error loss

$$\mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) = \frac{1}{2\sigma^2} \sum_{j=\tau_i+1}^{\tau_{i+1}} \left(y_j - \phi_{\tau_i} - \frac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i}(j - \tau_i) \right)^2. \tag{4.3}$$

This corresponds to the cost being minus the log-likelihood for a model where we assume the errors, Z_t in (4.2), are normally distributed with variance σ^2. If σ is unknown then it needs to be estimated. One way to do this is to use the Median Absolute Deviation estimator (Hampel, 1974).

As in the independent segment case this minimisation can be done for either the constrained case (known number of changepoints) or the penalised case (unknown number of changepoints, where adding changepoints is penalised via a linear penalty). For this chapter we consider the penalised case. The minimisation problem we wish to solve is as follows

$$\min_{\boldsymbol{\tau},m,\boldsymbol{\phi}} \left[\sum_{i=0}^{m} \mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(m+1)\right], \tag{4.4}$$

where β is a penalty to stop overfitting.

Solving the minimisation problem in (4.4) by complete enumeration takes $\mathcal{O}(2^n)$ time and therefore is infeasible for large values of n. Below we propose a pruned dynamic programming approach to calculate the exact solution to (4.4) efficiently.

4.3 Optimal Partitioning for Piecewise Linear Data

We wish to minimise (4.4) exactly whilst keeping the computational cost at a minimum to enable the algorithm to be able to deal with large data sets efficiently. With this in mind, we suggest a dynamic programming based algorithm similar to that used in Optimal Partitioning where the segments are independent. We call this method Optimal Partitioning for Piecewise Linear data (OPPL).

When the model is assumed to have independent segments then, given knowledge of a changepoint at time τ, the data can be segmented separately before and after the change. With our model this is no longer the case and to segment separately before and after a change we also require the fitted value at the change, ϕ_τ.

We will introduce a dynamic programming algorithm which conditions on the location of the last changepoint and the value of the function at the last changepoint. We explain why a naive implementation of this algorithm is not viable before introducing two different methods for speeding the algorithm up by pruning the search space. The pseudocode for the OPPL method is given in Algorithm 4.

4.3.1 Dynamic Programming Approach

We will consider segmenting the data up to time t, that is $\mathbf{y}_{1:t}$, for $t = 1, \ldots, n$. When segmenting $\mathbf{y}_{1:t}$ with k changepoints, $\tau_1, \ldots, \tau_k$, we also use the notation $\tau_0 = 0$ and $\tau_{k+1} = t$. We define the function $f^t(\phi)$ to be the minimum penalised cost for segmenting $\mathbf{y}_{1:t}$ conditional on $\phi_t = \phi$, that is the fitted value at time t is ϕ. Formally this is defined as

$$f^t(\phi) = \min_{\boldsymbol{\tau},k,\boldsymbol{\phi}} \left[\sum_{i=0}^{k} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(k+1)\right], \tag{4.5}$$

$$\text{s.t.} \quad \phi_t = \phi.$$

By manipulation of (4.5), and using the initial condition that $f^0(\phi) = 0$, we can construct a dynamic programming recursion for $f^t(\phi)$ based on minimising over the position of the last changepoint, which we will denote s.

$$\begin{aligned}
f^t(\phi) &= \min_{\boldsymbol{\tau},k,\boldsymbol{\phi}:\phi_t=\phi} \left[\sum_{i=0}^{k} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(k+1)\right], \\
&= \min_{\boldsymbol{\tau},k,\boldsymbol{\phi}:\phi_t=\phi} \left[\sum_{i=0}^{k-1} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta k + \mathcal{C}(y_{\tau_k+1:t}, \phi_{\tau_k}, \phi_t) + \beta\right], \\
&= \min_{\phi',s} \left\{ \min_{\boldsymbol{\tau}',k',\boldsymbol{\phi}:\phi_s=\phi'} \left[\sum_{i=0}^{k'} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(k'+1)\right] + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + \beta \right\}, \\
&= \min_{\phi',s} \left\{ f^s(\phi') + \mathcal{C}(y_{s+1:t}, \phi', \phi) + \beta \right\}.
\end{aligned}$$

The idea is that we split the minimisation into first minimising over the time of the most recent changepoint and the fitted value at that changepoint, and minimising over the earlier changepoints and fitted values. On the third line we have let s denote the time of the most recent changepoint, and ϕ' the fitted value at s. The inner minimisation is over the number k', the locations of the changepoints prior to s, $\boldsymbol{\tau}'$, and the fitted values at the changepoints prior to s, $\boldsymbol{\phi}$. This inner minimisation gives the minimum penalised cost for segmenting $\mathbf{y}_{1:s}$ conditional on $\phi_s = \phi'$, which is $f^s(\phi')$. This recursion is similar to that derived for Optimal Partitioning. However for Optimal Partitioning we just needed to store a scalar value for each $t = 1, \dots n$. Here we need to store functions of the continuous parameter ϕ for each value of t.

To store $f^t(\phi)$ we will write it as a piecewise minimum of the cost functions defined over ϕ for a given vectors of changepoints $\boldsymbol{\tau}$. We define each of these functions $f^t_{\boldsymbol{\tau}}(\phi)$ as the minimum cost of segmenting

$\mathbf{y}_{1:t}$ with changepoints at $\boldsymbol{\tau} = \tau_1, \ldots, \tau_k$ and fitted value $\phi_t = \phi$ at time t

$$f^t_{\boldsymbol{\tau}}(\phi) = \min_{\boldsymbol{\phi}} \left[\sum_{i=0}^{k} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(k+1) \right], \quad \text{s.t.} \quad \phi_t = \phi. \tag{4.6}$$

Then $f^t(\phi)$ is the piecewise minimum of these functions where, for a given value of ϕ

$$f^t(\phi) = \min_{\boldsymbol{\tau} \in \mathcal{T}_t} f^t_{\boldsymbol{\tau}}(\phi), \tag{4.7}$$

where we define $\mathcal{T}_t$ as the set of all possible changepoint vectors at time t.

By manipulation of (4.6) we can get a recursion for $f^t_{\boldsymbol{\tau}}(\phi)$ which will enable us to update these functions at each time point

$$\begin{aligned} f^t_{\boldsymbol{\tau}}(\phi) &= \min_{\boldsymbol{\phi}:\phi_t=\phi} \left[\sum_{i=0}^{k} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(k+1) \right], \\ &= \min_{\boldsymbol{\phi}:\phi_t=\phi} \left[\sum_{i=0}^{k-1} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta k + \mathcal{C}(y_{\tau_k+1:t}, \phi_{\tau_k}, \phi_t) + \beta \right], \\ &= \min_{\phi'} \left\{ \min_{\boldsymbol{\phi}:\phi_{\tau_k}=\phi'} \left[\sum_{i=0}^{k-1} \mathcal{C}(y_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta k \right] + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + \beta \right\}. \end{aligned}$$

Noting that the first half of this expression is just the optimal segmentation up to time τ_k with changepoints at $\boldsymbol{\tau} = \tau_1, \ldots, \tau_{k-1}$ and the condition that $\phi_{\tau_k} = \phi'$. This gives the following recursion

$$f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'} \left\{ f^{\tau_k}_{\tau_1,\ldots,\tau_{k-1}}(\phi') + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + \beta \right\}. \tag{4.8}$$

These functions can be stored easily as each one is just a quadratic and hence we can just store the coefficients, we explain how to update these quadratic coefficients in Appendix B. Therefore we can iteratively compute these functions and thus calculate $f^n(\phi)$. We then calculate the optimal segmentation of $\mathbf{y}_{1:n}$ by minimising $f^n(\phi)$ over ϕ. The value of $\boldsymbol{\tau}$ that achieves the minimum value will be the optimal segmentation. This approach, however, is computationally expensive; both in time, $\mathcal{O}(n2^n)$, and space needed to store the functions, $\mathcal{O}(2^n)$. A more efficient algorithm can be constructed by pruning changepoint vectors from the search space as they become redundant. There are two ways in which this can be achieved; functional and inequality based pruning.

4.3.2 Functional Pruning

Our algorithm relies on computing and storing cost functions for each possible changepoint vector, $\boldsymbol{\tau}$. This is computationally expensive and so we utilise pruning methods to discard potential changepoint vectors which can be shown to never be optimal. This will result in less functions to be updated at each time step increasing the overall efficiency of the algorithm.

One way we can prune these candidate changepoint vectors from the minimisation problem is when they can be shown to be dominated by other vectors for any given value of ϕ. Similar approaches are found in Rigaill (2015) and Maidstone et al. (2016) for independent segment models and is known as *functional pruning.*

In Theorem 3 we show how if a candidate changepoint vector, $\boldsymbol{\tau}$ is not optimal at time s for any value of ϕ, then the related candidate changepoint vector $(\boldsymbol{\tau}, s)$ (the concatenation of $\boldsymbol{\tau}$ and s) is not optimal for any value of ϕ at time t where $t > s$. If this is the case, the vector $(\boldsymbol{\tau}, s)$ can be pruned from the candidate changepoint set.

First we define the set $\overset{*}{\mathcal{T}}_t$ as the set of changepoint vectors that are optimal for some ϕ at time t

$$\overset{*}{\mathcal{T}}_t = \left\{ \boldsymbol{\tau} \in \mathcal{T}_t : f^t(\phi) = f^t_{\boldsymbol{\tau}}(\phi), \text{ for some } \phi \in (-\infty, \infty) \right\}, \tag{4.9}$$

where $\mathcal{T}_t$ is the set of all possible changepoint vectors at time t. If a candidate vector $\boldsymbol{\tau}$ is not in this set at time s then the related candidate vector $(\boldsymbol{\tau}, s)$ is not in the set at time t. This means that at time t we will need to store only the functions $f^t_{\boldsymbol{\tau}}(\phi)$ corresponding to segmentations that are in $\overset{*}{\mathcal{T}}_t$. Empirically we show in Section 4.4.1 that, for most data sets, the size of $\overset{*}{\mathcal{T}}_t$ remains roughly constant as we increase t.

Theorem 3 *If $\boldsymbol{\tau} \notin \overset{*}{\mathcal{T}}_s$ then $(\boldsymbol{\tau}, s) \notin \overset{*}{\mathcal{T}}_t$ for all $t > s$.*

The proof of Theorem 3 works by contrapositive. We show that if $(\boldsymbol{\tau}, s) \in \overset{*}{\mathcal{T}}_t$ then a necessary condition of this is that $\boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s$, taking the contrapositive of this gives Theorem 3.

Proof. Assume $(\boldsymbol{\tau}, s) \in \overset{*}{\mathcal{T}}_t$, then there exists ϕ such that

$$f^t(\phi) = f^t_{(\boldsymbol{\tau},s)}(\phi),$$

Now for any ϕ^*,

$$\begin{aligned} f^s(\phi^*) + \mathcal{C}(y_{s+1:t}, \phi^*, \phi) + \beta &\geq \min_{\phi', r} \left[f^r(\phi') + \mathcal{C}(y_{r+1:t}, \phi', \phi) + \beta \right], \\ &= f^t(\phi), \\ &= f^t_{(\boldsymbol{\tau}, s)}(\phi), \\ &= \min_{\phi''} \left\{ f^s_{\boldsymbol{\tau}}(\phi'') + \mathcal{C}(y_{s+1:t}, \phi'', \phi) + \beta \right\}, \qquad (4.10) \\ &= f^s_{\boldsymbol{\tau}}(\phi^A) + \mathcal{C}(y_{s+1:t}, \phi^A, \phi) + \beta, \end{aligned}$$

where ϕ^A is the value of ϕ'' which minimises (4.10). As ϕ^* can be chosen as any value, we can choose it as ϕ^A. By cancelling terms we get $f^s(\phi^A) \geq f^s_{\boldsymbol{\tau}}(\phi^A)$ and hence (from (4.7)), $f^s(\phi^A) = f^s_{\boldsymbol{\tau}}(\phi^A)$ and therefore $\boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s$. We have shown that if $(\boldsymbol{\tau}, s) \in \overset{*}{\mathcal{T}}_t$ then $\boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s$, by taking the contrapositive the theorem holds. ■

The key to an efficient algorithm will be a way of efficiently calculating $\overset{*}{\mathcal{T}}_t$. We can use the above theorem to help us do this. From Theorem 3 we can define a set

$$\hat{\mathcal{T}}_t = \left\{ (\boldsymbol{\tau}, s) : s \in \{0, \ldots, t-1\}, \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s \right\}, \qquad (4.11)$$

and we will have that $\hat{\mathcal{T}}_t \supseteq \overset{*}{\mathcal{T}}_t$. So assume that we have calculated the sets $\overset{*}{\mathcal{T}}_s$ for $s = 0, \ldots, t-1$. We can calculate $f^t_{\boldsymbol{\tau}}(\phi)$ only for $\boldsymbol{\tau} \in \hat{\mathcal{T}}$. When calculating $f^t(\phi)$, as defined by (4.7), we can just minimise over the set of changepoint vectors in $\hat{\mathcal{T}}_t$ rather than the full set. Furthermore we can calculate which of the sets of changepoints in $\hat{\mathcal{T}}_t$ contribute to this minimum and remove those that do not contribute. The remaining sets of changepoints define $\overset{*}{\mathcal{T}}_t$.

To find out which sets of changepoints, $\boldsymbol{\tau}$, contribute to the minimisation of (4.7) we store the interval (or set of intervals) of the parameter space for which it is optimal. We define this interval as follows

$$Int^t_{\boldsymbol{\tau}} = \left\{ \phi : f^t_{\boldsymbol{\tau}}(\phi) = \min_{\boldsymbol{\tau}' \in \hat{\mathcal{T}}_t} f^t_{\boldsymbol{\tau}'}(\phi) \right\}. \qquad (4.12)$$

For a given t the union of these intervals over $\boldsymbol{\tau}$ is just the real line (as for a given ϕ at least one changepoint vector $\boldsymbol{\tau}$ corresponds to the optimal segmentation). Using this we can derive a simple algorithm for updating these intervals. We initialise the algorithm by setting the current parameter value as $\phi_{curr} = -\infty$ and comparing the cost functions in our current set of candidates (which we initialise as $\mathcal{T}_{temp} = \hat{\mathcal{T}}_t$) to get the optimal segmentation for this value, $\boldsymbol{\tau}_{curr}$. For each $\boldsymbol{\tau} \in \mathcal{T}_{curr}$ we calculate where $f^t_{\boldsymbol{\tau}}$ next intercepts with $f^t_{\boldsymbol{\tau}_{curr}}$ (smallest value of ϕ for which $f^t_{\boldsymbol{\tau}}(\phi) = f^t_{\boldsymbol{\tau}_{curr}}(\phi)$ and $\phi > \phi_{curr}$) and store this as $x_{\boldsymbol{\tau}}$. If for a $\boldsymbol{\tau} \in \mathcal{T}_{temp}$ we have $x_{\boldsymbol{\tau}} = \emptyset$ (i.e. $f^t_{\boldsymbol{\tau}}$ doesn't intercept with $f^t_{\boldsymbol{\tau}_{curr}}$ for any $\phi > \phi_{curr}$) then we remove $\boldsymbol{\tau}$ from $\mathcal{T}_{temp}$. We take the minimum of $x_{\boldsymbol{\tau}}$ (the first of the intercepts) and set it as our new ϕ_{curr} and the corresponding changepoint vector that produces it as $\boldsymbol{\tau}_{curr}$. We repeat this procedure until the set $\mathcal{T}_{temp}$ consists of only a single value $\boldsymbol{\tau}_{curr}$ which is the optimal segmentation for all future $\phi > \phi_{curr}$. This method is given in full in Algorithm 5.

Having calculated $Int^t_{\boldsymbol{\tau}}$ for all $\boldsymbol{\tau} \in \hat{\mathcal{T}}$ we can use these to calculate $\overset{*}{\mathcal{T}}$. We remove $\boldsymbol{\tau}$ from $\hat{\mathcal{T}}$ if $Int^t_{\boldsymbol{\tau}} = \emptyset$ and after doing this for all $\boldsymbol{\tau} \in \hat{\mathcal{T}}$ we are left with precisely those values of $\boldsymbol{\tau}$ which make up $\overset{*}{\mathcal{T}}$. This is used to recursively calculate $\hat{\mathcal{T}}_{t+1}$

$$\hat{\mathcal{T}}_{t+1} = \hat{\mathcal{T}}_t \cup \left\{ (\boldsymbol{\tau}, t) : \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_t \right\}. \tag{4.13}$$

The computational cost for updating the intervals $Int^t_{\boldsymbol{\tau}}$ for all $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ depends on the number of elements in $\hat{\mathcal{T}}_t$ and the number of intervals which the real line is partitioned into by $Int^t_{\boldsymbol{\tau}}$. The size of this partitioning is bounded above by $2|\overset{*}{\mathcal{T}}_t|(|\overset{*}{\mathcal{T}}_t| - 1) + 1$ (as $|\overset{*}{\mathcal{T}}_t|$ is the number of quadratics that partition the space and the bound is given by the number of possible intersections of these quadratics). Further the size of set $\hat{\mathcal{T}}_t$ is t times the size of $\overset{*}{\mathcal{T}}_{t-1}$ (from the definition of $\hat{\mathcal{T}}_t$ in equation (4.11)) and so the overall computational cost for updating the intervals is bounded by $\mathcal{O}(|\overset{*}{\mathcal{T}}_t|^2 \times t|\overset{*}{\mathcal{T}}_{t-1}|)$.

In Section 4.4.1 we will show that, in practice, $|\overset{*}{\mathcal{T}}_t|$ is roughly constant. This means that empirically the computational cost for updating $Int^t_{\boldsymbol{\tau}}$ for all $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ is $\mathcal{O}(n)$.

4.3.3 Inequality Based Pruning

A further way pruning can be used to speed up the dynamic programming algorithm is by applying *inequality based pruning.* This is used in Killick et al. (2012) for models with independent segments, and we use similar arguments here.

The result needed to do this additional pruning is given in Theorem 4. For a candidate changepoint vector, if the best cost at time t is worse than the best cost over all changepoint vectors plus twice our penalty term, β, we can show that the candidate is sub-optimal at all future times as well.

Theorem 4 *If for some* $\boldsymbol{\tau}$*,*

$$\min_{\phi} f^t_{\boldsymbol{\tau}}(\phi) > \min_{\phi'} \left[f^t(\phi') \right] + 2\beta, \tag{4.14}$$

holds, then at any future time $T > t + 1$, the set of changepoints $\boldsymbol{\tau}$ can never be optimal for the data $y_{1:T}$.

The proof for Theorem 4 follow a similar argument to the corresponding proof in Killick et al. (2012). However we have to add a segment consisting of the single point y_{t+1} to deal with the dependence between the segments.

Proof. Firstly note that by adding changes, at any point, without the penalty term and minimising over the corresponding ϕ values will also decrease the cost. Therefore we have

$$f^T_{\boldsymbol{\tau}}(\phi) \geq \min_{\phi',\phi''} \left[f^t_{\boldsymbol{\tau}}(\phi') + \mathcal{C}(y_{t+1}, \phi', \phi'') + \mathcal{C}(y_{t+2:T}, \phi'', \phi) \right]. \tag{4.15}$$

Then assuming that (4.14) is true, it can be shown that the segmenting the data $\mathbf{y}_{1:T}$ with changepoints $\boldsymbol{\tau}$ is always sub-optimal.

So from (4.15) and using (4.14)

$$\begin{aligned} f^T_{\boldsymbol{\tau}}(\phi) &\geq \min_{\phi',\phi''} \left[f^t_{\boldsymbol{\tau}}(\phi') + \mathcal{C}(y_{t+1}, \phi', \phi'') + \mathcal{C}(y_{t+2:T}, \phi'', \phi) \right], \\ &\geq \min_{\phi'} [f^t_{\boldsymbol{\tau}}(\phi')] + \min_{\phi',\phi''} [\mathcal{C}(y_{t+1}, \phi', \phi'') + \mathcal{C}(y_{t+2:T}, \phi'', \phi)], \\ &> \min_{\phi'} \left[f^t(\phi') \right] + 2\beta + \min_{\phi',\phi''} [\mathcal{C}(y_{t+1}, \phi', \phi'') + \mathcal{C}(y_{t+2:T}, \phi'', \phi)], \\ &= \min_{\phi',\phi''} \left[f^t(\phi') + \mathcal{C}(y_{t+1}, \phi', \phi'') + \mathcal{C}(y_{t+2:T}, \phi'', \phi) + 2\beta \right]. \end{aligned}$$

The last step is due to the cost on a single point, $\mathcal{C}(y_t, \phi', \phi'')$ only depending on ϕ''.

Therefore the cost of segmenting $\mathbf{y}_{1:T}$ with changepoints $\boldsymbol{\tau}$ is always greater than the cost of segmenting $\mathbf{y}_{1:T}$ with changepoints $(\boldsymbol{\tau}^*, t, t+1)$ (where $\boldsymbol{\tau}^*$ is the optimal segmentation of $\mathbf{y}_{1:t}$) and this holds for all $T > t + 1$ and hence $\boldsymbol{\tau}$ can be pruned. ■

The proof of Theorem 4 requires a minimum segment size of 2 to work and hence if equation (4.14) holds for some t we can only prune for $T > t + 1$.

