Binary segmentation

Toby Dylan Hocking

Introduction

Finite Sample Complexity Analysis of Binary Segmentation (arXiv:2410.08654)

Comparing binsegRcpp with other implementations of binary segmentation (under review at Journal of Statistical Software)

Motivation for change-point detection in time series data

- Detecting changes/abnormalities important in medicine.

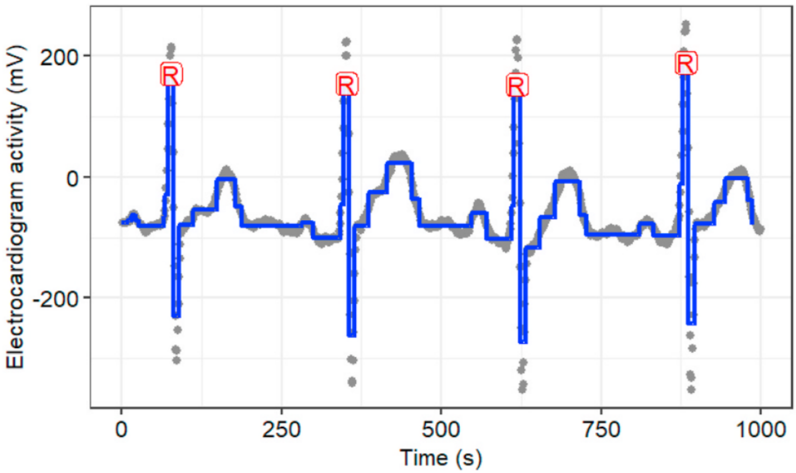


Figure 1: Electrocardiograms (heart monitoring), Fotoohinasab et al, Asilomar conference 2020.

Motivation for change-point detection in time series data

- Detecting the time when a spike occurs is important in neuroscience.

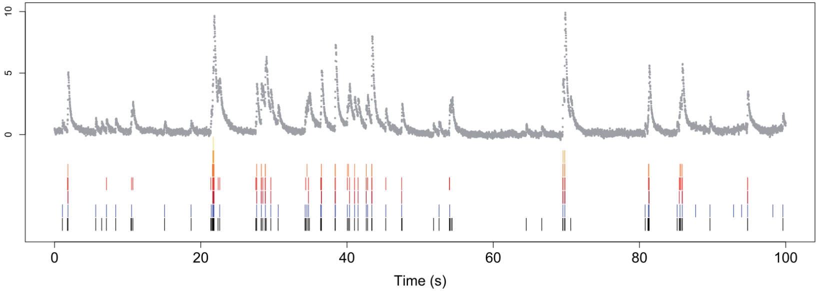


Figure 2: Neural spikes in calcium imaging data, Jewell et al, Biostatistics 2019.

Motivation for change-point detection in genomic data sequences

- Detecting breakpoints is important in diagnosis of some types of cancer, such as neuroblastoma.

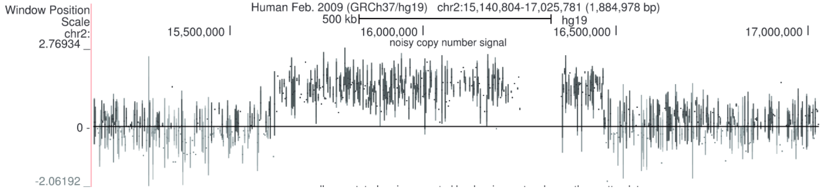


Figure 3: DNA copy number data, breakpoints associated with aggressive cancer, Hocking et al, Bioinformatics 2014.

Motivation for change-point detection in genomic data sequences

- Detecting peaks (up/down changes) in genomic data is important in order to understand which genes are active or inactive.

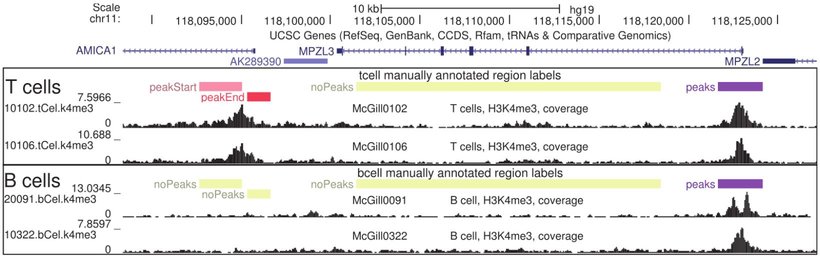


Figure 4: ChIP-seq data for characterizing active regions in the human genome, Hocking et al, Bioinformatics 2017.