In Section 4.3.2 we considered candidate changepoints vectors that belonged to the set $\hat{\mathcal{T}}_t$, and updated the related cost functions. We then used functional pruning to reduce this set to only those values that are optimal for some value of ϕ, namely the set $\overset{*}{\mathcal{T}}$. Using Theorem 4 we can reduce the size of $\hat{\mathcal{T}}_{t+1}$ before the cost functions are updated, discarding candidates from the set if (4.14) is true. As this reduces the size of the set $\hat{\mathcal{T}}_t$, it also reduces the computational cost of the algorithm.

Both pruning steps can be used to restrict the set of candidate changepoint vectors that the dynamic program is run over. The pseudocode for the full method with these pruning steps is outlined in Algorithm 4.

4.4 Evaluation of the Algorithm

To evaluate the OPPL algorithm we run it on simulated data. We consider data of length n with m equidistant changes $\tau_i = \lfloor \frac{ni}{m+1} \rfloor$, for $i = 1, \ldots, m$. We simulate the values at the changes as well as the values at $\tau_0 = 0$ and $\tau_{m+1} = t$. We simulate these from the normal distribution $\phi_i \sim N(0, 4)$ for $i = 0, \ldots, m+1$. We then simulate the data y_t for $t = 1, \ldots, n$ as

$$y_t = \phi_{\tau_i} + \tfrac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i}(t - \tau_i) + z_t, \quad \text{for } t = \tau_i, \ldots, \tau_{i+1}, \tag{4.16}$$

where $z_t \sim N(0, 1)$ and for $i = 0, \ldots, m$.

Firstly in Section 4.4.1 we will look at how the computational time of the algorithm behaves. Then in Section 4.4.2 we will compare to other methods in terms of both the time and the accuracy of the methods at fitting the continuous piecewise linear model.

4.4.1 Empirical Evaluation of the Computational Time

As discussed in Section 4.3.2 the computational time of the OPPL algorithm depends on the size of the two sets; $\hat{\mathcal{T}}_t$ and $\overset{*}{\mathcal{T}}_t$ for $t = 1, \ldots, n$. Empirically we explore the sizes of these sets. We run the algorithm on data sets of length $n = 1000$ with $m = 19$ equally spaced changepoints and randomly distributed values at the changes ($\phi_i \sim N(0, 4)$) and plot the average size of $\overset{*}{\mathcal{T}}_t$ (Figure 4.3(a)) and $\hat{\mathcal{T}}_t$ (Figure 4.3(b)).

Figure 4.3(a) shows that the size of $\overset{*}{\mathcal{T}}_t$ (the set of changepoint vectors which are optimal for some value of ϕ) stays relatively small and remains roughly constant as t increases. This indicates that $|\hat{\mathcal{T}}_t|$ should increase linearly as t increases (from the definition (4.11)) leading to the updating of the intervals $Int^t_{\boldsymbol{\tau}}$ to be roughly $\mathcal{O}(n)$ and the entire algorithm to be $\mathcal{O}(n^2)$.

Figure 4.3 (b) shows the actual size of $|\hat{\mathcal{T}}_t|$ to be increasing at a lower rate than what we'd expect. This is due to the inequality based pruning working. As discussed in Killick et al. (2012) and Maidstone et al. (2016) inequality based pruning works better as the number of changes in the data set increases.

Algorithm 4: Algorithm for Optimal Partitioning of Piecewise Linear Data (OPPL)

```
Input  : Set of data of the form $\mathbf{y}_{1:n} = (y_1, \dots, y_n)$,
         A cost function to be minimised, $\mathcal{C}$, dependent on the data and parameters $\phi_{\tau_i}$
         which correspond to the values fitted at the changepoints,
         A penalty $\beta$ which does not depend on the number or location of the changepoints.
Let $n =$ length of data;
And set $\hat{\mathcal{T}}_1 = \{1\}$;
for $t = 1, \dots, n$ do
    $f^t_{\emptyset}(\phi) = \min_{\phi'} \mathcal{C}(y_{1:t}, \phi', \phi)$;
    for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ do
        $f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'} \left\{ f^{\tau_k}_{\tau_1,\dots,\tau_{k-1}}(\phi') + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + \beta \right\}$;
    for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ do
        $Int^t_{\boldsymbol{\tau}} = \left\{ \phi : f^t_{\boldsymbol{\tau}}(\phi) = \min_{\boldsymbol{\tau}' \in \hat{\mathcal{T}}_t} f^t_{\boldsymbol{\tau}'}(\phi) \right\}$;
        $\overset{*}{\mathcal{T}}_t = \{\boldsymbol{\tau} : Int^t_{\boldsymbol{\tau}} \neq \emptyset\}$;
        $\hat{\mathcal{T}}_{t+1} = \hat{\mathcal{T}}_t \cup \left\{ (\boldsymbol{\tau}, t) : \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_t \right\}$;
    $\hat{\mathcal{T}}_{t+1} = \left\{ \boldsymbol{\tau} \in \hat{\mathcal{T}}_{t+1} : \min_{\phi} f^t_{\boldsymbol{\tau}}(\phi) \leq \min_{\phi', \boldsymbol{\tau}'} \left[ f^t_{\boldsymbol{\tau}'}(\phi') \right] + 2\beta \right\}$;
$f_{opt} = \min_{\boldsymbol{\tau}, \phi} f^n_{\boldsymbol{\tau}}(\phi)$;
$\boldsymbol{\tau}_{opt} = \arg\min_{\boldsymbol{\tau}} \left[ \min_{\phi} f^n_{\boldsymbol{\tau}}(\phi) \right]$;
Output: The optimal cost, $f_{opt}$ and the corresponding changepoint vector, $\boldsymbol{\tau}_{opt}$.
```

Algorithm 5: Algorithm for calculation of $Int^t_{\boldsymbol{\tau}}$ at time t

```
Input  : Set of changepoint candidate vectors $\hat{\mathcal{T}}_t$ for current timestep, $t$,
         Optimal segmentation functions $f^t_{\boldsymbol{\tau}}(\phi)$ for current time step $t$ and $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$.
$\mathcal{T}_{temp} = \hat{\mathcal{T}}_t$;
$Int^t_{\boldsymbol{\tau}} = \emptyset$ for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$;
$\phi_{curr} = -\infty$;
$\boldsymbol{\tau}_{curr} = \arg\min_{\boldsymbol{\tau} \in \mathcal{T}_{temp}} \left[ f^t_{\boldsymbol{\tau}}(\phi_{curr}) \right]$;
while $\mathcal{T}_{temp} \backslash \{\boldsymbol{\tau}_{curr}\} \neq \emptyset$ do
    for $\boldsymbol{\tau} \in \mathcal{T}_{temp} \backslash \{\boldsymbol{\tau}_{curr}\}$ do
        $x_{\boldsymbol{\tau}} = \{\min(\phi) : f^t_{\boldsymbol{\tau}}(\phi) - f^t_{\boldsymbol{\tau}_{curr}}(\phi) = 0 \ \& \ \phi > \phi_{curr}\}$;
        if $x_{\boldsymbol{\tau}} = \emptyset$ then
            $\mathcal{T}_{temp} = \mathcal{T}_{temp} \backslash \{\boldsymbol{\tau}\}$
    $\boldsymbol{\tau}_{new} = \arg\min_{\boldsymbol{\tau}}(x_{\boldsymbol{\tau}})$;
    $\phi_{new} = \min_{\boldsymbol{\tau}}(x_{\boldsymbol{\tau}})$;
    $Int^t_{\boldsymbol{\tau}_{curr}} = [\phi_{curr}, \phi_{new}] \cup Int^t_{\boldsymbol{\tau}_{curr}}$;
    $\boldsymbol{\tau}_{curr} = \boldsymbol{\tau}_{new}$;
    $\phi_{curr} = \phi_{new}$;
Output: The intervals $Int^t_{\boldsymbol{\tau}}$ for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$
```

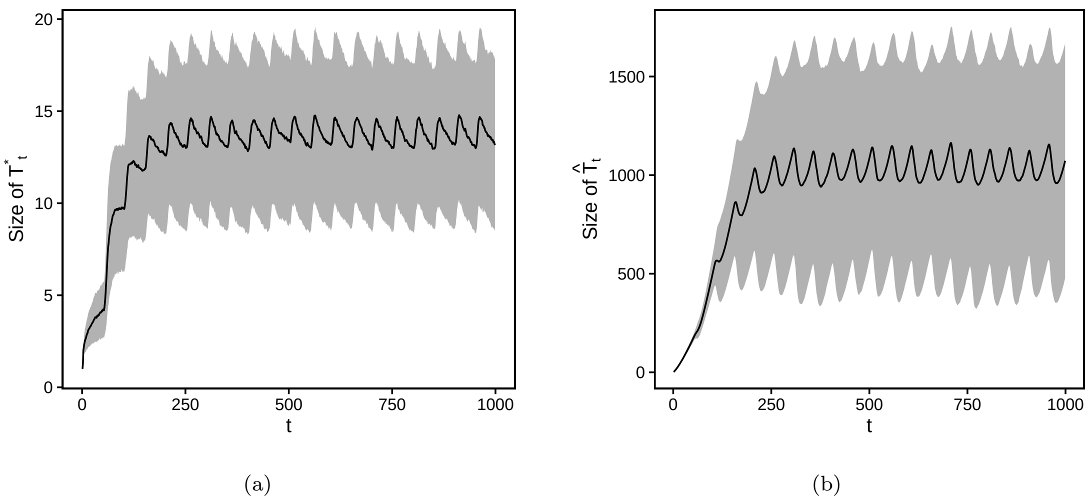


Figure 4.3: Length of the sets $\overset{*}{\mathcal{T}}_t$ (a) and $\hat{\mathcal{T}}_t$ (b) for data with 19 true changepoints located at $\boldsymbol{\tau} = (50, 100, \ldots, 950)$. Averaged over 1000 data sets. Shaded regions show 1 standard deviation away from the mean on either side.

The same is true in our algorithm. Figure 4.4 shows the average sizes of $\hat{\mathcal{T}}_t$ and $\overset{*}{\mathcal{T}}_t$ for data sets of length 1000 with no changepoints. Whilst the size of $\overset{*}{\mathcal{T}}_t$ remains roughly the same as before, the size of $\hat{\mathcal{T}}_t$ increases at a closer to linear rate and does not show as much pruning as before.

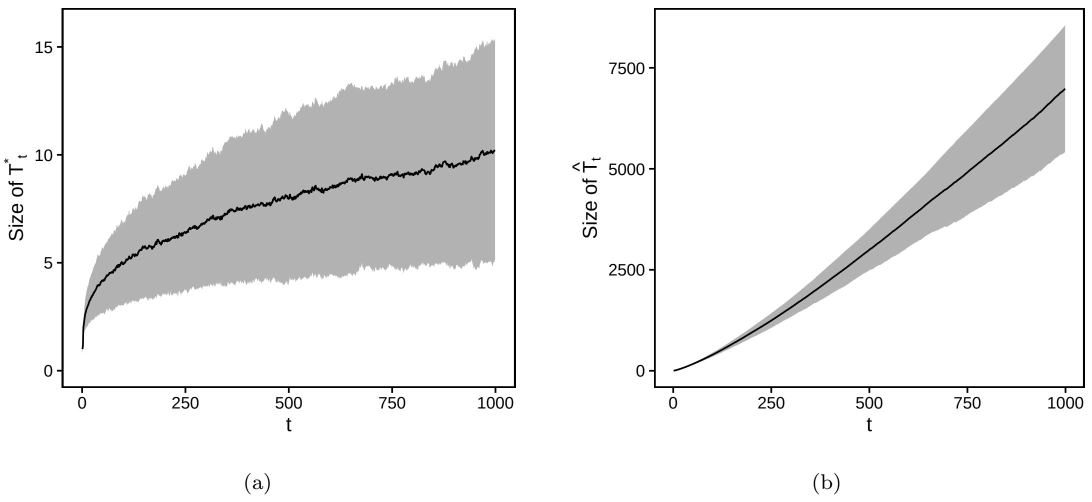


Figure 4.4: Length of the sets $\overset{*}{\mathcal{T}}_t$ (a) and $\hat{\mathcal{T}}_t$ (b) for data with no changepoints. Averaged over 1000 data sets. Shaded regions show 1 standard deviation away from the mean on either side.

Figures 4.3 and 4.4 suggest that, based on the empirical size of the sets $\hat{\mathcal{T}}_t$ and $\overset{*}{\mathcal{T}}_t$, the computational time taken by our algorithm roughly $\mathcal{O}(n^2)$ and sometimes less than this when the changepoints are densely distributed in the data. To further show this expected quadratic running time of the algorithm we evaluate the computational time in a range of scenarios.

Firstly we look at how it copes as the data length, n, increases. Figure 4.5(a) compares the time taken as the length of the data increases. We simulate data from a signal with equally spaced changepoints, random values at the changepoints and normal noise with variance equal to one. This is done for three scenarios; for a fixed number of changepoints ($m = 50$, black line), for linearly distributed changes ($m = \frac{n}{50}$, red line) and for sub-linearly distributed changes ($m = \lfloor\sqrt{n}\rfloor$, blue line).

OPPL can be seen to running in quadratic time in Figure 4.5(a), we also note that as the number of true changepoints becomes more dense in the data the algorithm runs faster. We further explore how the computational time is affected by the number of changepoints in the data set in Figure 4.5(b). Using the same signal as used in Figure 4.5(a) we allow the number of true changes to vary. Theoretically the more changes the more pruning can be done and hence the computational time should decrease as the number of changepoints increases. We can see evidence of this in Figure 4.5(b).

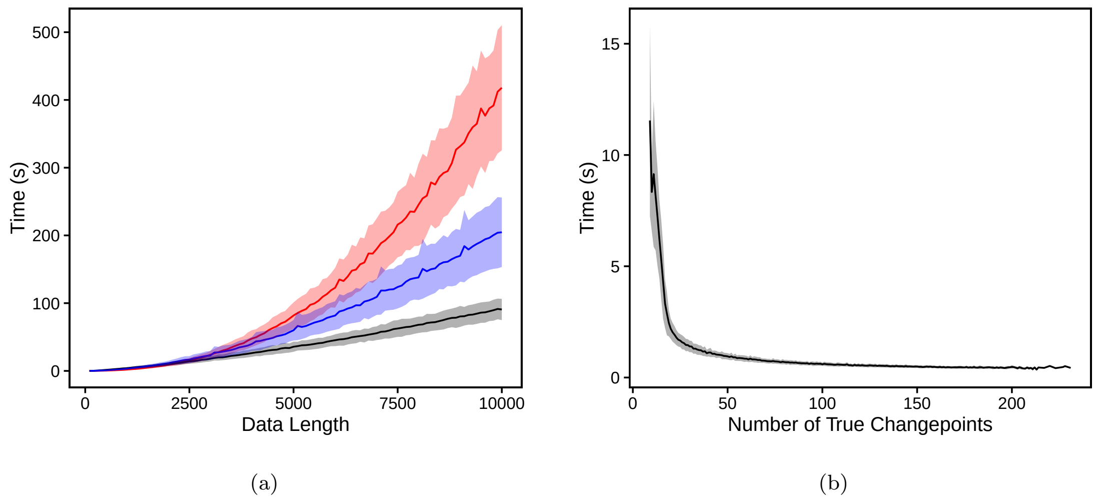


Figure 4.5: (a) Computational time taken by OPPL as the data length increases with $m = 50$ (red), $m = \frac{n}{50}$ (black) and $m = \lfloor\sqrt{n}\rfloor$ (blue) true changepoints. (b) Computational time taken by OPPL as the number of true changepoints increases for a data set of length $n = 1000$. Both (a) and (b) are averaged over 200 replicates. Shaded regions show 1 standard deviation away from the mean on either side.

4.4.2 Comparison with other methods

We compare OPPL with a number of alternative approaches for fitting a piecewise linear function to the data. We will compare these methods both for a fixed value of the tuning parameter and also as this value varies.

A naive way of fitting a piecewise linear function using existing changepoint methods is to analyse the first differences of the data. Then a simple change in mean can be fitted to the first differences under a quadratic loss. We can find the optimal segmentation for criteria using dynamic programming, and to do this we again use the PELT algorithm. However, in practice this provides sub-optimal segmentations to the other methods; giving estimates for the true function which both over estimate the true number of changes and give a poor fit to the true function. Because of this we exclude this method from our comparison.

The first method we compare against is the Trend Filtering algorithm (Kim et al., 2009). Trend Filtering aims to minimise the sum of squares, whilst adding a penalty based on the l_1 norm to penalise large variations in the estimated trend. The code we compare against uses a primal-dual interior point method to do the minimisation and is available at `http://stanford.edu/~boyd/l1_tf`.

Secondly we implement a greedy optimisation procedure to get approximate solutions to (4.4). This involves adding changepoints one at a time, with each new changepoint that we add being the one which reduces the cost (4.4) the most. This "best" changepoint can easily be computed by consider all possible values for the new change and computing the cost with this added in. This involves solving a $(k+1)$ dimensional quadratic form (which can be done in $\mathcal{O}(k+1)$ time by the Thomas Algorithm as the associated matrix is tridiagonal). We add changepoints until we can no longer reduce the cost. This is equivalent to a forward-variable selection approach in linear-regression, and is similar in spirit to a standard binary segmentation approach in simpler change in mean problems. We call this GREEDY.

Lastly we consider using standard changepoint methods to provide approximate solutions to (4.4). One way to do this is to ignore the continuity constraint at the changepoints, we call this INDEP. The resulting model consists of finding a change in both trend and intercept and can be solved by changepoint methods that do not deal with dependence between segments, in this section we use the PELT algorithm (Killick et al., 2012) to do this.

As Data Length Increases

Our new approach, and each of these alternatives requires the choice of the penalty. We will use the BIC penalty as a default with a cost function of twice the negative log-likelihood, and investigate how performance depends on this choice. Note the number of parameters added with each changepoint differs across method. For OPPL and GREEDY we add 2 parameters (the change in slope and changepoint location) with each changepoint so the penalty is $2\log(n)$. For INDEP we add 3 parameters with each changepoint, as we also have freedom to choose the intercept in the linear model, so we use the penalty $3\log(n)$.

For Trend Filtering the penalty choice is less obvious as the minimisation problem solved is different to the other methods. However by running Trend Filtering over a range of penalty values and we can still choose the segmentation that minimises the BIC as in the other methods. This requires us to pick the segmentation which minimises

$$\sum_{t=1}^{n}(y_t - \hat{y}_t)^2 + (k+2)\log(n) \tag{4.17}$$

where $\hat{y}_t$ is the estimate at time t and $k+2$ is the effective degrees of freedom (Tibshirani, 2014) for a Trend Filtering segmentation with k changepoints.

We run the methods on simulated data. A piecewise linear signal with normal noise with variance 1 is constructed. This is done for randomly chosen values at the changes ($\phi_{\tau_i} \sim \text{Normal}(0, 4^2)$).

We look at two scenarios. Firstly in Figure 4.6 we keep the segment size constant letting the changepoints occur every 50 data points (for a data set of length n we have $\frac{n}{50}$ true changepoints). Then in Figure 4.7 we let the segment size increase with n and keep the number of true changepoints constant (19 true changepoints). This should enable us to compare situations where the changepoints are both sparse in the data set as well as when they are more densely distributed. In both figures we compare the four methods in terms of their computational run time (a), the mean squared error (b), the true positive count (c) and the false positive count (d).

A true positive is defined as being a detected changepoint that is within ϵ time points either side of a true change. A false positive is defined a detected changepoint which is not a true positive. For the scenario analysed in Figure 4.6, where the segment size is constant, we choose $\epsilon = 5$. For the scenario analysed in Figure 4.7, where the segment size increases with n, we choose $\epsilon = 5\log n$. This comes from a theoretical result in Fryzlewicz (2014) which shows that error term should be proportional to $\frac{n^2}{\delta^2}\log n$ where δ is the minimum spacing between adjacent changepoints (in our case, $\delta \propto n$).

Both Figures 4.6 and 4.7 show that OPPL and Trend Filtering perform better than GREEDY and INDEP both in terms of mean squared error and true positive count. The improvement of OPPL over

GREEDY can be seen to be quite substantial in the penalised costs of the segmentations. For example for $n = 10000$, this reduction was on average 170.57 for $m = 19$ and 1064.64 for $m = 199$.

OPPL and Trend Filtering perform very similar in terms of MSE and the main difference is an increased false positive count for Trend Filtering. This is most apparent in Figure 4.6 where the number of changepoints is increasing linearly with the data and the false positve count of Trend Filtering increases at a much greater rate than the false positive count of OPPL. A similar result is shown for the fixed number of changepoints case in Figure 4.7. In Figure 4.7 the false positive count for Trend Filtering is better behaved, however it is still significantly higher than the false positive count for OPPL. Interestingly the false positive count for Trend Filtering seems to be decreasing as the data length increases, however it should be noted that it is still overestimating the number of changes at this point. Both Figures 4.6 and 4.7 show that Trend Filtering is a much more efficient algorithm than OPPL, however OPPL is still suprisingly efficient considering the complexity of the optimisation problem it is solving.

Varying the Penalty

In Figures 4.6 and 4.7 we choose the tuning parameters in each of the methods as either the BIC (for OPPL, INDEP and GREEDY) or by minimising the residual sum of squares plus the BIC (for Trend Filtering). However it is unclear as to how good these choices are. For OPPL and Trend Filtering we evaluate how changing the tuning parameter affects the given segmentations.

We compare the methods on individual data sets allowing the tuning parameters to vary. This is done for 100 data sets of length 1000 generated from a signal with 19 equally spaced changepoints, random values at the changes ($\phi_{\tau_i} \sim \text{Normal}(0, 4^2)$) and normal noise with variance 1. We compare the outputted segmentations by average mean squared error with average number of changepoints, Figure 4.8, and average true and false positive counts, Figure 4.9. Marked on the plots are the values of the tuning parameters which were used in Section 4.4.2.

In Figure 4.8 we can see that OPPL gives a good estimate of the number of changepoints and a low mean squared error for a similar parameter value. In comparison, getting both a low MSE and a good estimate of the number of changepoints is difficult using Trend Filtering, Figure 4.9 shows that this is due to the high false positive count given by trend filtering.

The fitted functions for the two methods are shown for the area around a true changepoint for an example data set in Figure 4.10. This is done for the BIC penalty for OPPL and the penalty value that minimises the RSS + BIC for Trend Filtering. Figure 4.10 shows how, while Trend Filtering estimates the true function well, it typically over estimates the number of changepoints by fitting multiple small slope changes, rather than one sharp change. This is due to the l_1 penalty used in Trend Filtering, as it penalises large changes in trend more than small ones.

Signal Types

The shape of the data can have an effect on the accuracy of the model fitted by the methods considered and we explore this by comparing the signal analysed before (with random true values at the changes) with a saw tooth function.

This saw tooth function is defined as in equation (4.16) with true changepoints, $\boldsymbol{\tau} = (0, 50, 100, \ldots, 1000)$, and values at the changes, $\boldsymbol{\phi} = (-1, 1, -1, 1, \ldots, 1, -1)$. We compare OPPL and Trend Filtering as the penalty changes as in Section 4.4.2. We compare the outputted segmentations by average mean squared error with average number of changepoints, Figure 4.11, and average true and false positive counts, Figure 4.12. Marked on the plots are the BIC value for OPPL and the corresponding value for Trend Filtering (the value of the MSE obtained by minimising (4.17)).

Similar results for the random function can be seen in the saw tooth function results, however the MSE for both OPPL and Trend Filtering deteriorates more as the parameter moves away from the best value. This makes penalty choice harder than before. For Trend Filtering, a parameter value that minimises the MSE still gives a large number of false positives, this makes parameter choice difficult. For OPPL the BIC still seems like a good choice of penalty.

4.5 Detecting Changes in the Rotation of Bacterial Flagellar Motors

Bacterial flagellar motors (BFMs) are rotary molecular machines which are embedded in the bacterial cell envelope. For more background on these biological systems see (Sowa et al., 2005; Sowa and Berry, 2008; Zhou et al., 1998).

Data on the motion of the BFMs has been collected by Ryu et al. (2000); Chen and Berg (2000a,b);

Sowa et al. (2003) amoung others. Sowa et al. (2005) observed that the rotation of the BFM exhibits a step like structure in the rotation. These steps are due to the motor being affected by external physiological conditions. Recent work in Weinmann and Storath (2015) applied an iterative Potts algorithm to fit a piecewise constant model to the data to capture the stepping motion of the BFM, however recent analysis by Nord and Berry (to be written) suggests that fitting a piecewise linear model may be more appropriate.

We apply the OPPL method to the data used by both Sowa et al. (2005) and Weinmann and Storath (2015) to fit a piecewise linear model with continuity at the changepoints. We show the resulting fitted model (red) as well as the original data (black) for the iterative Potts approach (Figure 4.13) and OPPL (Figure 4.14). As can be seen the piecewise linear approach seems more appropriate than the piecewise constant approach used in Weinmann and Storath (2015). This is most apparent where the segments ressemble a slope, this occurs in multiple places and can be seen clearly in (b) of Figures 4.13 and 4.14. Iterative Potts fits these cases as multiple small segments whereas OPPL can fit a single segment here.

We explore the advantages of fitting a piecewise linear model over a piecewise constant one further by comparing, for a given number of parameters, the residual sum of squares. To obtain a comparable piecewise constant model we minimise a penalised sum of squares cost function using the PELT algorithm. Both methods minimise the square error loss and therefore we can compare them. We run both methods for a range of penalties using the CROPS algorithm (Haynes et al., 2015) and compare the number of parameters in the model ($2m + 2$ for the piecewise linear model and $2m + 1$ for the piecewise constant model, where m is the number of changepoints) to the unpenalised cost (which is proportional to the residual sum of squares). As can be seen in Figure 4.15 the cost for a given number of parameters is less for the piecewise linear model, implying that the piecewise linear model consistently provides a better fit.

Our method can be used to analyse the step-like structure of the rotation of the bacterial flagellar motor. This step-like nature of the motor is of interest to biologists, and particular interest is in the regions where the motor stops moving, known as "dwell regions", and the angles at which these occur. Recent thought on this is that these dwells occur at periodic angles (Sowa et al., 2005), and we use our method to test this. We define a dwell as a segment with gradient close to zero, $|\frac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i}| < \delta$, and of sufficient length, $\tau_{i+1} - \tau_i > l$. To choose δ and l we look at how the segment length and segment gradients are distributed (Figure 4.16). Assuming that the “dwell” regions (close to zero gradient) are distributed differently from the rest of the segments we take the first local minima in Figure 4.16(a) as the threshold between them. We compute this point as $\delta = 437.5$. We use a similar technique in Figure 4.16(b) to separate out the segments of meaningful length and estimate $l = 87.5$.

Figure 4.17(a) shows these dwell regions plotted as the red segments. We are interested in how the angles that relate to these dwell regions are distributed. We use the means of these segments to determine whether or not these segments occur at periodic angles at each rotation. To do so we look at how well they fit for a given periodicity by first transforming our means, X, onto $[0, 1]$ (where 1 is a full rotation) and then for $d = 1, \ldots, 50$ minimising the mean absolute error

$$\min_{\mu, j_1, \ldots, j_m} \frac{d}{m} \sum_{i=1}^{m} |X_i - \mu - \frac{j_i}{d}| \tag{4.18}$$

where $j_i \in 0, \ldots, d-1$.

We can then see for which value of d this is smallest to give an estimate of the period. We plot the mean absolute errors for multiple values of the period, d, in Figure 4.17(b). As can be seen the smallest mean absolute error occurs for $d = 8$ and has a value of 0.1573. We compare this value against a null distribution constructed by simulating 1000 Uniform zero one distributions of the same length as the number of dwell regions for our segmentation in (a) and computing the mean absolute errors. The 5% empirical quartile of this distribution is 0.1558, which is less than the value given by the dwell regions in our data set, indicating that the angles that the dwell regions occur at can be explained as being uniformly distributed and the observed period of 8 is not significant.

Our analysis indicates that whilst we can get an estimate for the period of 8, this is not very close to the period of 26 observed in Sowa et al. (2005) and can be shown to not be significant. Thus our analysis shows that there is no evidence that the “dwell” regions occur at regularly spaced periodic angles.

4.6 Discussion

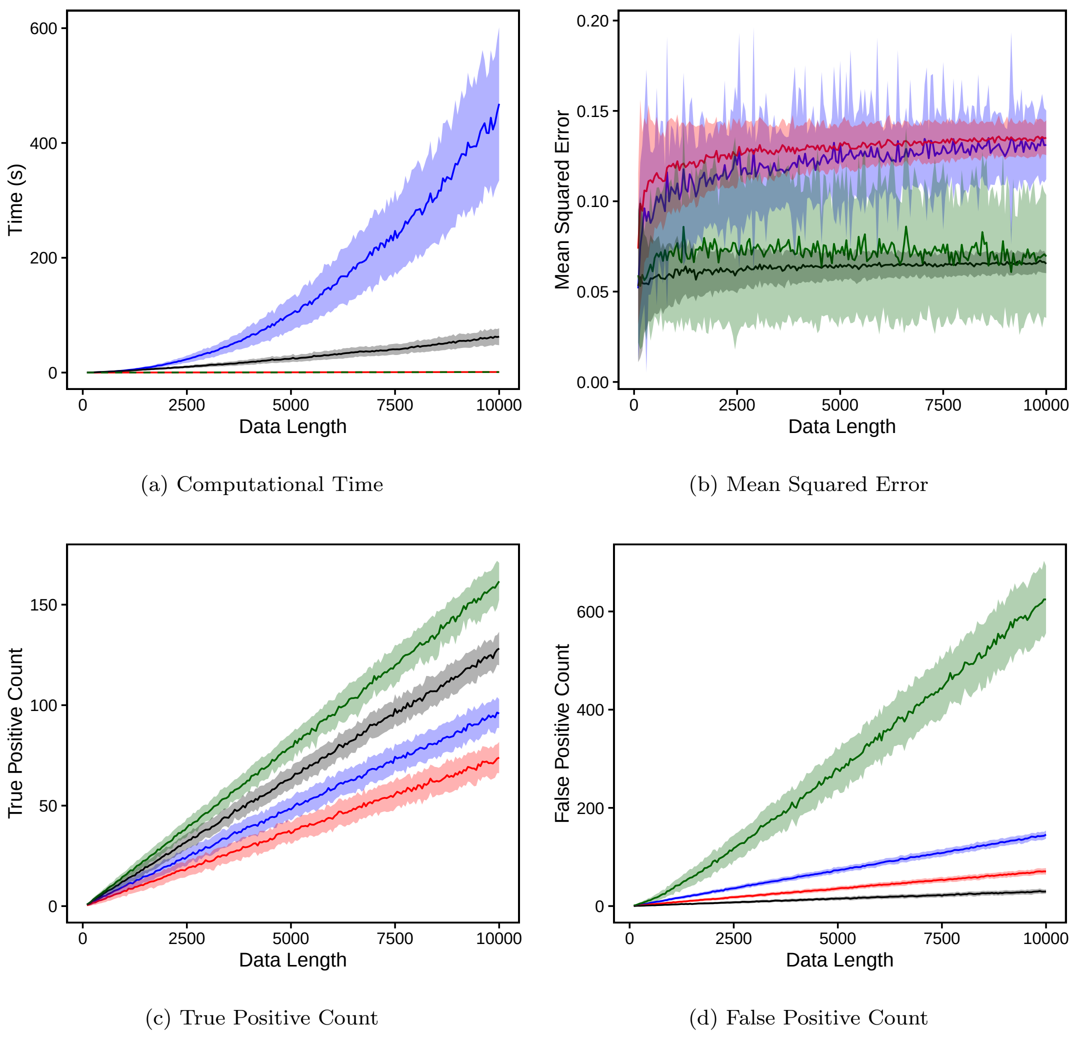
In this chapter we have introduced a pruned dynamic programming algorithm for detection of changepoints under the assumption of a continuous piecewise linear model. Our method, Optimal Partitioning for Piecewise Linear data (OPPL), has been shown to be both accurate and computationally viable.

The OPPL method exactly minimises a square error loss cost function on a piecewise linear signal which is continuous at the changepoints. Solving this minimisation problem by complete enumeration is computational expensive, so we suggested a dynamic programming approach. We defined a recursion based on the minimising over the location of all previous changepoints, to keep the efficiency of the algorithm to a minimum we utilised both inequality based and functional pruning.

In Section 4.4.1 we have shown that pruning techniques improve the computational time of our algorithm from exponential in the length of the data, to being empirically $\mathcal{O}(n^2)$ (for data of length n). We have then shown, in Section 4.4.2, that our method performs well both in terms of estimating the underlying function (obtaining a low value for the mean squared error) and in terms of estimating the changepoints accurately (by looking at the true and false positives) when compared to competing methods. We have first shown this for penalties values which we feel perform well; the BIC for OPPL, GREEDY and INDEP and for Trend Filtering a related penalty obtained by minimising the residual sum of squares plus the BIC. Our simulations have shown that the closest competing method is Trend Filtering, however Trend Filtering tends to over estimate the number of changes. We have observed that this is inherent in Trend Filtering as due to the l_1 penalty term when choosing a parameter value there is a trade off between fitting the function and correctly identifying the number of changes.

In Section 4.5 we applied the OPPL method to detecting changes in the rotation of a bacterial flagellar motor. We described how, despite a piecewise constant model being fitted by previous authors, the BFM data is best modelled by a continuous piecewise linear model. Addressing whether there is a period to the stepping motion of the BFM, we used our method to identify dwell regions and determine whether they are distributed periodically. We determined that any periodicity that can be observed is not significant.

The OPPL algorithm offers an exact dynamic programming algorithm for efficiently fitting a continuous piecewise linear model. Although here only a continuous change in trend is considered the theory of the method can be adapted for a more general class of dependence between segments, for example fitting splines. The issue with doing this is that it may affect the efficiency of the method. We discuss this potential extension further in Chapter 6.



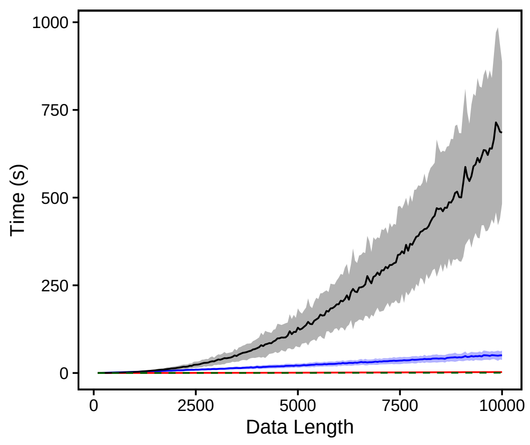
(a) Computational Time

(b) Mean Squared Error

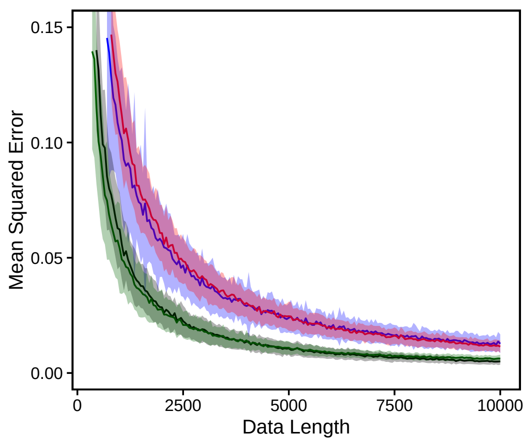
(c) True Positive Count

(d) False Positive Count

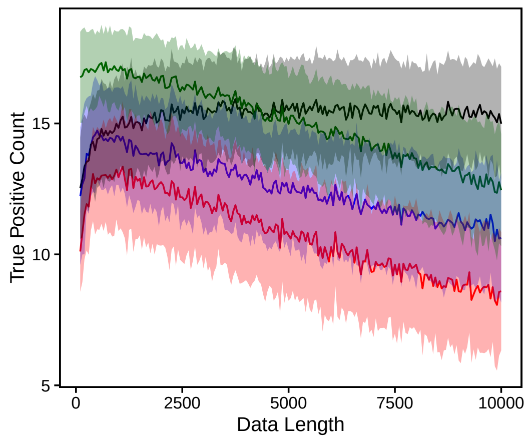
Figure 4.6: Comparison of computational run-time (a), mean squared error (b), true positive count (c), and false positive count (d). This is done for OPPL (black), GREEDY (blue), INDEP (red) and Trend Filtering (green) on data sets with $\frac{n}{50}$ true changepoints (where n is the length of the data set). Averaged over 100 data sets. Shaded regions show 1 standard deviation away from the mean on either side.



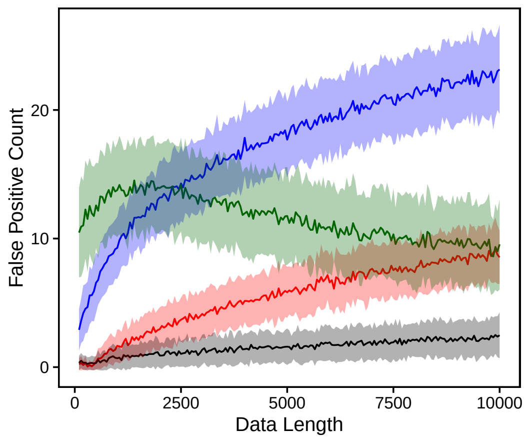
(a) Computational Time



(b) Mean Squared Error

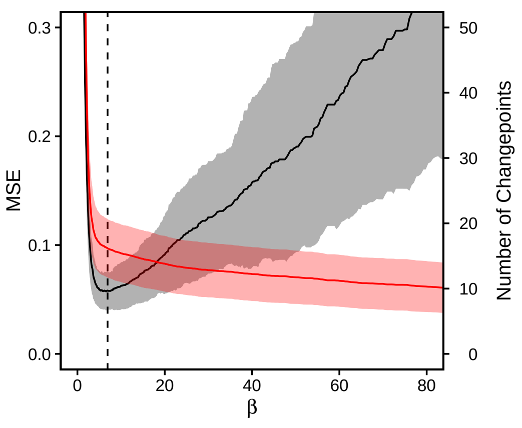


(c) True Positive Count

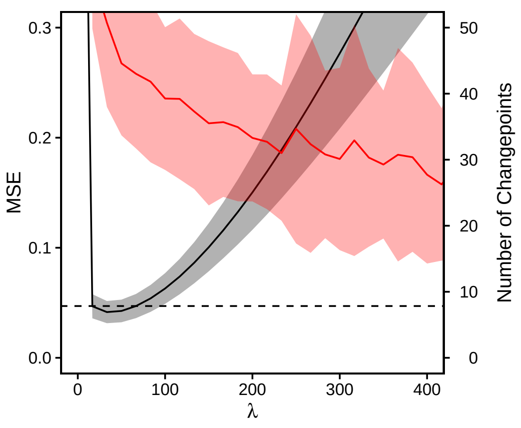


(d) False Positive Count

Figure 4.7: Comparison of computational run-time (a), mean squared error (b), true positive count (c), and false positive count (d). This is done for OPPL (black), GREEDY (blue), INDEP (red) and Trend Filtering (green) on data sets with 19 true changepoints. Averaged over 100 data sets. Shaded regions show 1 standard deviation away from the mean on either side.

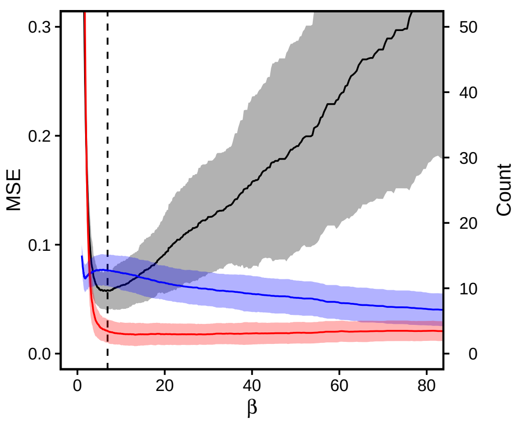


(a) OPPL

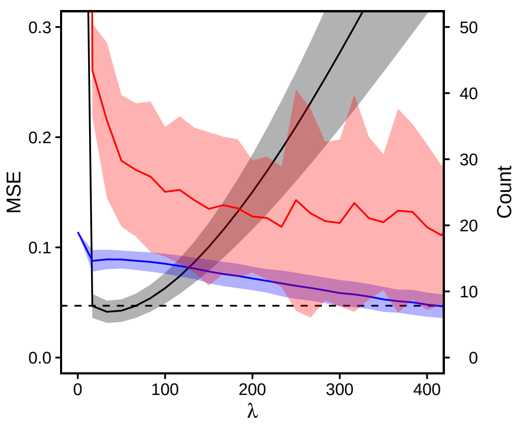


(b) Trend Filtering

Figure 4.8: Mean squared error (black) and number of changepoints detected (red) as the parameter changes. Ran on data sets with 19 equidistant changepoints and randomly drawn values at the changes and averaged over 100 such data sets. The dotted line corresponds to the BIC segmentation (where for Trend Filtering this is the value of the MSE obtained by minimising (4.17)). Shaded regions show 1 standard deviation away from the mean on either side.

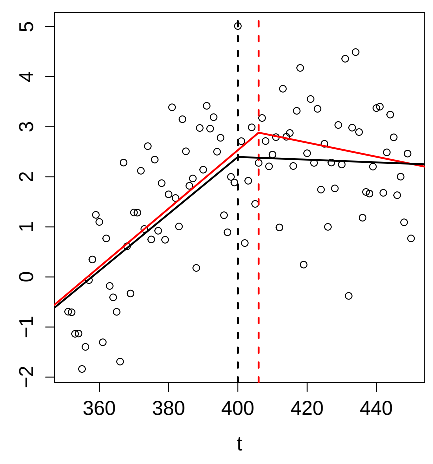


(a) OPPL

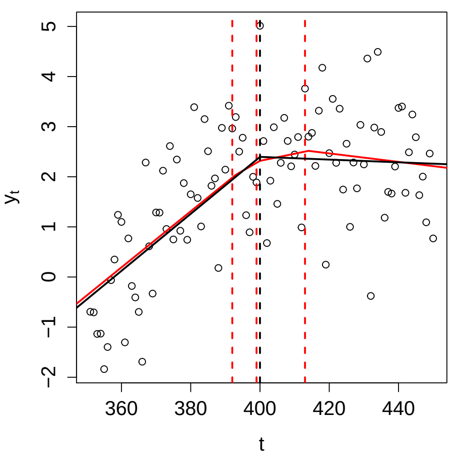


(b) Trend Filtering

Figure 4.9: Mean squared error (black), true positive count (blue) and false positive count (red) as the parameter changes. Ran on data sets with 19 equidistant changepoints and randomly drawn values at the changes and averaged over 100 such data sets. The dotted line corresponds to the BIC segmentation (where for Trend Filtering this is the value of the MSE obtained by minimising (4.17)). Shaded regions show 1 standard deviation away from the mean on either side.