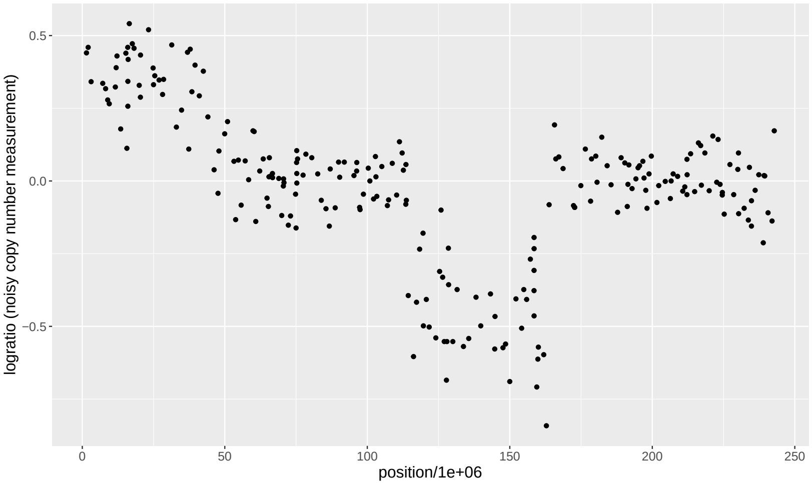
Segmentation / change-point detection framework

- Let $x_1, \dots, x_n \in \mathbb{R}$ be a data sequence over space or time (logratio column in DNA copy number data below).
- Where are the abrupt changes in the data sequence?

```
##        profile.id chromosome  position    logratio
##            <fctr>     <fctr>     <int>       <num>
##   1:            4          2   1472476  0.44042072
##   2:            4          2   2063049  0.45943162
##   3:            4          2   3098882  0.34141652
##   4:            4          2   7177474  0.33571191
##   5:            4          2   8179390  0.31730407
##  ---
## 230:            4          2 239227603  0.01863417
## 231:            4          2 239471307  0.01720929
## 232:            4          2 240618997 -0.10935876
## 233:            4          2 242024751 -0.13764780
## 234:            4          2 242801018  0.17248752
```

Segmentation / change-point data visualization

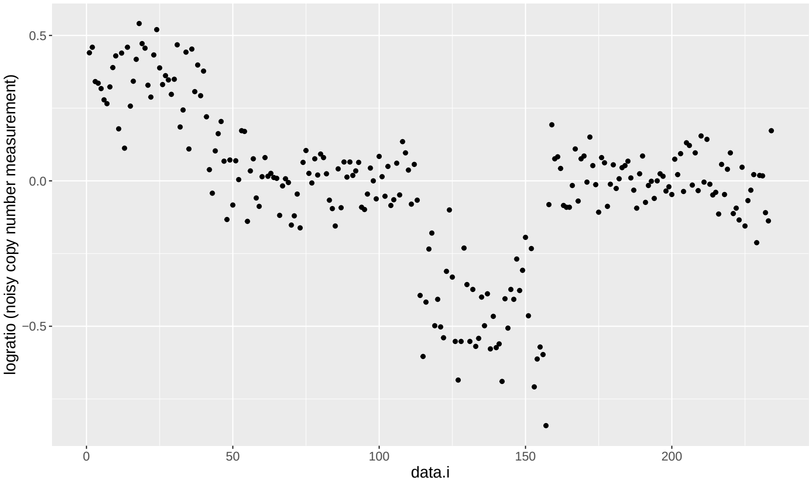
- Let $x_1, \ldots, x_n \in \mathbb{R}$ be a data sequence over space or time.
- Where are the abrupt changes in the data sequence?