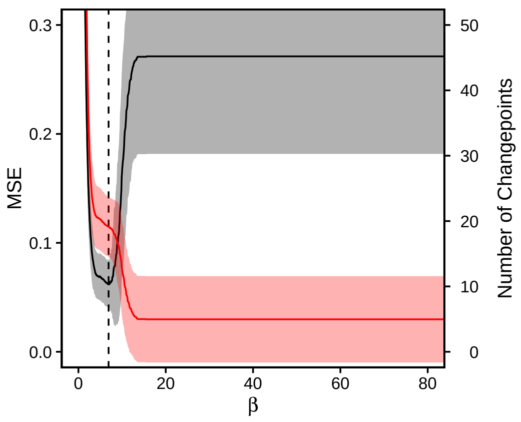


(a) OPPL

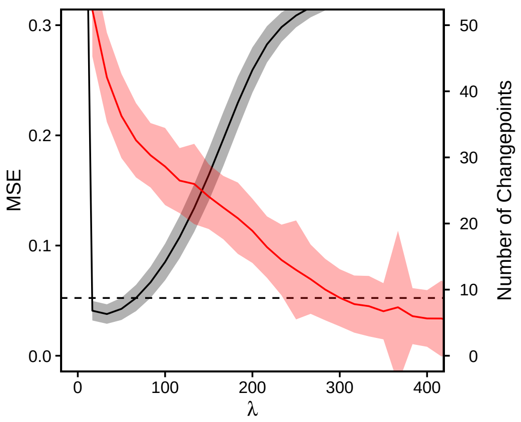


(b) Trend Filtering

Figure 4.10: Examples of fit given by OPPL and Trend Filtering. Data is simulated with 9 true changepoints and randomly drawn values at the changes on a data set of length 1000. Shown is the data and true function (black) and the fit and estimated changepoints (red) around one of these changes at $t = 400$. Methods are run with the BIC penalty.

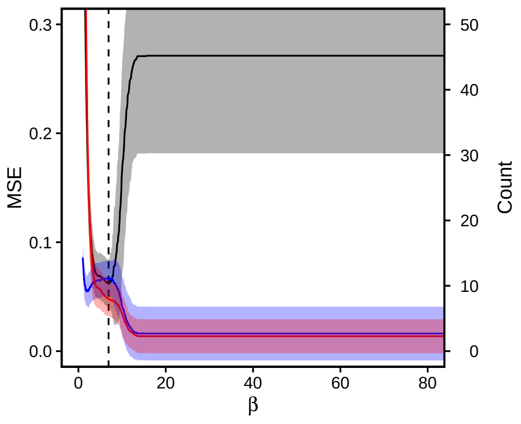


(a) OPPL

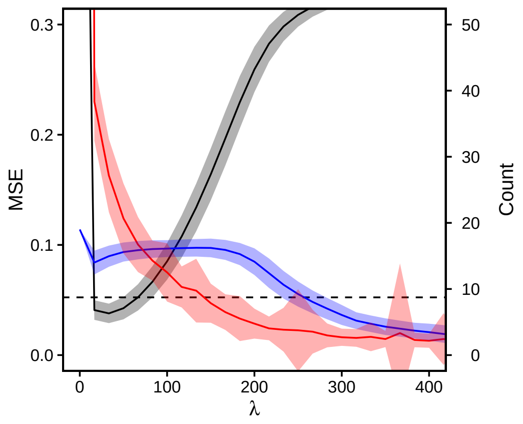


(b) Trend Filtering

Figure 4.11: Mean squared error (black) and number of changepoints detected (red) as the parameter changes. Ran on data sets with 19 equidistant changepoints and with alternating values at the changes. Results are averaged over 100 such data sets. The dotted line corresponds to the BIC segmentation (where for Trend Filtering this is the value of the MSE obtained by minimising (4.17)). Shaded regions show 1 standard deviation away from the mean on either side.

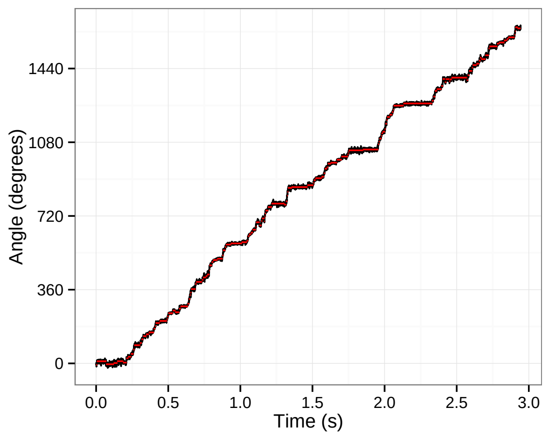


(a) OPPL

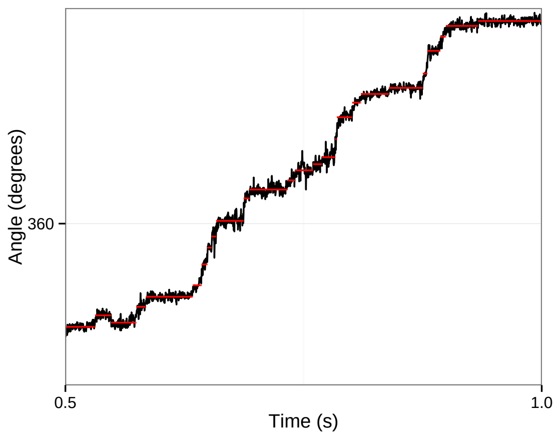


(b) Trend Filtering

Figure 4.12: Mean squared error (black), true positive count (blue) and false positive count (red) as the parameter changes. Ran on data sets with 19 equidistant changepoints and with alternating values at the changes. Results are averaged over 100 such data sets. The dotted line corresponds to the BIC segmentation (where for Trend Filtering this is the value of the MSE obtained by minimising (4.17)). Shaded regions show 1 standard deviation away from the mean on either side.



(a)



(b)

Figure 4.13: The rotation of a bacterial flagellar motor over time, exhibiting a “step-like” structure. Red line corresponds to the estimates given by using a Potts functional (Weinmann and Storath, 2015). The entire data set is shown in (a) and a zoomed in section of the graph in (b). (Original data by courtesy of Sowa et al. (2005)).

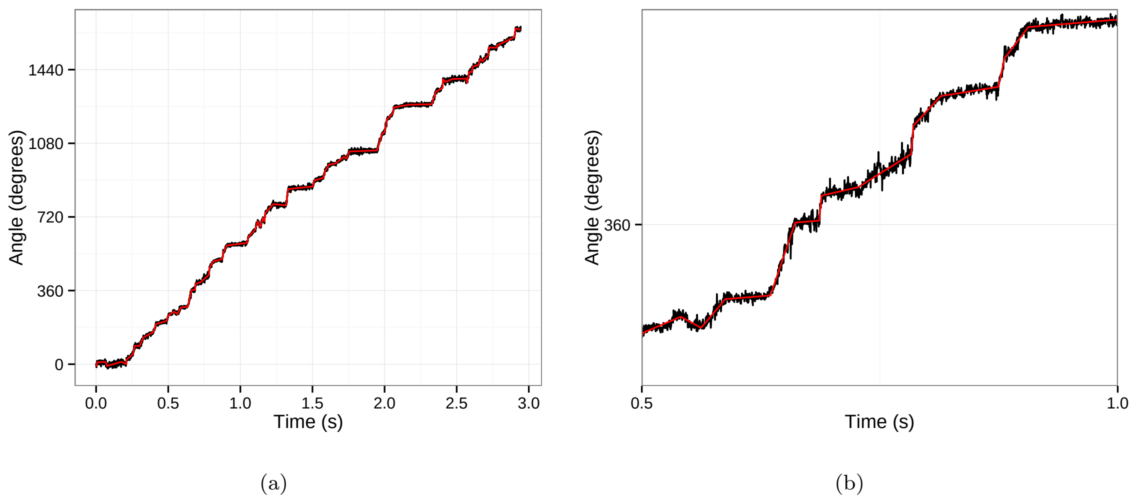


Figure 4.14: The rotation of a bacterial flagellar motor over time, exhibiting a "step-like" structure. Red line corresponds to the estimates given by our method, OPPL. The entire data set is shown in (a) and a zoomed in section of the graph in (b). (Original data by courtesy of Sowa et al. (2005)).

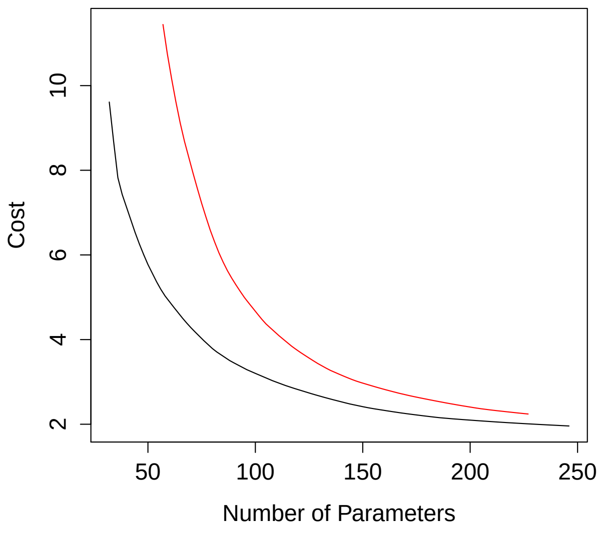
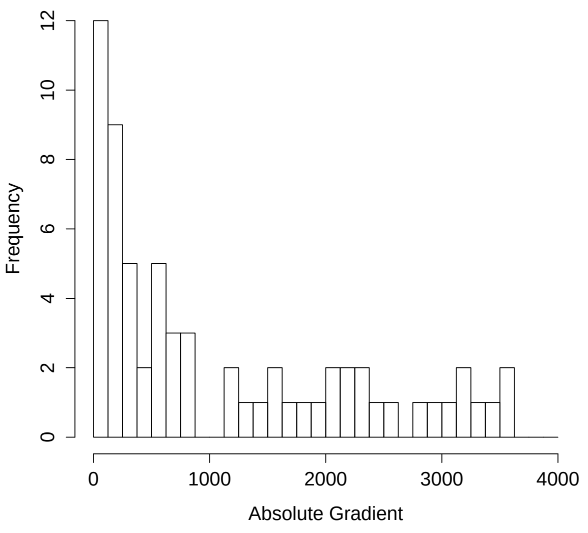
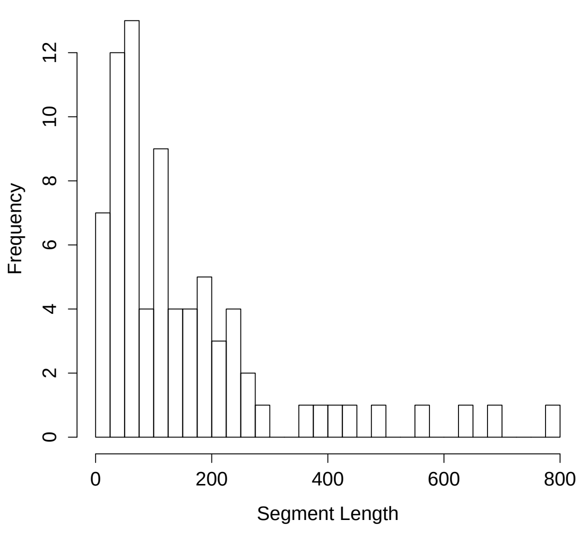


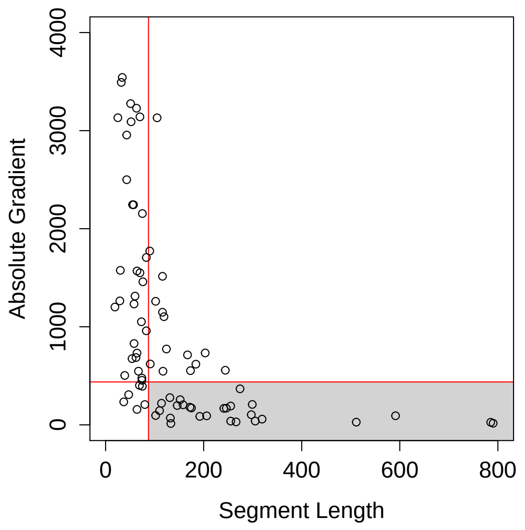
Figure 4.15: Comparing the unpenalised cost for the piecewise constant (red) and piecewise linear (black) models as the number of parameters increases.



(a)

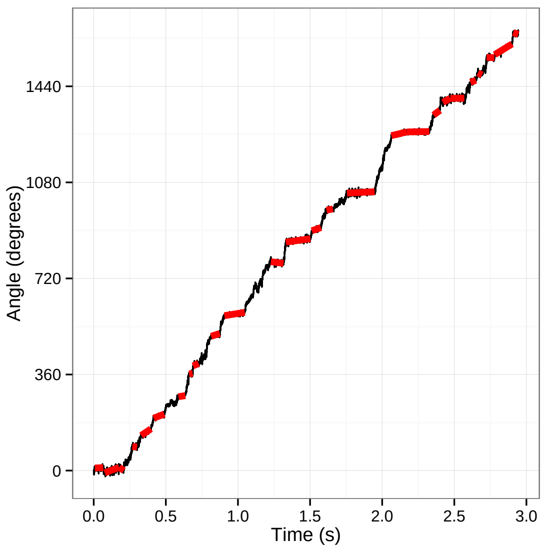


(b)

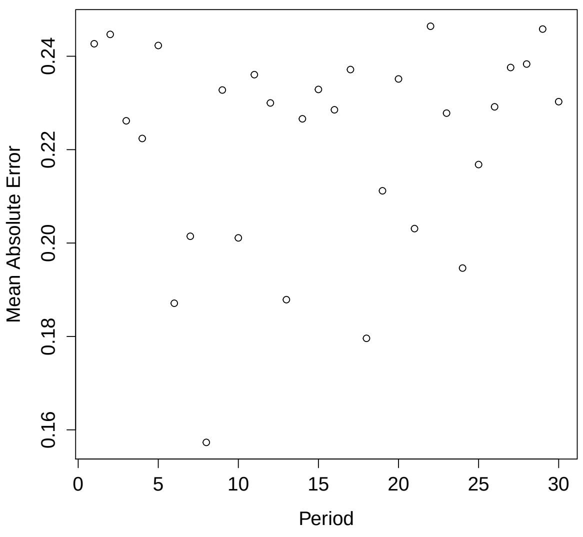


(c)

Figure 4.16: Histograms of the absolute gradient (a) and the length of the segments (b) and a scatter plot of both of them (c) in the segmentation found in Figure 4.14, as well as a scatter plot of the two. Red lines correspond to the threshold values $\delta = 437.5$ and $l = 87.5$ and the grey region the segments we class as “dwell segments”.



(a)



(b)

Figure 4.17: Analysis of the dwell regions (defined by $\delta = 437.5$ and $l = 87.5$). These dwells are shown in (a) by the red segments. The mean absolute errors (4.18) of these dwell regions for various periods are shown in (b).

Chapter 5

OPPL for Penalties which Depend on the Segment Length

5.1 Introduction

In Chapter 4 we looked at fitting a piecewise linear model with continuity at the changes and developed the method Optimal Partitioning for Piecewise Linear data (OPPL). We achieved this by minimising the sum of a cost function over all segments and penalising to avoid over fitting

$$\min_{\boldsymbol{\tau},m,\boldsymbol{\phi}} \left[\sum_{i=0}^{m} \mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \beta(m+1)\right]. \tag{5.1}$$

We consider generalisations of the penalty that depend on the segment lengths. In particular we introduce a penalty function of the form $\sum_{i=0}^{m} h(\tau_{i+1} - \tau_i)$ for some function h that depends on the segment lengths. Note that penalties of the form in (5.1) are a specific case of this with $h(s_i) = \beta$.

Examples of such penalty functions arise in Zhang and Siegmund (2007) where the modified Bayesian information criterion (mBIC) is introduced and when using the minimum description length (MDL, Rissanen (1989)). In both these cases $h(s_i) = a + b\log(s_i)$ for appropriate constants a and b and the segment length s_i.

In this chapter we show how the OPPL method from Chapter 4 can easily be extended to solve the resulting minimisation problem. This is achieved via slight modifications to both the dynamic program and the pruning steps. We then evaluate the impact of using the mBIC condition, where

$h(s_i) = \log(n) + \frac{1}{2}\log(s_i)$, on both the speed of OPPL and the accuracy of the resulting segmentations.

5.2 Penalised Cost with General Penalty Function

We aim to fit the same continuous piecewise linear model as described in equation (4.2). Once again we define a cost function $\mathcal{C}(\mathbf{y}_{s+1:t}, \phi_s, \phi_t)$ for $s \leq t$ on the segment from y_{s+1} to y_t where ϕ_s and ϕ_t are the parameters associated with the values at the start and end of the segment respectively. As before a natural choice of cost function is the square error loss

$$\mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) = \frac{1}{2\sigma^2} \sum_{j=\tau_i+1}^{\tau_{i+1}} \left(y_j - \phi_{\tau_i} - \frac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i}(j - \tau_i) \right)^2 . \tag{5.2}$$

Rather than adding a penalty that is a constant value for every segment, we allow the penalty to depend on the segment length and denote it by the function $h(\tau_{i+1} - \tau_i)$. The minimisation problem we wish to solve is as follows

$$\min_{\boldsymbol{\tau}, m, \boldsymbol{\phi}} \left[\sum_{i=0}^{m} \mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \sum_{i=0}^{m} h(\tau_{i+1} - \tau_i) \right] . \tag{5.3}$$

We will then solve the minimisation problem in (5.3) using the pruned dynamic programming algorithm, OPPL, that was developed in Chapter 4.

5.3 OPPL with a Penalty that Depends on the Segment Length

To solve (5.3) exactly we will use a modified version of the OPPL method from Chapter 4. Whilst the main algorithm remains the same, modifications will have to be made to account for the non-linear penalty term. This more complex penalty will also affect the pruning steps in the algorithm.

Using the same notation as in Chapter 4 we define the function $f^t(\phi)$ for the optimal segmentation up to time t with changepoints with $\phi_{\tau_{k+1}} = \phi$ where $\phi_{\tau_{k+1}}$ is the fitted value at the end point of the segment at $\tau_{k+1} = t$

$$\begin{aligned} f^t(\phi) = \quad & \min_{\boldsymbol{\tau}, k, \boldsymbol{\phi}} \quad \left[\sum_{i=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \sum_{i=0}^{k} h(\tau_{i+1} - \tau_i) \right] , \\ & \text{s.t.} \quad\quad \phi_t = \phi. \end{aligned} \tag{5.4}$$

As in Chapter 4 we wish to construct a dynamic programming recursion for $f^t(\phi)$ based on minimising over the position of the last changepoint, which we will call s. As our penalty term is additive in the segments we can do this in much the same way as before but changing the penalty term. Instead of adding a constant term, β, with each segment we add a penalty which depends on the length of the new segment being added

$$f^t(\phi) = \min_{\phi', s} \left[f^s(\phi') + \mathcal{C}(\mathbf{y}_{s+1:t}, \phi', \phi) + h(t-s) \right].$$

As before this updating is difficult due to the continuous parameter ϕ which means that $f^t(\phi)$ cannot be easily stored. Instead, we store the functions $f^t_{\boldsymbol{\tau}}(\phi)$ which are the cost functions for the optimal segmentation up to time t with changepoints at $\boldsymbol{\tau} = \tau_1, \ldots, \tau_k$ and $\phi_t = \phi$. These are defined as follows

$$\begin{aligned} f^t_{\boldsymbol{\tau}}(\phi) = \quad & \min_{\boldsymbol{\phi}} \quad \left[\sum_{i=0}^{k} \mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) + \sum_{i=0}^{k} h(\tau_{i+1} - \tau_i) \right], \\ & \text{s.t.} \quad \phi_t = \phi. \end{aligned} \tag{5.5}$$

These functions $f^t_{\boldsymbol{\tau}}(\phi)$ are easier to store as they are quadratics and the coefficients can be stored. Once again we can then get a recursion for these functions, adapting for the none constant penalty

$$f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'} \left\{ f^{\tau_k}_{\tau_1, \ldots, \tau_{k-1}}(\phi') + \mathcal{C}(\mathbf{y}_{\tau_k+1:t}, \phi', \phi) + h(t - \tau_k) \right\}. \tag{5.6}$$

The updating of these functions can be done by updating the coefficients of the quadratics. This is discussed in Appendix B.1. The function $f^t(\phi)$ can then be returned by taking the piecewise minimum of these functions where, for a given value of ϕ

$$f^t(\phi) = \min_{\boldsymbol{\tau} \in \mathcal{T}_t} f^t_{\boldsymbol{\tau}}(\phi), \tag{5.7}$$

where we define $\mathcal{T}_t$ as the set of all possible changepoint vectors at time t. We can compute $f^n(\phi)$ and then, by minimising over ϕ, calculate the optimal segmentation of $\mathbf{y}_{1:n}$.

As before we speed up the algorithm by using pruning techniques. Due to the modified recursion in (5.6) these pruning steps will also have to be adapted.