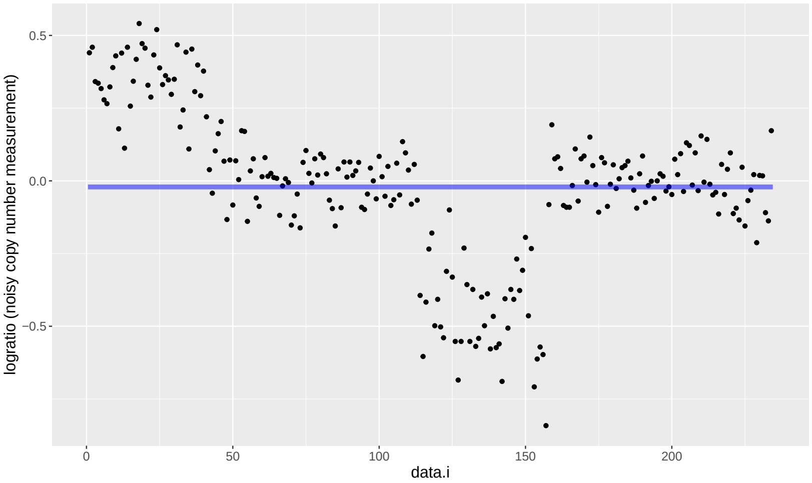
Assume normal distribution with change in mean, constant variance

- There are a certain number of clusters/segments $K \in \{1, \ldots, n\}$.
- Each segment $k \in \{1, \ldots, K\}$ has its own mean parameter $\mu_k \in \mathbb{R}$.
- There is some constant variance parameter $\sigma^2 > 0$ which is common to all segments.
- For each data point i on segment $k \in \{1, \ldots, K\}$ we have $x_i \sim N(\mu_k, \sigma^2)$ – normal distribution.
- This normal distribution assumption means that we want to find segments/change-points with mean m that minimize the square loss, $(x - m)^2$.
- Other distributional assumptions / loss functions are possible.

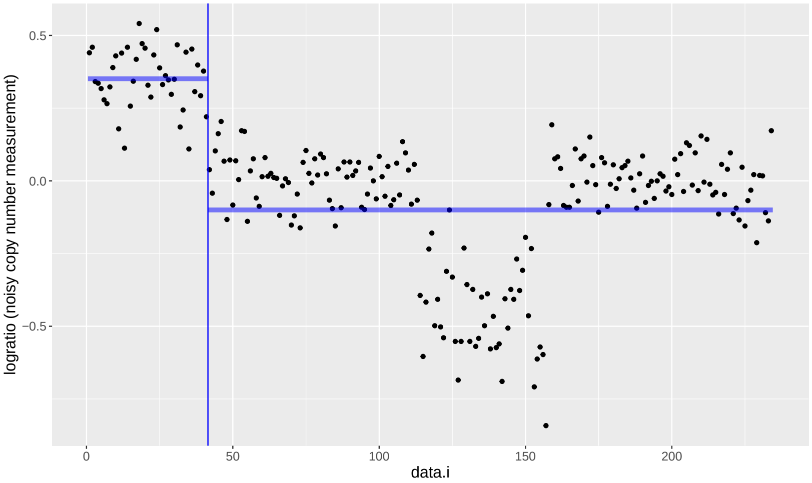
Visualize data sequence



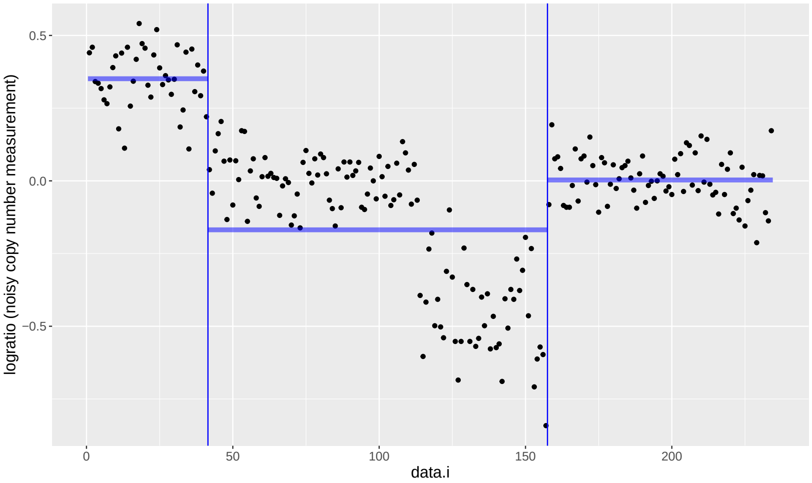
Simplest model, 1 segment, 0 change-points