5.3.1 Functional Pruning

Functional pruning as outlined in Section 4.3.2 prunes the set of changepoint vectors for which cost functions are stored, $\mathcal{T}_t$. This process relies on comparing the cost functions, $f^t_{\boldsymbol{\tau}}(\phi)$, to one another and storing only those which form part of the piecewise minimum function, $f^t(\phi)$ (as defined in (5.7)). For our new model this can be done in much the same way as before.

We again define $\overset{*}{\mathcal{T}}_t$ as the set of all changepoints vectors that are optimal for some ϕ at time t, as in (4.9). We can then calculate estimates for $\overset{*}{\mathcal{T}}_{t+1}$ based on knowledge of $\overset{*}{\mathcal{T}}_t$ by using Theorem 3 from Section 4.3.2. Theorem 3 states that if for a candidate changepoint vector $\boldsymbol{\tau}$ at time s we have $\boldsymbol{\tau} \notin \overset{*}{\mathcal{T}}_s$ then $(\boldsymbol{\tau}, s) \notin \overset{*}{\mathcal{T}}_t$ for all future time steps, $t > s$. Whilst this theorem still holds for our general penalty function, the proof of it needs to be slightly modified.

Proof of Theorem 3 for a general penalty function. This follows the layout of the original proof, substituting β with the function h evaluated on the relevant segment. Assume $(\boldsymbol{\tau}, s) \in \overset{*}{\mathcal{T}}_t$, then there exists ϕ such that

$$f^t(\phi) = f^t_{(\boldsymbol{\tau},s)}(\phi),$$

Now for any ϕ^*,

$$\begin{aligned} f^s(\phi^*) + \mathcal{C}(\mathbf{y}_{s+1:t}, \phi^*, \phi) + h(t-s) &\geq \min_{\phi', r} \left[f^r(\phi') + \mathcal{C}(\mathbf{y}_{r+1:t}, \phi', \phi) + h(t-r) \right], \\ &= f^t(\phi), \\ &= f^t_{(\boldsymbol{\tau},s)}(\phi), \\ &= \min_{\phi''} \left\{ f^s_{\boldsymbol{\tau}}(\phi'') + \mathcal{C}(\mathbf{y}_{s+1:t}, \phi'', \phi) + h(t-s) \right\}, \qquad (5.8) \\ &= f^s_{\boldsymbol{\tau}}(\phi^A) + \mathcal{C}(\mathbf{y}_{s+1:t}, \phi^A, \phi) + h(t-s), \end{aligned}$$

where ϕ^A is the value of ϕ'' which minimises (5.8). Then as ϕ^* can be chosen as any value, we can choose it as ϕ^A. By cancelling terms we get $f^s(\phi^A) \geq f^s_{\boldsymbol{\tau}}(\phi^A)$ and hence (from (5.7)), $f^s(\phi^A) = f^s_{\boldsymbol{\tau}}(\phi^A)$ and therefore $\boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s$. We have shown that if $(\boldsymbol{\tau}, s) \in \overset{*}{\mathcal{T}}_t$ then $\boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s$, by taking the contrapositive the theorem holds. ■

We can, as before, define the set

$$\hat{\mathcal{T}}_t = \left\{ (\boldsymbol{\tau}, s) : s \in \{0, \ldots, t-1\}, \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_s \right\}, \qquad (5.9)$$

and by Theorem 3 we have $\hat{\mathcal{T}}_t \supseteq \overset{*}{\mathcal{T}}_t$. This set provides an estimate for the set $\overset{*}{\mathcal{T}}_t$ and we limit the updating of the cost functions to this subset. $\overset{*}{\mathcal{T}}_t$ can be calculated from $\hat{\mathcal{T}}_t$ in the same way as in Section 4.3.2 by computing the intervals of the parameter space for which the segmentations are optimal, $Int^t_{\boldsymbol{\tau}}$ (this process is outlined in Algorithm 5). Using the sets $\overset{*}{\mathcal{T}}_t$ and $\hat{\mathcal{T}}_t$ we can then calculate $\hat{\mathcal{T}}_{t+1}$

$$\hat{\mathcal{T}}_{t+1} = \hat{\mathcal{T}}_t \cup \left\{ (\boldsymbol{\tau}, t) : \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_t \right\}. \qquad (5.10)$$

5.3.2 Inequality Based Pruning

We can also utilise inequality based pruning as in Section 4.3.3. Here the pruning depends on the value of the penalty in the model and as of such will have to be modified to work for a general penalty function. We modify Theorem 4 to account for the change in penalty.

Theorem 5 *If for some $\boldsymbol{\tau}$,*

$$\min_{\boldsymbol{\phi}} f^t_{\boldsymbol{\tau}}(\boldsymbol{\phi}) > \min_{\boldsymbol{\phi}',\boldsymbol{\tau}'} \left[f^t_{\boldsymbol{\tau}'}(\boldsymbol{\phi}')\right] + h(1) + \max_{s_i=1,\dots,n} h(s_i), \tag{5.11}$$

holds, then at any future time $T > t$, the set of changepoints $\boldsymbol{\tau}$ can never be optimal for the data $y_{1:T}$.

The proof follows a similar format as for the proof to Theorem 4. However, instead of adding penalty of 2β (to account for adding two changes at t and $t+1$) we will have to add the corresponding values of the penalty function h.

Proof. Firstly note that by adding changes, at any point, without the penalty term and minimising over the corresponding ϕ values will decrease the cost. Therefore we have

$$f^T_{\boldsymbol{\tau}}(\phi) \geq \min_{\phi',\phi''} \left[f^t_{\boldsymbol{\tau}}(\phi') + \mathcal{C}(y_{t+1},\phi',\phi'') + \mathcal{C}(\mathbf{y}_{t+2:T},\phi'',\phi)\right]. \tag{5.12}$$

Then assuming that (5.11) is true, it can be shown that the segmenting the data $\mathbf{y}_{1:T}$ with changepoints $\boldsymbol{\tau}$ is always sub-optimal.

So from (5.12) and using (5.11)

$$\begin{aligned} f^T_{\boldsymbol{\tau}}(\phi) &\geq \min_{\phi',\phi''} \left[f^t_{\boldsymbol{\tau}}(\phi') + \mathcal{C}(y_{t+1},\phi',\phi'') + \mathcal{C}(\mathbf{y}_{t+2:T},\phi'',\phi)\right], \\ &\geq \min_{\phi'}[f^t_{\boldsymbol{\tau}}(\phi')] + \min_{\phi',\phi''}[\mathcal{C}(y_{t+1},\phi',\phi'') + \mathcal{C}(\mathbf{y}_{t+2:T},\phi'',\phi)], \\ &> \min_{\phi',\boldsymbol{\tau}'} \left[f^t_{\boldsymbol{\tau}'}(\phi')\right] + h(1) + \max_{s_i=1,\dots,n} h(s_i) + \min_{\phi',\phi''}[\mathcal{C}(y_{t+1},\phi',\phi'') + \mathcal{C}(\mathbf{y}_{t+2:T},\phi'',\phi)], \\ &= \min_{\phi',\phi'',\boldsymbol{\tau}'} \left[f^t_{\boldsymbol{\tau}'}(\phi') + \mathcal{C}(y_{t+1},\phi',\phi'') + \mathcal{C}(\mathbf{y}_{t+2:T},\phi'',\phi) + h(1) + \max_{s_i=1,\dots,n} h(s_i)\right], \end{aligned} \tag{5.13}$$

the last step is due to the cost on a single point, $\mathcal{C}(y_t,\phi',\phi'')$ only depending on ϕ''.

The final line can be shown to be greater than the cost of segmenting $\mathbf{y}_{1:T}$ with changepoints $(\boldsymbol{\tau}^*, t, t+1)$ (where $\boldsymbol{\tau}^*$ is the argmin of (5.13)). This is due to the penalty term in (5.13) being greater than the penalty for adding the two new segments should be

$$\begin{aligned} h(t+1-t)+h(T-(t+1)) &= h(1)+h(T-t-1), \\ &\leq h(1)+\max_{s_i=1,\dots,n} h(s_i), \end{aligned}$$

where the bound comes from evaluating the maximum value the function h can take for any feasible segment size.

Therefore the cost of segmenting $\mathbf{y}_{1:T}$ with changepoints $\boldsymbol{\tau}$ is always greater than the cost of segmenting $\mathbf{y}_{1:T}$ with changepoints $(\boldsymbol{\tau}^*, t, t+1)$ (where $\boldsymbol{\tau}^*$ is the argmin of (5.13)) and this holds for all $T > t+1$ and hence $\boldsymbol{\tau}$ can be pruned. ■

We can use Theorem 5 to reduce the size of $\hat{\mathcal{T}}_{t+1}$ before the cost functions are updated.

For the mBIC, condition (5.11) is simply to calculate. Recall that for the mBIC we have $h(s_i) = \log(n) + \frac{1}{2}\log(s_i)$ then $\max_{s_i=1,\dots,n} h(s_i) = \dfrac{3}{2}\log(n)$ as log is an increasing function. (5.11) then becomes

$$\min_{\boldsymbol{\phi}} f^t_{\boldsymbol{\tau}}(\boldsymbol{\phi}) > \min_{\boldsymbol{\phi}', \boldsymbol{\tau}'} \left[f^t_{\boldsymbol{\tau}'}(\boldsymbol{\phi}') \right] + \frac{5}{2}\log(n).$$

Using both functional and inequality based pruning we can restrict the set of candidate changepoint vectors that the dynamic program is run over. The pseudocode for the OPPL method with a penalty function that depends on the segment length is outlined in Algorithm 6. As in Chapter 4 the computational time of the algorithm depends on the size of the sets $\overset{*}{\mathcal{T}}_t$ and $\hat{\mathcal{T}}_{t+1}$. We explore the empirical sizes of these sets in the next section.

5.4 Comparing OPPL with the mBIC penalty to OPPL with the BIC penalty

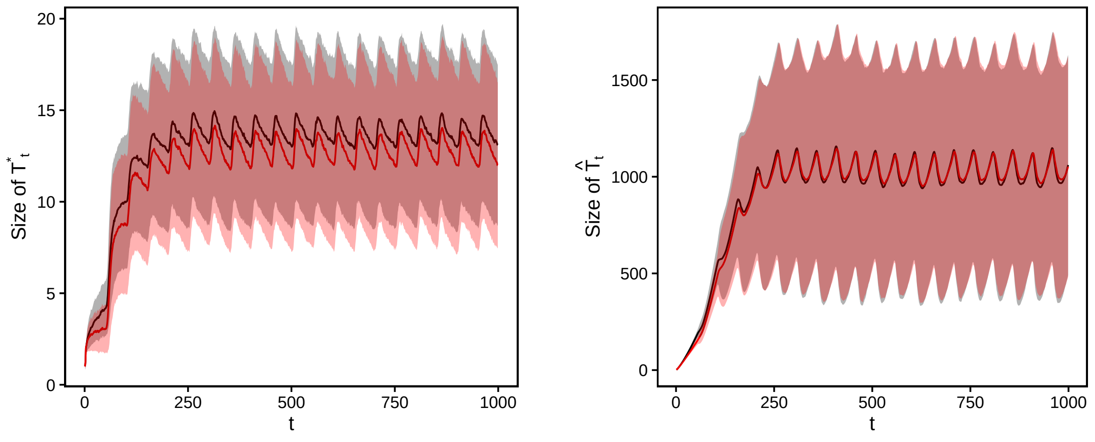
As an example of OPPL with a penalty that depends on the segment length we consider the mBIC where $h(s_i) = \log(n) + \frac{1}{2}\log(s_i)$. We compare the mBIC method to the standard OPPL method with the BIC penalty from Chapter 4. As OPPL involves adding 2 parameters (change in slope and changepoint location) with each additional segment, this corresponds to using $\beta = 2\log n$.

First we look at how the lengths of the sets $\overset{*}{\mathcal{T}}_t$ and $\hat{\mathcal{T}}_t$ change as the current time point of the algorithm, t, increases. This is done in Figure 5.1 for data simulated from a piecewise linear signal with

Algorithm 6: Algorithm for OPPL with a penalty that depends on the segment length

```
Input : Set of data of the form $\mathbf{y}_{1:n} = (y_1, \ldots, y_n)$,
        A cost function to be minimised, $\mathcal{C}$, dependent on the data and parameters $\phi_{\tau_i}$
        which correspond to the values fitted at the changepoints,
        A penalty function $h(s_i)$ which depends on the segment length $s_i = \tau_{i+1} - \tau_i$.
Let $n$ = length of data;
And set $\hat{\mathcal{T}}_1 = \{1\}$;
for $t = 1, \ldots, n$ do
    $f^t_{\emptyset}(\phi) = \min_{\phi'} \mathcal{C}(y_{1:t}, \phi', \phi)$;
    for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ do
        $f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'} \left\{ f^{\tau_k}_{\tau_1,\ldots,\tau_{k-1}}(\phi') + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + h(t-\tau_k) \right\}$;
    for $\boldsymbol{\tau} \in \hat{\mathcal{T}}_t$ do
        $Int^t_{\boldsymbol{\tau}} = \left\{ \phi : f^t_{\boldsymbol{\tau}}(\phi) = \min_{\boldsymbol{\tau}' \in \hat{\mathcal{T}}_t} f^t_{\boldsymbol{\tau}'}(\phi) \right\}$;
        $\overset{*}{\mathcal{T}}_t = \{\boldsymbol{\tau} : Int^t_{\boldsymbol{\tau}} \neq \emptyset\}$;
        $\hat{\mathcal{T}}_{t+1} = \hat{\mathcal{T}}_t \cup \left\{ (\boldsymbol{\tau}, t) : \boldsymbol{\tau} \in \overset{*}{\mathcal{T}}_t \right\}$;
    $\hat{\mathcal{T}}_{t+1} = \left\{ \boldsymbol{\tau} \in \hat{\mathcal{T}}_{t+1} : \min_{\phi} f^t_{\boldsymbol{\tau}}(\phi) \leq \min_{\phi', \boldsymbol{\tau}'} \left[ f^t_{\boldsymbol{\tau}'}(\phi') \right] + h(1) + \max_{s_i = 1,\ldots,n} h(s_i) \right\}$;
$f_{opt} = \min_{\boldsymbol{\tau}, \phi} f^n_{\boldsymbol{\tau}}(\phi)$;
$\boldsymbol{\tau}_{opt} = \arg\min_{\boldsymbol{\tau}} \left[ \min_{\phi} f^n_{\boldsymbol{\tau}}(\phi) \right]$;
Output: The optimal cost, $f_{opt}$, and the corresponding changepoint vector, $\boldsymbol{\tau}_{opt}$.
```

normal ($\mu = 0, \sigma^2 = 1$) noise and randomly drawn values at the changes ($\phi_{\tau_i} \sim \text{Normal}(0, 4^2)$). We compare the lengths of the sets for the mBIC penalty with the BIC penalty.



(a) (b)

Figure 5.1: Length of the sets $\overset{*}{\mathcal{T}}_t$ (a) and $\hat{\mathcal{T}}_t$ (b) for data with 19 true changepoints located at $\boldsymbol{\tau} = (50, 100, \ldots, 950)$. This is done for OPPL with the BIC (black) and mBIC (red) penalties. Averaged over 1000 data sets. Shaded regions show 1 standard deviation away from the mean on either side.

The length of $\overset{*}{\mathcal{T}}_t$ is empirically slightly less for the mBIC penalty, more importantly it is still empirically constant as t increases. Similarly, the length of $\hat{\mathcal{T}}_t$ also seems to remain constant as t increases and matches the length of the set for the BIC penalty. This suggests that the computational cost of the algorithm with the mBIC penalty should be similar to before.

To test this and compare the accuracy of the two alternative penalties we run the two methods on the same data as was used in Figures 4.6 and 4.7 from Chapter 4 . This data was simulated from a piecewise linear signal with normal ($\mu = 0, \sigma^2 = 1$) noise and randomly drawn values at the changes ($\phi_{\tau_i} \sim \text{Normal}(0, 4)$). This is done with data sets of increasing length $n = 50, 100, \ldots, 10000$. We then look at the same two scenarios as before; firstly with the segment size kept constant ($m = \frac{n}{50}$ true changepoints) in Figure 5.2 and then with the segment size increasing with the data length ($m = 19$ true changepoints) in Figure 5.3.

As before we compare the methods by looking at the computational time, the mean squared error (MSE), the true positive count and the false positive count. A true positive is defined as being a detected changepoint that is within ϵ time points either side of a true change. A false positive is defined a detected changepoint which is not a true positive. For the scenario analysed in Figure 5.2, where the segment size is constant, we choose $\epsilon = 5$. For the scenario analysed in Figure 5.3, where the segment size increases with n, we choose $\epsilon = 5 \log n$.

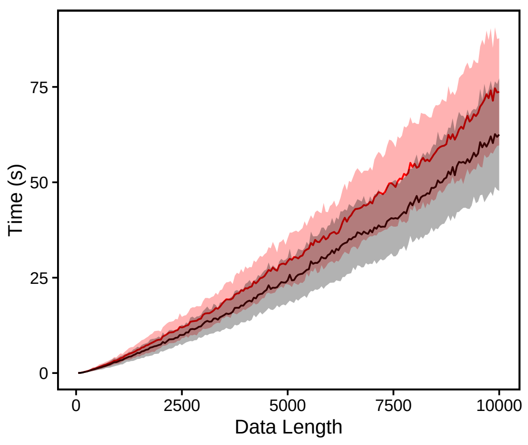
As can be seen in both Figures 5.2 and 5.3 the mBIC penalty does not seem to offer many advantages over the BIC penalty, often performing slightly worse. It also seems to take slightly more computational time. The mBIC penalises segmentations more heavily where the changes are not equally spaced. In both Figure 5.2 and 5.3 the true changepoints are equally spaced and hence should be favoured by both the BIC and the mBIC penalties. To explore this further we take the scenario where the number of changepoints is linearly increasing with the data (as was used in Figure 5.2) and also randomise the positions of the changes. We randomly sample $\frac{n}{50}$ changepoint locations from the set $[\![2, n-1]\!]$ without replacement. The results of this are shown in Figure 5.4. For the true positive error in Figure 5.4 we again choose $\epsilon = 5$.

As can be seen in Figure 5.4, the mBIC performs better when the changes are not equally spaced. Not only is the mean squared error less than for the BIC penalty, but also the computational time is lower.

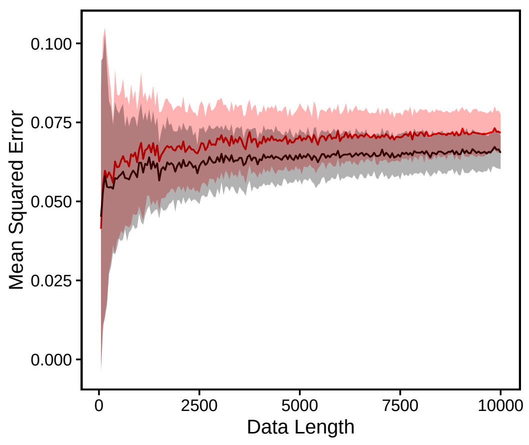
5.5 Discussion

We have adapted Optimal Partitioning for Piecewise Linear data (OPPL) to work with a more general class of penalties. Penalties which depend on the lengths of the fitted segments arise in a number of methods including the modified Bayesian information criterion (Zhang and Siegmund, 2007) and the minimum description length (Rissanen, 1989). Adaptation of OPPL to include these penalties is relatively straightforward, involving some slight adjustments to the rules used within the method to both update the cost functions and apply pruning techniques.

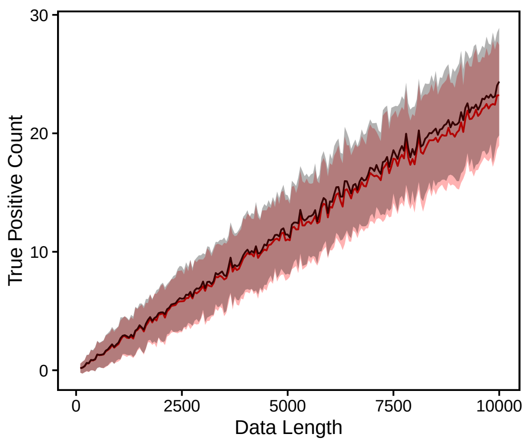
We have analysed the performance of the OPPL method with the mBIC penalty. We compared this against using the BIC penalty (which was shown to perform well in Chapter 4). One issue with using a more complex penalty is that it may have an adverse effect on the computational time of the algorithm. Whilst some evidence of this is apparent for some scenarios in the results shown in Section 5.4, the algorithm can still be seen to be computationally viable. One potential issue for using other penalty functions than the mBIC is that less pruning may be done depending on the value of $h(1) + \max_{s_i=1,\dots,n} h(s_i)$. Hence we can offer no guarantees that other penalty functions behave as well as the mBIC in terms of



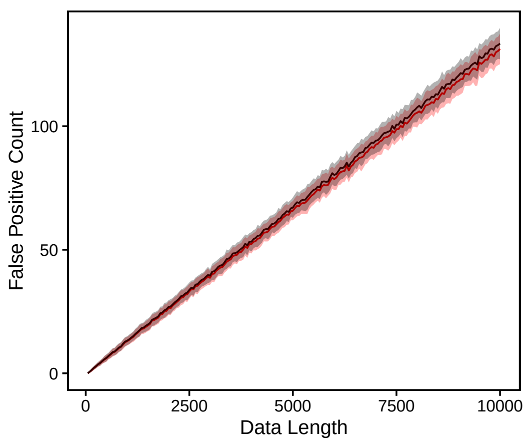
(a) Computational Time



(b) Mean Squared Error

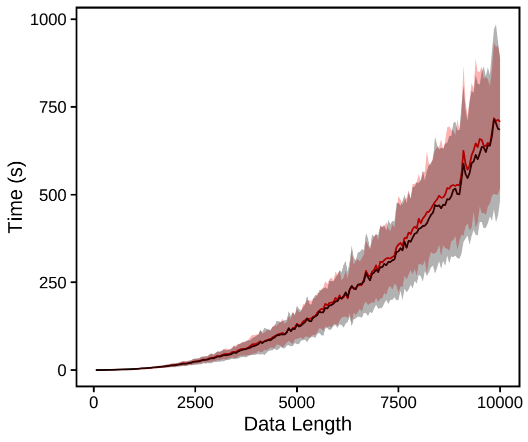


(c) True Positive Count

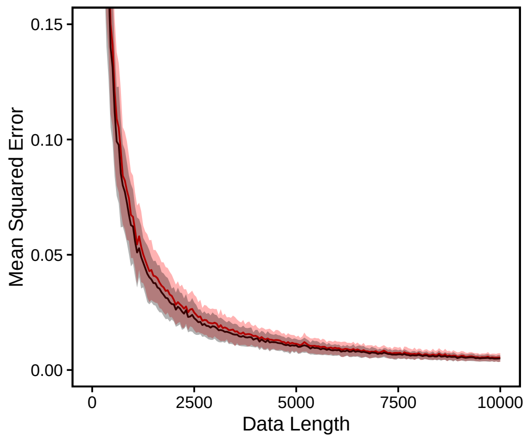


(d) False Positive Count

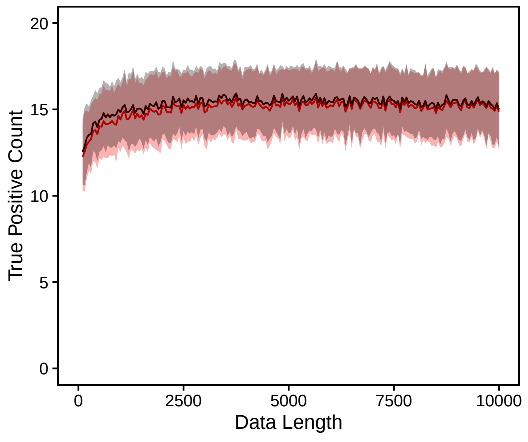
Figure 5.2: Comparison of computational run-time (a), mean squared error (b), true positive count (c), and false positive count (d). This is done for OPPL with the BIC penalty (black) and with the mBIC penalty (red) on data sets with $\frac{n}{50}$ true changepoints (where n is the length of the data set). Averaged over 100 data sets. Shaded regions show 1 standard deviation away from the mean on either side.



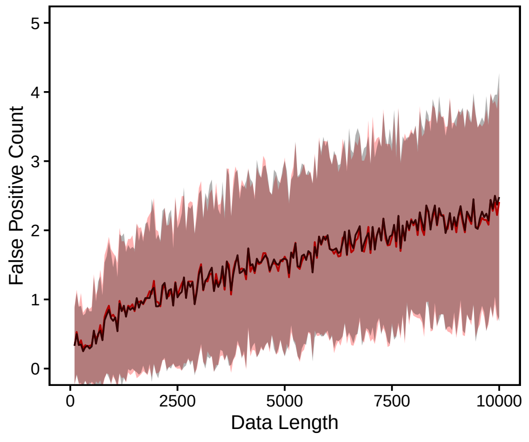
(a) Computational Time



(b) Mean Squared Error

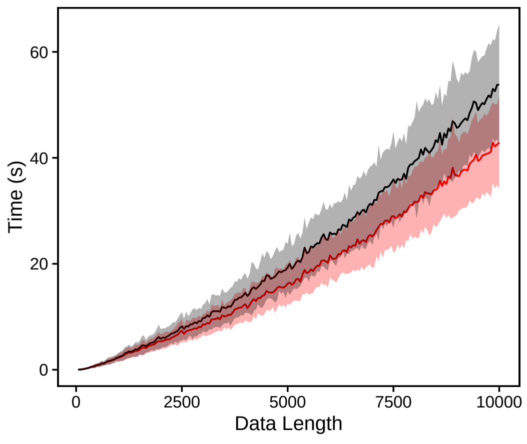


(c) True Positive Count

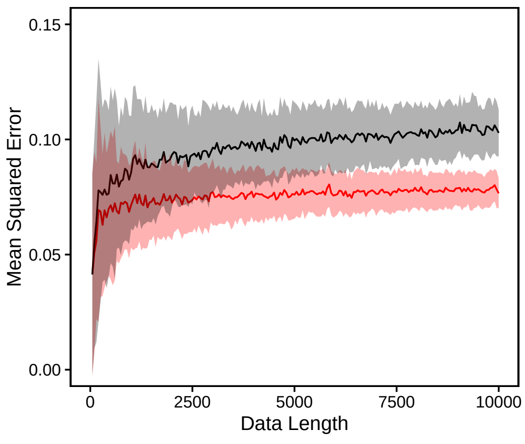


(d) False Positive Count

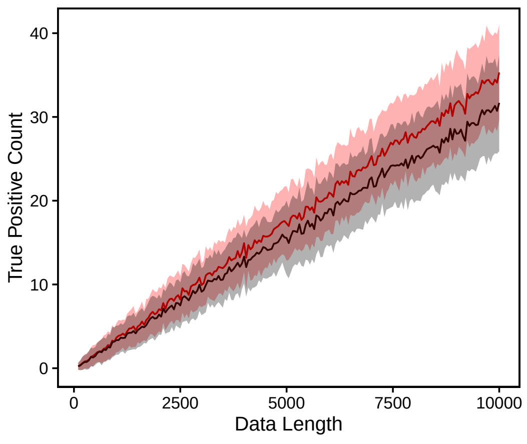
Figure 5.3: Comparison of computational run-time (a), mean squared error (b), true positive count (c), and false positive count (d). This is done for OPPL with the BIC penalty (black) and with the mBIC penalty (red) on data sets with 19 true changepoints. Averaged over 100 data sets. Shaded regions show 1 standard deviation away from the mean on either side.



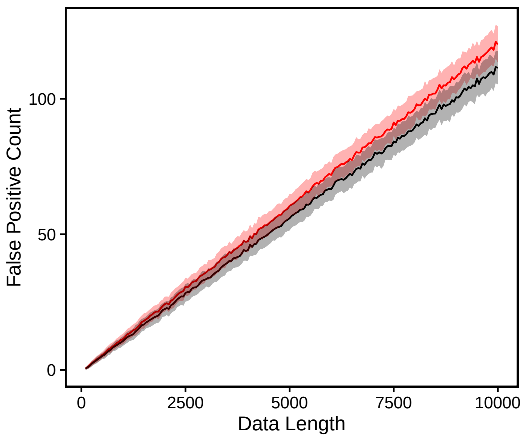
(a) Computational Time



(b) Mean Squared Error



(c) True Positive Count



(d) False Positive Count

Figure 5.4: Comparison of computational run-time (a), mean squared error (b), true positive count (c), and false positive count (d). This is done for OPPL with the BIC penalty (black) and with the mBIC penalty (red) on data sets with $\frac{n}{50}$ true changepoints (where n is the length of the data set) which are located at positions drawn from $1, \ldots, n-1$. Averaged over 100 data sets. Shaded regions show 1 standard deviation away from the mean on either side.

the computationally time taken.

We also observed that change of penalty also impacted the accuracy of the algorithm. For scenarios where the true changes are equally spaced the mBIC performed slightly worse than the BIC. However, when the true changepoints are not equally spaced, as in Figure 5.4, an increase in accuracy was observed, with both the mean squared error decreasing and true positive count increasing (although the later is probably due to the number of changes detected also increasing).

We suggest that the mBIC provides a viable alternative to the BIC for changepoint detection. When the changes are known to be not equally spaced in the data then it is a natural choice. If the distribution of the changes in the data is unknown then more care needs to be taken. Further, our formulation allows a wider class of penalties which depend on the segment length to be applied, offering the user much more versatility when applying the OPPL method to their data set.

Chapter 6

Discussion

6.1 Summary of the Contribution of the Thesis

In this thesis we have explored the area of changepoint detection and presented some novel ideas of our own.

In Chapter 2 we looked at the vast body of literature in changepoint detection. Briefly we discussed techniques for detecting a single change; namely the likelihood-ratio test, the penalised likelihood and taking a Bayesian approach. We then looked at the more complex problem of detecting multiple unknown changepoints. Often the single changepoint detection methods can be extended. For example the likelihood-ratio test can be used as part of the Binary Segmentation algorithm (Scott and Knott, 1974), penalised likelihood is used as part of algorithms such as Optimal Partitioning (Jackson et al., 2005) and PELT (Killick et al., 2012), and Bayesian methods can be extended to the multiple changepoint case as well (Stephens, 1994; Barry and Hartigan, 1992). Other methods for changepoint detection include nonparametric approaches (Zou et al., 2014; Haynes et al., 2016), SMUCE (Frick et al., 2014), and the fused LASSO (Tibshirani et al., 2005).

In this thesis we have concentrated primarily on fitting a cost function that needs to be minimised and methods which use dynamic programming to perform this minimisation. In Section 3.2.1 we introduced two ways of formulating the problem; the *constrained minimisation problem* and the *penalised minimisation problem.* They differ, as whilst the constrained minimisation problem places a limit on the total number of changepoints to search for, the penalised minimisation problem penalises each additional changepoint added to the model. Dynamic programming algorithms for solving the two minimisation problems have been developed. The constrained minimisation problem can be solved by the Segment Neighbourhood Search (SNS, Auger and Lawrence (1989)) and the penalised minimisation problem can be solved by Optimal Partitioning (OP, Jackson et al. (2005)).

Both OP and SNS are computationally slow and in Sections 3.3.2 and 3.4.2 we introduced *inequality based pruning* and *functional pruning*. Inequality based pruning is applied to OP in Killick et al. (2012) to form the PELT algorithm and functional pruning is applied to SNS in Rigaill (2015) to form the pDPA algorithm. In Sections 3.5.1 and 3.5.2 we introduced two new algorithms; Functional Pruning Optimal Partitioning (FPOP) and Segment Neighbourhood with Inequality Pruning (SNIP). FPOP applies functional pruning to the Optimal Partitioning algorithm whilst SNIP applies inequality based pruning to Segment Neighbourhood Search.

The two types of pruning rely on different conditions on the cost function in order to be used. These conditions were outlined in Section 3.2.2. The condition on using functional pruning is stronger than that on inequality based pruning, however we have shown in Theorem 2 that the pruning is stronger. Theorem 2 states that any point that is pruned by inequality based pruning will also have been pruned by functional pruning. What this means is that, in terms of the pruning, FPOP should always be at least as efficient as PELT and pDPA should always be at least as efficient as SNIP (if the condition on the cost function for functional pruning holds). This is discussed in Chakar (2015) where the pDPA algorithm is chosen over SNIP as, for the cost function they consider, the condition to apply functional pruning holds.

From the simulations presented in Section 3.7, FPOP performs well in terms of efficiency. It outperforms both PELT and Binary Segmentation when the density of the true changepoints in the data is high. Further, we show that the accuracy of FPOP performs well when compared to WBS, Binary Segmentation and SMUCE.

In Chapter 4 we extended the ideas of functional and inequality based pruning in dynamic programming algorithms for changepoint detection to the problem of changepoint detection in continuous linear piecewise models. Detecting a change in trend with continuity is computationally expensive and hence many methods in the literature are heuristics (Horner and Beauchamp, 1996; Kim et al., 2000; Goldberg et al., 2014). Ideally an efficient, exact algorithm is required.

In Section 4.3 we developed a dynamic programming algorithm for minimising a penalised cost function with the condition that the fitted value at the end of a segment is the same as the fitted value at the start of the next segment. The dynamic program works by minimising over the vector of changepoints prior to the current time point. Because of this, it is computationally more expensive than complete enumeration. We applied functional and inequality based pruning methods to it to reduce the computational time and, by looking at the number of candidate changepoint vectors stored, show that empirically the algorithm is $\mathcal{O}(n^2)$ in computational time. We called this algorithm Optimal Partitioning for Piecewise Linear data (OPPL).

Comparisons between OPPL and other methods for continuous piecewise linear changepoint detection were presented in Section 4.4.2. One method we compared against was Trend Filtering. Trend Filtering is a LASSO based method that fits a residual sum of squares cost function with an l_1 penalty term. We have shown empirically that, while Trend Filtering performs well, it forces the user to make a trade off between fitting the function well (i.e. obtaining a low mean squared error) and estimating the correct number of changepoints. OPPL performs better in this respect. We observed that by using the BIC penalty choice both the continuous piecewise linear function and the number of changepoints can be estimated accurately.

Whilst the BIC penalty performs well, for certain underlying functions other penalty choices may be preferable. Often alternative penalty functions depend on the segment length, for example the mBIC (Zhang and Siegmund, 2007) and the MDL (Rissanen, 1989). We have extended OPPL to penalties that depend on the segment length in Chapter 5. We modified both the dynamic programming algorithm and the pruning applied to it to work with more complex penalties. In Section 5.4 we looked in greater detail at the mBIC penalty, comparing against using the BIC penalty from Chapter 4 and showing that the two penalties perform similarly. One scenario where they differ is when the changepoints are unevenly spaced in the data. Here a notable improvement is obtained by using the mBIC penalty.

In conclusion, we have developed three novel pruned dynamic programming methods for changepoint detection; FPOP, SNIP and OPPL. For changepoint detection in univariate independent data, FPOP has been shown to be competitive in both efficiency and accuracy when compared to other methods in the literature. SNIP, whilst inferior in terms of efficiency, provides a pruned alternative to SNS in certain scenarios where the condition for functional pruning does not hold. Pruned dynamic programming algorithms are extended further to the continuous piecewise linear case and our method OPPL is shown to perform well at estimating these functions and be adaptatable to different penalty choices.

6.2 Future Directions

The methods developed in this thesis can be extended further. In this section we will discuss what we feel is a few key directions which related research could take in the future. First we will discuss issues FPOP has when the cost function is more complicated than was used in Chapter 3, for example when detecting changes in more than one parameter. We will then discuss how OPPL can be extended from the continuous piecewise linear case to other models with dependence between the segments. Lastly we discuss how confidence bounds can potentially be added to both FPOP and OPPL.

6.2.1 Expanding FPOP to a more complex cost functions

In Chapter 3 we introduced the FPOP algorithm. FPOP requires the condition C1 to hold which is as follows

C1 The cost function satisfies

$$\mathcal{C}(\mathbf{y}_{s+1:t}) = \min_{\mu} \sum_{i=s+1}^{t} \gamma(y_i, \mu),$$

for some function $\gamma(\cdot,\cdot)$, with parameter μ.

As mentioned before many cost functions satisfy this condition of being additive in the data, for example the negative log-likelihood for the normal distribution. However this is not the only restriction on the cost function. In Section 3.5.1, we derived the following cost function for the minimal cost up to time t, conditional on the last changepoint being at $\tau \leq t-1$ and the last segment having parameter μ

$$Cost_t^{\tau}(\mu) = F(\tau) + \beta + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu). \tag{6.1}$$

FPOP depends on the form of this cost function in order to run efficiently. The functions $Cost_t^{\tau}(\mu)$ need to be able to be stored simply and the intersections between functions for different values of τ computed efficiently. For a simple cost function this is easy. For example consider the cost function for the negative log-likelihood of a normal distribution, given in equation (3.3). The related function $Cost_t^{\tau}(\mu)$ is just a quadratic in μ and hence the coefficients can be stored. By calculating the intersections of the $Cost_t^\tau(\mu)$ functions we can calculate the intervals Set_t^τ which are required for functional pruning. For the case where $Cost_t^\tau(\mu)$ is quadratic then this is again simple.

For more complicated cost functions these calculations can be computationally challenging. As an example consider if a change occurs in more than one variable, in particular a change in both the mean and the variance of normally distributed data. In this case our cost function depends on both the segment mean μ and the segment standard deviation σ, which are both unknown

$$\gamma(y_i, \mu, \sigma) = \frac{1}{2\sigma^2}\left(y_i - \mu\right)^2.$$

The cost function defined in equation (6.1) we then need to be modified to deal with the 2-dimensional parameter

$$Cost_t^\tau(\mu, \sigma) = F(\tau) + \beta + \sum_{i=\tau+1}^{t} \gamma(y_i, \mu, \sigma).$$

These new cost functions are quadratics in μ and σ so are still easy to store by storing the coefficients. The problem that arises is the calculation of Set_t^τ. Set_t^τ was defined in Section 3.5.1 as the subset of the parameter space for which τ is the optimal last changepoint. This parameter space is now 2-dimensional and hence the sets Set_t^τ are also 2-dimensional. Storing and updating these 2-dimensional sets is a computational hurdle which will need to be addressed if FPOP is to be extended to detecting changes in two variables. One potential solution may be approximate Set_t^τ by a simpler set, for example as a rectangle formed by considering each variable individually. As long as this approximation to Set_t^τ contains Set_t^τ within it, then the algorithm will retain its exactness. However, the worse the approximation the less efficient the algorithm will be which may lead FPOP becoming inferior, in terms of efficiency, when compared to rival methods.

6.2.2 Expanding OPPL to a larger class of models

OPPL can be extended to a wider class of problems than just continuous piecewise linear changepoint detection. This can be done in one of two ways; firstly the model fitted to individual segments can extended, and secondly the condition of continuity at the changepoints can be modified.

The model fitted to a each segment, and the related cost function, can be changed subject to a few conditions. For example, piecewise quadratic data can be modelled by changing the cost function as follows

$$\mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) = \min_{A\in\mathbb{R}} \left\{ \frac{1}{2\sigma^2} \sum_{j=\tau_i+1}^{\tau_{i+1}} \left[y_j - \phi_{\tau_i} - \left(\frac{\phi_{\tau_{i+1}} - \phi_{\tau_i}}{\tau_{i+1} - \tau_i} - A(\tau_{i+1} - \tau_i) \right) (j - \tau_i) - A(j - \tau_i)^2 \right]^2 \right\},$$

this corresponds fitting a quadratic such that the fitted value at τ_i is ϕ_{τ_i} and the fitted value at τ_{i+1} is $\phi_{\tau_{i+1}}$. The parameter A is minimised over and corresponds to the curvature of the fitted quadratic. OPPL then proceeds in the same way as before and both functional and inequality based pruning can still be used. Other similar models can be fitted provided that they can be formulated in terms of the fitted values at the start and end of the segment and other parameters which can be minimised over.

OPPL can also be extended by modifying the condition at the segment boundary that the fitted value at the start of a segment is equal to the fitted value at the end of the previous segment. For example we can fit a piecewise constant model with a boundary condition that the mean of neighbouring segments is no more than δ apart. We can formulate this in the same way as in equation (4.4) by modifying our cost function

$$\mathcal{C}(\mathbf{y}_{\tau_i+1:\tau_{i+1}}, \phi_{\tau_i}, \phi_{\tau_{i+1}}) = \begin{cases} \frac{1}{2\sigma^2} \sum_{j=\tau_i+1}^{\tau_{i+1}} \left(y_j - \phi_{\tau_{i+1}} \right)^2, & \text{if } \phi_{\tau_{i+1}} \in [\phi_{\tau_i} - \delta, \phi_{\tau_i} + \delta], \\ \infty, & \text{otherwise.} \end{cases} \tag{6.2}$$

In equation (6.2), $\phi_{\tau_{i+1}}$ is the mean of the segment finishing at τ_{i+1} and we condition it to be within δ of the previous segment mean ϕ_{τ_i}. The functions $f^t_{\boldsymbol{\tau}}(\phi)$, as defined in equation (4.6) are then still quadratics and thus easy to manipulate. Functional pruning will still work this cost function, however care will have to be taken with inequality pruning as part of the proof depends on $\mathcal{C}(y_t, \phi', \phi'')$ depending only on ϕ'' and this is no longer the case and will have to be reconsidered.

One area in which it seems natural to extend OPPL to is to fitting a spline with unknown knots. For example consider fitting a piecewise quadratic with unknown changepoints subject to the condition that at the changepoints we have continuity in both the function and the first derivative. The dependence between segments now depends on two parameters; the fitted value at the changepoint and the derivative of the fitted function at the changepoint. What this means for using OPPL is that we now need to store functions in two variables. As mentioned in Section 6.2.1, dealing with multi-dimensional functions leads to problems when considering the sets over which these functions are optimal. We now have 2-dimensional sets which can be difficult to both compute and store. Developing techniques to deal with these 2-dimensional sets is more relevant here than in Section 6.2.1, as slight losses in the efficiency will matter less as there is no rival exact method.

6.2.3 Confidence Bounds

As part of their method SMUCE, Frick et al. (2014) introduce a way of constructing confidence intervals for the changepoint locations and the signal itself. This is done for a change in mean problem (or equivalent change in a single variable). These intervals are useful in determining the variability in the model fitted and currently none of the methods developed in this thesis give anything similar. They define a confidence set, denoted by $\mathcal{C}(q)$, on the space of piecewise constant functions

$$\mathcal{C}(q) = \{\vartheta \in S : \#J(\vartheta) = \hat{K}(q), \text{ and } T_n(Y, \vartheta) \leq q\}.$$

This set is all piecewise constant functions, ϑ, such that the number of changepoints in the function, $\#J(\vartheta)$, is equal to the optimal number of changepoints found by SMUCE, $\hat{K}(q)$, and which satisfy $T_n(Y, \vartheta) \leq q$ for the SMUCE multiscale test statistic, T, and a chosen threshold, q. Using the set $\mathcal{C}(q)$ confidence intervals on the changepoint locations and fitted values in the segments can be obtained.

In a discussion piece on Frick et al. (2014) (published alongside the paper), Rebecca Killick states that this concept of confidence intervals can be extended to other algorithms, in particular PELT (Killick et al., 2012). She suggests that at each time point PELT retains the potential changepoint locations whose costs are within the penalty, β, of optimal changepoints cost. These candidate changepoints could then be used to define a confidence set.

This approach could also be applied to the FPOP algorithm. In FPOP we also store the candidate changepoint locations which have costs within β of the optimum cost. Further, as the cost functions in FPOP depend on the parameter value μ this offers more information on how the cost changes as μ changes. This could be exploited to give confidence bands on the segment means.

Similarly, OPPL retains candidate changepoint vectors. This may actual make it easier to construct confidence sets as the best segmentations can be outputted at the end of the algorithm with no need for passing back through the algorithm again. Again as the functions depend on the parameter, ϕ, this information may be able to be exploited in the calculation of confidence bounds.

Chapter 7

Appendix

Appendix A

Further Simulation Results from Chapter 3

A.1 Short Description of the WBS Benchmark Scenarios

The 6 scenarios considered in sub-section 3.7.4 are :

- (1) block profiles of length 2048. Changepoints are at 205, 267, 308, 472, 512, 820, 902, 1332, 1557, 1598, 1659. Means are 0, 14.64, -3.66, 7.32, -7.32, 10.98, -4.39, 3.29, 19.03,7.68, 15.37, 0. The standard deviation is 10.
- (2) fms profiles of length 497. Changepoints are at 139, 226, 243, 300, 309, 333. Means are -0.18, 0.08, 1.07, -0.53, 0.16, -0.69,-0.16. The standard deviation is 0.3.
- (2') fms', profiles of length 497. Changepoints are at 139, 226, 243, 300, 309, 333. Means are -0.18, 0.08, 1.07, -0.53, 0.16, -0.69,-0.16. The standard deviation is 0.2.
- (3) mix profiles of length 560. Changepoints are at 11, 21, 41, 61, 91, 121, 161, 201, 251, 301, 361, 421, 491. Means are 7, -7, 6, -6, 5, -5, 4, -4, 3, -3, 2, -2, 1, -1. The standard deviation is 4.
- (4) teeth10 profiles of length 140. Changepoints are at 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111, 121, 131. Means are 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1. The standard deviation is 0.4.
- (5) stairs 10, profiles of length 150. Changepoints are at 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111,

121, 131, 141. Means are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15. The standard deviation is 0.3.

A.2 Accuracy Results for WBS Benchmark Scenarios

Performances of WBS, FPOP and SMUCE on the scenarios given in Appendix A.1 over 50 replications. Quantities of interest are; $\hat{K}$: proportion of true K recovery, MSE: average mean squared error, exact TP: average number of exactly recovered changepoints, exact FP: average number of false changepoints and BkpE: average breakpoint error (see R package).

Table A.1: Scenario 1

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.52	0.0260	4.40	7.15	1.12
WBS (BIC)	**0.54**	**0.0258**	**4.41**	7.16	**1.11**
WBS (mBIC)	0.41	0.0273	4.35	**6.99**	1.23
FPOP ($3\log(n)$)	0.20	0.0289	4.33	**6.65**	1.45
FPOP ($2\log(n)$)	<u>**0.57**</u>	<u>**0.0244**</u>	4.50	7.10	<u>**1.06**</u>
FPOP ($\log(n)$)	0.05	0.0445	<u>**4.52**</u>	12.56	5.92
SMUCE (0.55)	**0.02**	**0.0365**	**4.01**	6.64	**1.94**
SMUCE (0.45)	0.01	0.0403	3.87	<u>**6.63**</u>	2.13
SMUCE (0.35)	0.01	0.0426	3.82	6.55	2.26

Table A.2: Scenario 2

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.95	0.0412	4.10	2.93	<u>**0.27**</u>
WBS (BIC)	<u>**0.96**</u>	**0.0409**	**4.11**	2.94	<u>**0.27**</u>
WBS (mBIC)	0.95	0.0429	4.07	**2.89**	0.29
FPOP ($3\log(n)$)	0.87	0.0471	4.06	<u>**2.70**</u>	0.44
FPOP ($2\log(n)$)	**0.94**	<u>**0.0381**</u>	<u>**4.20**</u>	2.84	**0.30**
FPOP ($\log(n)$)	0.14	0.0921	4.18	6.58	3.97
SMUCE (0.55)	**0.76**	**0.0571**	**3.90**	**2.93**	**0.62**
SMUCE (0.45)	0.70	0.0630	3.80	2.94	0.70
SMUCE (0.35)	0.64	0.0681	3.70	2.95	0.78

Table A.3: Scenario 2'

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.96	0.0338	4.90	2.15	0.12
WBS (BIC)	0.95	0.0335	**4.91**	2.14	0.12
WBS (mBIC)	**0.97**	**0.0329**	**4.91**	**2.12**	**0.09**
FPOP ($3\log(n)$)	<u>**0.99**</u>	<u>**0.0297**</u>	<u>**5.00**</u>	<u>**2.01**</u>	<u>**0.08**</u>
FPOP ($2\log(n)$)	0.95	0.0307	<u>**5.00**</u>	2.06	0.12
FPOP ($\log(n)$)	0.15	0.0821	4.97	5.59	3.63
SMUCE (0.55)	0.95	0.0300	<u>**5.00**</u>	2.05	0.12
SMUCE (0.45)	0.97	0.0298	<u>**5.00**</u>	2.03	0.10
SMUCE (0.35)	**0.98**	<u>**0.0297**</u>	<u>**5.00**</u>	**2.02**	**0.09**

Table A.4: Scenario 3

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.29	**0.0966**	**7.30**	5.59	1.94
WBS (BIC)	<u>**0.32**</u>	0.0967	7.27	5.70	<u>**1.89**</u>
WBS (mBIC)	0.15	0.1040	7.20	**5.07**	2.34
FPOP ($3\log(n)$)	0.06	0.1190	7.02	<u>**4.56**</u>	2.92
FPOP ($2\log(n)$)	<u>**0.32**</u>	<u>**0.0965**</u>	7.32	5.56	**1.97**
FPOP ($\log(n)$)	0.09	0.1320	<u>**7.43**</u>	9.84	4.57
SMUCE (0.55)	**0.08**	**0.1150**	**6.95**	5.51	**2.42**
SMUCE (0.45)	0.05	0.1250	6.86	5.40	2.66
SMUCE (0.35)	0.02	0.1360	6.76	**5.25**	2.93

Table A.5: Scenario 4

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.74	0.370	9.21	4.44	1.84
WBS (BIC)	<u>**0.75**</u>	**0.358**	**9.33**	4.46	<u>**1.75**</u>
WBS (mBIC)	0.43	0.869	5.43	**2.93**	6.43
FPOP ($3\log(n)$)	0.18	0.924	5.27	<u>**2.73**</u>	6.59
FPOP ($2\log(n)$)	**0.62**	0.424	8.93	4.03	**2.20**
FPOP ($\log(n)$)	0.38	<u>**0.356**</u>	<u>**9.60**</u>	5.79	2.54
SMUCE (0.55)	**0.02**	**1.19**	**3.15**	5.47	**7.86**
SMUCE (0.45)	0.01	1.23	2.84	5.26	8.25
SMUCE (0.35)	0.01	1.29	2.39	**5.10**	8.84

Table A.6: Scenario 5

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.59	**0.260**	**12.19**	3.35	1.02
WBS (BIC)	0.58	0.261	**12.19**	3.37	1.05
WBS (mBIC)	**0.62**	0.262	12.15	**3.32**	**0.98**
FPOP ($3\log(n)$)	0.94	0.229	12.47	**<u>2.48</u>**	**<u>0.44</u>**
FPOP ($2\log(n)$)	**<u>0.95</u>**	**<u>0.215</u>**	**<u>12.52</u>**	2.53	**<u>0.44</u>**
FPOP ($\log(n)$)	0.39	0.263	12.50	3.79	1.66
SMUCE (0.55)	**0.15**	**1.17**	**8.43**	**4.58**	**4.02**
SMUCE (0.45)	0.10	1.33	7.80	4.85	4.54
SMUCE (0.35)	0.07	1.45	7.44	4.90	4.89

A.3 Accuracy Results for Speed Simulations

Performances of WBS, FPOP and SMUCE on the speed simulation data from Section 3.7.2 over 50 replications. Quantities of interest are; $\hat{K}$: proportion of true K recovery, MSE: average mean squared error, exact TP: average number of exactly recovered changepoints, exact FP: average number of false changepoints and BkpE: average breakpoint error (see R package).

Table A.7: Data length $n = 2 \times 10^5$ with $K = 10$ true changepoints

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	**<u>1.00</u>**	**0.000157**	**<u>5.70</u>**	**<u>4.30</u>**	**<u>0.00</u>**
WBS (BIC)	**<u>1.00</u>**	0.000160	5.66	4.34	**<u>0.00</u>**
WBS (mBIC)	**<u>1.00</u>**	0.000161	5.64	4.36	**<u>0.00</u>**
FPOP $(3\log(n))$	**<u>1.00</u>**	**<u>0.000156</u>**	5.64	**4.36**	**<u>0.00</u>**
FPOP $(2\log(n))$	**<u>1.00</u>**	**<u>0.000156</u>**	5.64	**4.36**	**<u>0.00</u>**
FPOP $(\log(n))$	0.00	0.000655	**5.66**	11.88	7.54
SMUCE (0.55)	0.86	0.000158	**5.64**	4.50	0.14
SMUCE (0.45)	0.94	**<u>0.000156</u>**	**5.64**	4.42	0.06
SMUCE (0.35)	**0.96**	**<u>0.000156</u>**	**5.64**	**4.40**	**0.04**

Table A.8: Data length $n = 2 \times 10^5$ with $K = 50$ true changepoints

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.54	**0.00106**	**30.36**	19.60	1.45
WBS (BIC)	**0.56**	0.00109	30.28	**19.34**	**1.12**
WBS (mBIC)	0.46	0.00112	30.08	19.48	1.29
FPOP $(3\log(n))$	0.72	0.000879	31.58	**<u>17.86</u>**	0.62
FPOP $(2\log(n))$	**<u>0.84</u>**	**<u>0.000845</u>**	31.76	17.92	**<u>0.39</u>**
FPOP $(\log(n))$	0.00	0.00143	**<u>31.80</u>**	26.80	8.97
SMUCE (0.55)	0.68	**0.000880**	**31.58**	18.06	0.65
SMUCE (0.45)	0.70	**0.000880**	**31.58**	18.04	0.62
SMUCE (0.35)	**0.72**	**0.000880**	**31.58**	**18.02**	**0.59**

Table A.9: Data length $n = 2 \times 10^5$ with $K = 100$ true changepoints

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.14	**0.00289**	57.44	41.08	**5.18**
WBS (BIC)	**0.16**	0.00292	57.22	41.78	5.82
WBS (mBIC)	0.14	0.00292	**57.88**	**40.70**	5.37
FPOP ($3\log(n)$)	0.38	0.00187	61.76	<u>**36.44**</u>	2.06
FPOP ($2\log(n)$)	<u>**0.58**</u>	<u>**0.00178**</u>	62.22	36.70	<u>**1.38**</u>
FPOP ($\log(n)$)	0.04	0.00232	<u>**62.44**</u>	45.50	9.58
SMUCE (0.55)	**0.36**	**0.00189**	**61.70**	**36.92**	**2.01**
SMUCE (0.45)	**0.36**	0.00191	61.60	36.96	2.13
SMUCE (0.35)	**0.36**	0.00191	61.60	36.96	2.11

Table A.10: Data length $n = 2 \times 10^5$ with $K = 500$ true changepoints

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	0.00	**0.0173**	264.94	230.38	84.81
WBS (BIC)	**0.04**	**0.0173**	**265.20**	229.88	83.55
WBS (mBIC)	0.02	0.0178	261.92	**228.96**	**82.51**
FPOP ($3\log(n)$)	0.00	0.0118	292.64	<u>**166.12**</u>	45.15
FPOP ($2\log(n)$)	0.00	<u>**0.0100**</u>	301.14	171.12	32.35
FPOP ($\log(n)$)	**0.08**	0.0985	<u>**310.52**</u>	190.86	<u>**32.02**</u>
SMUCE (0.55)	<u>**0.00**</u>	**0.0138**	**286.34**	176.30	**48.17**
SMUCE (0.45)	<u>**0.00**</u>	0.0142	284.78	176.42	49.90
SMUCE (0.35)	<u>**0.00**</u>	0.0147	283.12	**176.26**	51.83

Table A.11: Data length $n = 2 \times 10^5$ with $K = 1000$ true changepoints

	$\hat{K}$	MSE	exact TP	exact FP	BkpE
WBS (sSIC)	$\underline{\mathbf{0.00}}$	0.0356	498.14	448.14	211.32
WBS (BIC)	$\underline{\mathbf{0.00}}$	**0.0355**	**499.12**	450.16	213.70
WBS (mBIC)	$\underline{\mathbf{0.00}}$	0.0367	495.04	$\underline{\mathbf{441.60}}$	**207.83**
FPOP ($3\log(n)$)	$\underline{\mathbf{0.00}}$	0.0297	537.54	**300.38**	172.82
FPOP ($2\log(n)$)	$\underline{\mathbf{0.00}}$	0.0222	572.26	320.98	120.18
FPOP ($\log(n)$)	$\underline{\mathbf{0.00}}$	$\underline{\mathbf{0.0194}}$	$\underline{\mathbf{605.82}}$	363.10	$\underline{\mathbf{86.42}}$
SMUCE (0.55)	$\underline{\mathbf{0.00}}$	**0.0398**	**502.74**	338.88	**196.30**
SMUCE (0.45)	$\underline{\mathbf{0.00}}$	0.0416	497.04	338.76	202.65
SMUCE (0.35)	$\underline{\mathbf{0.00}}$	0.0443	490.00	**337.82**	211.98

Appendix B

Coefficient updating for OPPL using a Normal likelihood cost function

In Section 4.3 (equation 4.6) we define a function, $f^t_{\boldsymbol{\tau}}(\phi)$, as the minimum cost of segmenting $\mathbf{y}_{1:t}$ with changepoints at $\boldsymbol{\tau} = \tau_1, \dots, \tau_k$ and fitted value $\phi_t = \phi$ at time t. We then derived a recursion for these functions as follows

$$f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'} \left\{ f^{\tau_k}_{\tau_1,\dots,\tau_{k-1}}(\phi') + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + \beta \right\}. \tag{B.1}$$

As mentioned in Section 4.3 if we use the negative normal log-likelihood as our cost function as in (5.2) then the functions $f^t_{\boldsymbol{\tau}}(\phi)$ are quadratics in ϕ. This can be seen as $f^t_{\boldsymbol{\tau}}(\phi)$ is just the sum of the costs for each segment with $\phi_0, \phi_1, \dots, \phi_k$ minimised out and the individual segment costs are quadratic. With this in mind we denote $f^t_{\boldsymbol{\tau}}(\phi)$ as follows

$$f^t_{\boldsymbol{\tau}}(\phi) = a^t_{\boldsymbol{\tau}} + b^t_{\boldsymbol{\tau}}\phi + c^t_{\boldsymbol{\tau}}\phi^2, \tag{B.2}$$

for some constants $a^t_{\boldsymbol{\tau}}$, $b^t_{\boldsymbol{\tau}}$ and $c^t_{\boldsymbol{\tau}}$. We then wish to calculate these coeffcients by updating the coefficients that make up $f^{\tau_k}_{\tau_1,\dots,\tau_{k-1}}(\phi')$ using (B.1). To do this we need to write the cost for the segment from $\tau_k + 1$ to t in quadratic form. Defining the length of the segment as $s = t - \tau_k$ this cost can be written as

$$\begin{aligned}\mathcal{C}(\mathbf{y}_{\tau_k+1:t}, \phi', \phi) =& \frac{(s+1)(2s+1)}{12s\sigma^2}\phi^2 + \left(\frac{(s+1)}{2\sigma^2} - \frac{(s+1)(2s+1)}{6s\sigma^2}\right)\phi'\phi \\ &- \left(\frac{1}{s\sigma^2}\sum y_j(j-\tau_k)\right)\phi + \left(\frac{s}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum y_i^2\right) \\ &+ \left(\frac{1}{s\sigma^2}\sum y_j(j-\tau_k) - \frac{1}{\sigma^2}\sum y_i\right)\phi' + \frac{(s-1)(2s-1)}{12s\sigma^2}\phi'^2.\end{aligned} \tag{B.3}$$

Then subbing (B.3) into (B.1) and minimising out ϕ' we can get formula for the updating the coefficients of the quadratic $f^t_{\boldsymbol{\tau}}(\phi)$

$$\begin{aligned}
a^t_{\boldsymbol{\tau}} &= A - \frac{B^2}{4\left(a^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + F\right)},\\
b^t_{\boldsymbol{\tau}} &= C - \frac{\left(b^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + E\right)B}{2\left(a^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + F\right)},\\
c^t_{\boldsymbol{\tau}} &= c^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + D - \frac{\left(b^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + E\right)^2}{4\left(a^{\tau_k}_{(\tau_1,\ldots,\tau_{k-1})} + F\right)} + \beta.
\end{aligned} \tag{B.4}$$

Where the quantities A, B, C, D, E and F come from simplifying equation (B.3) as $A\phi^2 + B\phi'\phi + C\phi + D + E\phi' + F\phi'^2$ so that

$$\begin{aligned}
A &= \frac{(s+1)(2s+1)}{12s\sigma^2},\\
B &= \left(\frac{(s+1)}{2\sigma^2} - \frac{(s+1)(2s+1)}{6s\sigma^2}\right),\\
C &= -\left(\frac{1}{s\sigma^2}\sum y_j(j-\tau_k)\right),\\
D &= \left(\frac{s}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum y_i^2\right),\\
E &= \left(\frac{1}{s\sigma^2}\sum y_j(j-\tau_k) - \frac{1}{\sigma^2}\sum y_i\right),\\
F &= \frac{(s-1)(2s-1)}{12s\sigma^2}.
\end{aligned} \tag{B.5}$$

B.1 Coefficient Updating for OPPL with a Penalty Function which Depends on the Segment Length

For a penalty function which depends on the segment length (discussed in Chapter 5) we need to modify the coefficient updating functions in (B.4) slightly. From 5.6 our recursion for the functions $f^t_{\boldsymbol{\tau}}(\phi)$ is now

$$f^t_{\boldsymbol{\tau}}(\phi) = \min_{\phi'}\left\{f^{\tau_k}_{\tau_1,\ldots,\tau_{k-1}}(\phi') + \mathcal{C}(y_{\tau_k+1:t}, \phi', \phi) + h(t-\tau_k)\right\}.$$

This will make the coefficient updates in (B.4) become

$$a_{\boldsymbol{\tau}}^{t} = A - \frac{B^2}{4\left(a_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + F\right)},$$

$$b_{\boldsymbol{\tau}}^{t} = C - \frac{\left(b_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + E\right)B}{2\left(a_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + F\right)},$$

$$c_{\boldsymbol{\tau}}^{t} = c_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + D - \frac{\left(b_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + E\right)^2}{4\left(a_{(\tau_1,\ldots,\tau_{k-1})}^{\tau_k} + F\right)} + h(t-\tau_k),$$

with the same values A, B, C, D, E and F as defined in (B.5).

Bibliography

Adak, S. (1998). Time-dependent spectral analysis of nonstationary time series. *Journal of the American Statistical Association*, 93(444):1488–1501.

Akaike, H. (1974). A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723.

Arnold, T. B. and Tibshirani, R. J. (2016). Efficient implementations of the generalized LASSO dual path algorithm. *Journal of Computational and Graphical Statistics*, 25(1):1–27.

Aston, J. A. D., Peng, J. Y., and Martin, D. E. K. (2012). Implied distributions in multiple change point problems. *Statistics and Computing*, 22(4):981–993.

Aue, A., Hörmann, S., Horváth, L., and Reimherr, M. (2009). Break detection in the covariance structure of multivariate time series models. *The Annals of Statistics*, 37(6):4046–4087.

Aue, A. and Horvth, L. (2013). Structural breaks in time series. *Journal of Time Series Analysis*, 34(1):1–16.

Auger, I. E. and Lawrence, C. E. (1989). Algorithms for the optimal identification of segment neighborhoods. *Bulletin of Mathematical Biology*, 51:39–54.

Bai, J. (1994). Least squares estimation of a shift in linear processes. *Journal of Time Series Analysis*, 15(5):453–472.

Bardwell, L. and Fearnhead, P. (2016). Bayesian detection of abnormal segments in multiple time series. *Bayesian Analysis*, pages 1–20.

Baron, M. I. (2000). Nonparametric adaptive change point estimation and on line detection. *Sequential Analysis*, 19(1-2):1–23.

Barry, D. and Hartigan, J. A. (1992). Product partition models for change point problems. *The Annals of Statistics*, 20(1):260–279.

Basseville, M. and Benveniste, A. (1983). Sequential segmentation of nonstationary digital signals using spectral analysis. *Information Sciences*, 29:57–73.

Batsidis, A., Horávth, L., Martín, N., Pardo, L., and Zografos, K. (2013). Change-point detection in multinomial data using phi-divergence test statistics. *Journal of Multivariate Analysis*, 118:53–66.

Braun, J. V., Braun, R. K., and Muller, H. G. (2000). Multiple changepoint fitting via quasilikelihood, with application to DNA sequence segmentation. *Biometrika*, 87:301–314.

Braun, J. V. and Müller, H. G. (1998). Statistical methods for DNA sequence segmentation. *Statistical Science*, 13(2):142–162.

Brown, R., Durbin, J., and Evans, J. (1975). Techniques for testing the constancy of regression relationships over time. *Journal of the Royal Statistical Society: Series B (Methodological)*, 37(2):149–192.

Bühlmann, P., Drineas, P., Kane, M., and van der Laan, M. (2016). *Handbook of Big Data*. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. CRC Press.

Cai, J. (1994). A Markov model of switching-regime ARCH. *Journal of Business & Economic Statistics*, 12(3):309–316.

Cappé, O., Moulines, E., and Rydén, T. (2009). Inference in hidden markov models. In *Proceedings of EUSFLAT Conference*, pages 14–16.

Carlin, B. P., Gelfand, A. E., and Smith, A. F. M. (1992). Hierarchical Bayesian analysis of changepoint problems. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):389–405.

Carlstein, E., Müller, H.-G., and Siegmund, D., editors (1994). *Change-Point Problems*. Institute of Mathematical Statistics.

Caron, F., Doucet, A., and Gottardo, R. (2012). On-line changepoint detection and parameter estimation with application to genomic data. *Statistics and Computing*, 22(2):579–595.

Chakar, S. (2015). *Segmentation de Processus avec un Bruit Autorégressif*. PhD thesis, Université Paris Sud-Paris XI.

Chamroukhi, F., Samé, A., Aknin, P., and Govaert, G. (2011). Model-based clustering with Hidden Markov Model regression for time series with regime changes. *Proceedings of the International Joint Conference on Neural Networks*, pages 2814–2821.

Chan, N. H., Yau, C. Y., and Zhang, R.-M. (2013). Group LASSO for structural break time series. *Journal of the American Statistical Association*, 109(506):590–599.

Chen, J. and Gupta, A. K. (2011). *Parametric Statistical Change Point Analysis: With Applications to Genetics, Medicine, and Finance.* Springer Science & Business Media.

Chen, K. M., Cohen, A., and Sackrowitz, H. (2011). Consistent multiple testing for change points. *Journal of Multivariate Analysis*, 102(10):1339–1343.

Chen, X. and Berg, H. C. (2000a). Solvent-isotope and pH effects on flagellar rotation in Escherichia coli. *Biophysical Journal*, 78(5):2280–2284.

Chen, X. and Berg, H. C. (2000b). Torque-speed relationship of the flagellar rotary motor of escherichia coli. *Biophysical Journal*, 78(2):1036–1041.

Chib, S. (1996). Calculating posterior distributions and modal estimates in Markov mixture models. *Journal of Econometrics*, 75(1):79–97.

Chib, S. (1998). Estimation and comparison of multiple change-point models. *Journal of Econometrics*, 86(2):221–241.

Cho, H. and Fryzlewicz, P. (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.

Cleynen, A., Koskas, M., Lebarbier, E., Rigaill, G., and Robin, S. (2014). Segmentor3IsBack: An R package for the fast and exact segmentation of seq-data. *Algorithms for Molecular Biology*, 9(6):1–11.

Dalalyan, A., Hebiri, M., and Lederer, J. (2014). On the prediction performance of the LASSO. Technical report, Centre de Recherche en Economie et Statistique.

Davis, R. A., Lee, T. C. M., and Rodriguez-Yam, G. A. (2006). Structural break estimation for nonstationary time series models. *Journal of the American Statistical Association*, 101(473):223–239.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids.* Cambridge University Press.

Eckley, I. A., Fearnhead, P., and Killick, R. (2011). Analysis of changepoint models. In Barber, D., Cemgil, A. T., and Chiappa, S., editors, *Bayesian Time Series Models*, pages 1120–1126. Cambridge University Press.

Fearnhead, P. (2005). Exact Bayesian curve fitting and signal segmentation. *IEEE Transactions on Signal Processing*, 53(6):2160–2166.

Fearnhead, P. (2006). Exact and efficient Bayesian inference for multiple changepoint problems. *Statistics and Computing*, 16(2):203–213.

Fearnhead, P. and Clifford, P. (2003). On-line inference for hidden Markov models via particle filters. *Journal of the Royal Statistical Society, Series B*, 65(4):887–899.

Fearnhead, P. and Liu, Z. (2007). On-line inference for multiple changepoint problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):589–605.

Frick, K., Munk, A., and Sieling, H. (2014). Multiscale change point inference. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(3):495–580.

Fryzlewicz, P. (2014). Wild binary segmentation for multiple change-point detection. *The Annals of Statistics*, 42(6):2243–2281.

Fryzlewicz, P. and Subba Rao, S. (2014). Multiple change-point detection for auto-regressive conditional heteroscedastic processes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(5):903–924.

Futschik, A., Hotz, T., Munk, A., and Sieling, H. (2014). Multiscale DNA partitioning: statistical evidence for segments. *Bioinformatics*, 30(16):2255–2262.

Goldberg, N., Kim, Y., Leyffer, S., and Veselka, T. D. (2014). Adaptively refined dynamic program for linear spline regression. *Computational Optimization and Applications*, 58(3):523–541.

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732.

Haas, M., Mittnik, S., and Paolella, M. S. (2004). A new approach to Markov-switching GARCH models. *Journal of Financial Econometrics*, 2(4):493–530.

Haccou, P., Meelis, E., and Geer, S. V. D. (1987). The likelihood ratio test for the change point problem for exponentially distributed random variables. *Stochastic Processes and their Applications*, 27:121–139.

Hamilton, J. D. (1990). Analysis of time series subject to changes in regime. *Journal of Econometrics*, 45(1):39–70.

Hampel, F. R. (1974). The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393.

Harchaoui, Z. and Lévy-Leduc, C. (2010). Multiple change-point estimation with a total variation penalty. *Journal of the American Statistical Association*, 105(492):1480–1493.

Hawkins, D. and Deng, Q. (2010). A nonparametric change-point control chart. *Journal of Quality Technology*, 42(2):165–173.

Haynes, K., Eckley, I. A., and Fearnhead, P. (2015). Computationally efficient changepoint detection for a range of penalties. *Journal of Computational and Graphical Statistics*, (to appear):1–28.

Haynes, K., Fearnhead, P., and Eckley, I. A. (2016). A computationally efficient nonparametric approach for changepoint detection. *arXiv preprint arXiv:1602.01254*.

Hinkley, D. V. (1970). Inference about the change-point in a sequence of random variables. *Biometrika*, 57(1):1–17.

Hocking, T. D., Boeva, V., Rigaill, G., Schleiermacher, G., Janoueix-Lerosey, I., Delattre, O., Richer, W., Bourdeaut, F., Suguro, M., Seto, M., Bach, F., and Vert, J.-P. (2014). SegAnnDB: Interactive web-based genomic segmentation. *Bioinformatics*, 30(11):1539–1546.

Hocking, T. D., Schleiermacher, G., Janoueix-lerosey, I., Boeva, V., Cappo, J., Delattre, O., Bach, F., and Vert, J.-P. (2013). Learning smoothing models of copy number profiles using breakpoint annotations. *BNC Bioinformatics*, 14(164):1–14.

Hoefling, H. (2009). A path algorithm for the fused LASSO signal approximator. *Journal of Computational and Graphical Statistics*, 19(4):38.

Horner, A. and Beauchamp, J. (1996). Piecewise-linear approximation of additive synthesis envelopes : a comparison of various methods. *Computer Music Journal*, 20(2):72–95.

Horváth, L. and Hušková, M. (2012). Change-point detection in panel data. *Journal of Time Series Analysis*, 33(4):631–648.

Hyun, S., G'Sell, M., and Tibshirani, R. J. (2016). Exact post-selection inference for changepoint detection and other generalized LASSO problems. *arXiv preprint arXiv:1606.03552*, pages 1–39.

Inclan, C. and Tiao, G. C. (1994). Use of cumulative sums of squares for retrospective detection of changes of variance. *Journal of the American Statistical Association*, 89(427):913–923.

Jackson, B., Scargle, J. D., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., Sangtrakulcharoen, P., Tan, L., and Tsai, T. T. (2005). An algorithm for optimal partitioning of data on an interval. *IEE Signal Processing Letters*, 12(2):105–108.

James, N. A. and Matteson, D. S. (2015). Change points via probabilistically pruned objectives. *arXiv preprint arXiv:1505.04302*.

Jeffreys, H. (1961). *The Theory of Probability.* Oxford University Press, third edition.

Jeng, X. J., Cai, T. T., and Li, H. (2013). Simultaneous discovery of rare and common segment variants. *Biometrika*, 100(1):157–172.

Julious, S. A. (2001). Inference and estimation in a changepoint regression problem. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 50(1):51–61.

Killick, R., Eckley, I. A., Ewans, K., and Jonathan, P. (2010). Detection of changes in variance of oceanographic time-series using changepoint analysis. *Ocean Engineering*, 37(13):1120–1126.

Killick, R., Fearnhead, P., and Eckley, I. A. (2012). Optimal detection of changepoints with a linear computational cost. *Journal of the American Statistical Association*, 107(500):1590–1598.

Kim, H.-J., Fay, M. P., Feuer, E. J., and Midthune, D. N. (2000). Permutation tests for joinpoint regression with applications to cancer rates. *Statistics in Medicine*, 19(4):335–351.

Kim, S. J., Koh, K., Boyd, S., and Gorinevsky, D. (2009). l_1 Trend Filtering. *SIAM Review*, 51(2):339–360.

Kirch, C. and Muhsal, B. (2014). A MOSUM procedure for the estimation of multiple random change points. *Preprint*.

Korkas, K. K. and Fryzlewicz, P. (2015). Multiple change-point detection for non-stationary time series using Wild Binary Segmentation. *Preprint*, pages 1–37.

Lai, W. R., Johnson, M. D., Kucherlapati, R., and Park, P. J. (2005). Comparative analysis of algorithms for identifying amplifications and deletions in array CGH data. *Bioinformatics*, 21(19):3763–3770.

Lavielle, M. (2005). Using penalized contrasts for the change-point problem. *Signal Processing*, 85(8):1501–1510.

Lavielle, M. and Lebarbier, E. (2001). An application of MCMC methods for the multiple change-points problem. *Signal Processing*, 81(1):39–53.

Lavielle, M. and Teyssière, G. (2006). Detection of multiple change-points in multivariate time series. *Lithuanian Mathematical Journal*, 46(3):287–306.

Lee, C.-B. (1995). Estimating the number of change points in a sequence of independent normal random variables. *Statistics & Probability Letters*, 25(3):241–248.

Lee, C.-B. (1996). Nonparametric multiple change-point estimators. *Statistics & Probability Letters*, 27(4):295–304.

Lee, S., Ha, J., Na, O., and Na, S. (2003). The CUSUM test for parameter change in time series models. *Scandinavian Journal of Statistics*, 30(4):781–796.

Levin, B. and Kline, J. (1985). The CUSUM test of homogeneity with an application in spontaneous abortion epidemiology. *Statistics in Medicine*, 4(4):469–488.

Liu, J. S. and Lawrence, C. E. (1999). Bayesian inference on biopolymer models. *Bioinformatics*, 15(1):38–52.

Lung-Yut-Fong, A., Lévy-Leduc, C., and Cappé, O. (2012). Distributed detection/localization of change-points in high-dimensional network traffic data. *Statistics and Computing*, 22(2):485–496.

Luong, T. M., Perduca, V., and Nuel, G. (2004). Hidden markov model applications in change-point analysis. *arXiv preprint arXiv:1212.1778*.

Maboudou, E. M. and Hawkins, D. M. (2009). Fitting multiple change-point models to a multivariate Gaussian model. *Proceedings of the Second International Workshop in Sequential Methodologies*, pages 2–6.

Maboudou-Tchao, E. M. and Hawkins, D. M. (2013). Detection of multiple change-points in multivariate time series. *Journal of Applied Statistics*, 40(9):1979–1995.

Maidstone, R., Hocking, T., Rigaill, G., and Fearnhead, P. (2016). On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, pages 1–15.

Matteson, D. S. and James, N. A. (2014). A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345.

Nam, C. F. H., Aston, J. A. D., and Johansen, A. M. (2012). Quantifying the uncertainty in change points. *Journal of Time Series Analysis*, 33(5):807–823.

Ninomiya, Y. (2005). Information criterion for Gaussian change-point model. *Statistics & Probability Letters*, 72(3):237–247.

Olshen, A. B., Venkatraman, E. S., Lucito, R., and Wigler, M. (2004). Circular Binary Segmentation for the analysis of array-based DNA copy number data. *Biostatistics*, 5(4):557–572.

Ombao, H., von Sachs, R., and Guo, W. (2005). SLEX analysis of multivariate nonstationary time series. *Journal of the American Statistical Association*, 100(470):519–531.

Page, E. (1954). Continuous inspection schemes. *Biometrika*, 41(1–2):100–115.

Pein, F., Sieling, H., and Munk, A. (2015). Heterogeneuous change point inference. *arXiv preprint arXiv:1505.04898*.

Pelletier, D. (2006). Regime switching for dynamic correlations. *Journal of Econometrics*, 131(1):445–473.

Pettitt, A. N. (1979). A non-parametric approach to the change-point problem. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 28(2):126–135.

Picard, F., Lebarbier, E., Hoebeke, M., Rigaill, G., Thiam, B., and Robin, S. (2011). Joint segmentation, calling, and normalization of multiple CGH profiles. *Biostatistics*, 12(3):413–428.

Pickering, B. J. (2015). *Changepoint Detection for Acoustic Sensing Signals*. PhD thesis, STOR-i CDT, Lancaster University.

Preuss, P., Puchstein, R., and Dette, H. (2015). Detection of multiple structural breaks in multivariate time series. *Journal of the American Statistical Association*, 110(510):654–668.

Rabiner, L. R. (1989). Tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Reeves, J., Chen, J., Wang, X. L., Lund, R., and Lu, Q. Q. (2007). A review and comparison of changepoint detection techniques for climate data. *Journal of Applied Meteorology and Climatology*, 46(6):900–915.

Rigaill, G. (2015). A pruned dynamic programming algorithm to recover the best segmentations with 1 to $K_{\max}$ change-points. *Journal de la Société Française de Statistique*, 156(4):180–205.

Rigaill, G., Hocking, T., Bach, F., and Vert, J.-P. (2013). Learning sparse penalties for change-point detection using max margin interval regression. *Proceedings of The 30th International Conference on Machine Learning*, 28(3):172–180.

Rigaill, G., Lebarbier, E., and Robin, S. (2012). Exact posterior distributions and model selection criteria for multiple change-point detection problems. *Statistics and Computing*, 22(4):917–929.

Rissanen, J. (1989). *Stochastic Complexity in Statistical Inquiry Theory*. World Scientific Publishing Co., Inc., River Edge, NJ, USA.

Rojas, C. R. and Wahlberg, B. (2014). On change point detection using the fused LASSO method. *arXiv preprint arXiv:1401.5408*, pages 1–52.

Ross, G. and Adams, N. M. (2012). Two nonparametric control charts for detecting arbitrary distribution changes. *Journal of Quality Technology*, 44(2):102–116.

Rudin, L. I., Osher, S., and Fatemi, E. (1992). Nonlinear total variation noise removal algorithm. *Physica D*, 60(1):259–268.

Rueda, O. M. and Díaz-Uriarte, R. (2007). Flexible and accurate detection of genomic copy-number changes from aCGH. *PLoS Computational Biology*, 3(6):1115–1122.

Ryu, W. S., Berry, R. M., and Berg, H. C. (2000). Torque-generating units of the flagellar motor of Escherichia coli have a high duty ratio. *Nature*, 403(6768):444–447.

Schwaller, L. and Robin, S. (2016). Exact Bayesian inference for off-line change-point detection in tree-structured graphical models. *arXiv preprint arXiv:1603.07871*, pages 1–16.

Schwaller, L., Robin, S., and Stumpf, M. (2015). Bayesian Inference of Graphical Model Structures Using Trees. *arXiv preprint arXiv:1504.02723*, pages 1–16.

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464.

Scott, A. J. and Knott, M. (1974). A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30(3):507–512.

Shao, X. and Zhang, X. (2010). Testing for change points in time series. *Journal of the American Statistical Association*, 105(491):1228–1240.

Shen, J., Gallagher, C. M., and Lu, Q. (2014). Detection of multiple undocumented change-points using adaptive LASSO. *Journal of Applied Statistics*, 41(6):1161–1173.

Siegmund, D., Yakir, B., and Zhang, N. R. (2011). Detecting simultaneous variant intervals in aligned sequences. *The Annal of Applied Statistics*, 5(2):645–668.

Sowa, Y. and Berry, R. M. (2008). Bacterial flagellar motor. *Quarterly Reviews of Biophysics*, 41(2):103–132.

Sowa, Y., Hotta, H., Homma, M., and Ishijima, A. (2003). Torque-speed relationship of the na+-driven flagellar motor of vibrio alginolyticus. *Journal of Molecular Biology*, 327(5):1043–1051.

Sowa, Y., Rowe, A., Leake, M., Yakushi, T., Homma, M., Ishijima, A., and Berry, R. (2005). Direct observation of steps in rotation of the bacterial flagellar motor. *Nature*, 437(7060):916–919.

Srivastava, M. S. and Worsley, K. J. (1986). Likelihood ratio tests for a change in the multivariate normal mean. *Journal of the American Statistical Association*, 81(393):199–204.

Stephens, D. A. (1994). Bayesian retrospective multiple-changepoint identification. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 43(1):159–178.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused LASSO. *Journal of the Royal Statistical Society Series B*, 67(1):91–108.

Tibshirani, R. J. (2014). Adaptive piecewise polynomial estimation via trend filtering. *The Annals of Statistics*, 42(1):285–323.

Tomé, A. R. and Miranda, P. M. A. (2004). Piecewise linear fitting and trend changing points of climate parameters. *Geophysical Research Letters*, 31(2).

Venkatraman, E. S. (1992). *Consistency Results in Multiple Change-Point Problems.* PhD thesis, Department of Statistics, Stanford University.

Venkatraman, E. S. and Olshen, A. B. (2007). A faster circular binary segmentation algorithm for the analysis of array CGH data. *Bioinformatics*, 23(6):657–663.

Vert, J.-P. and Bleakley, K. (2010). Fast detection of multiple change-points shared by many signals using group LARS. In Lafferty, J. D., Williams, C. K. I., Shawe-Taylor, J., Zemel, R. S., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2343–2351. Curran Associates, Inc.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Vostrikova, L. (1981). Detecting "disorder" in multidimensional random processes. *Doklady Mathematics*, 24:55–59.

Weinmann, A. and Storath, M. (2015). Iterative Potts and Blake–Zisserman minimization for the recovery of functions with discontinuities from indirect measurements. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 471(2176):1–25.

Wilks, S. S. (1938). The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The Annals of Mathematical Statistics*, 9(1):60–62.

Willenbrock, H. and Fridlyand, J. (2005). A comparison study: Applying segmentation to array CGH data for downstream analyses. *Bioinformatics*, 21(22):4084–4091.

Xie, Y. and Siegmund, D. (2013). Sequential multi-sensor change-point detection. *Annals of Statistics*, 41(2):670–692.

Yao, Y. C. (1988). Estimating the number of change-points via Schwarz' criterion. *Statistics & Probability Letters*, 6(2):181–189.

Yao, Y.-C. and Au, S. T. (1989). Least-squares estimation of a step function. *The Indian Journal of Statistics*, 51(3):370–381.

Yao, Y.-C. and Davis, R. A. (1986). The asymptotic behavior of the likelihood ratio statistic for testing a shift in mean in a sequence of independent normal variates. *The Indian Journal of Statistics, Series A*, 48(3):339–353.

Zhang, N. R. and Siegmund, D. O. (2007). A modified Bayes information criterion with applications to the analysis of comparative genomic hybridization data. *Biometrics*, 63(1):22–32.

Zhang, N. R., Siegmund, D. O., Ji, H., and Li, J. Z. (2010). Detecting simultaneous changepoints in multiple sequences. *Biometrika*, 97(3):631–645.

Zhou, J. and Liu, S. Y. (2009). Inference for mean change-point in infinite variance AR(p) process. *Statistics & Probability Letters*, 79(1):6–15.

Zhou, J., Lloyd, S. A., and Blair, D. F. (1998). Electrostatic interactions between rotor and stator in the bacterial flagellar motor. *Proceedings of the National Academy of Sciences of the United States of America*, 95(11):6436–6441.

Zou, C., Yin, G., Feng, L., and Wang, Z. (2014). Nonparametric maximum likelihood approach to multiple change-point problems. *Annals of Statistics*, 42(3):970–1002.