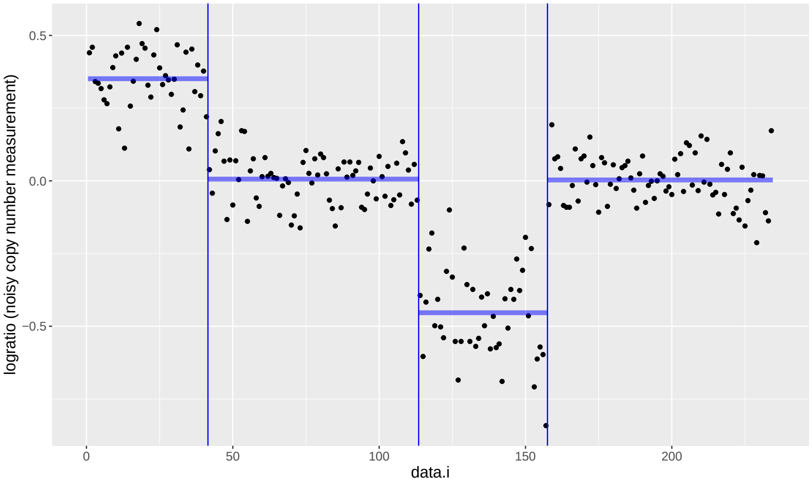
Find best single change-point (two segments)



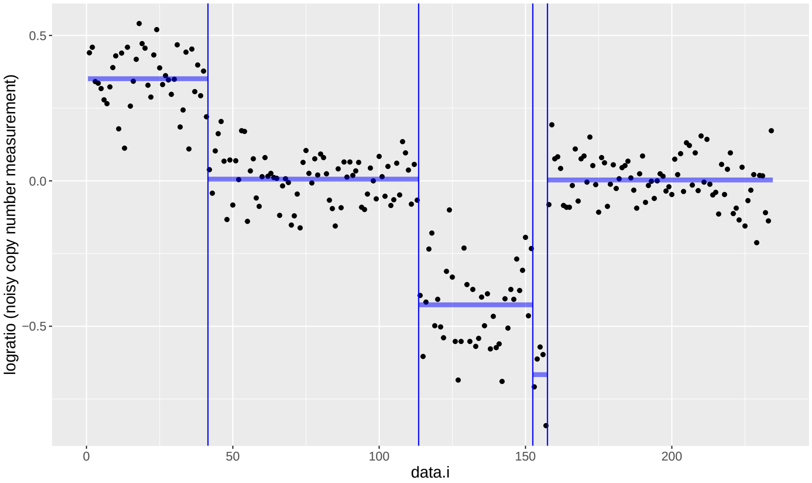
Find next best change-point (given first change)



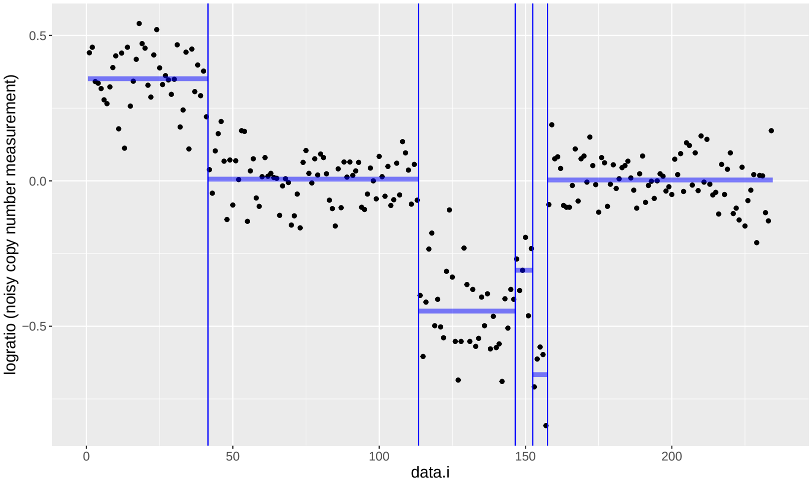
Find four segments



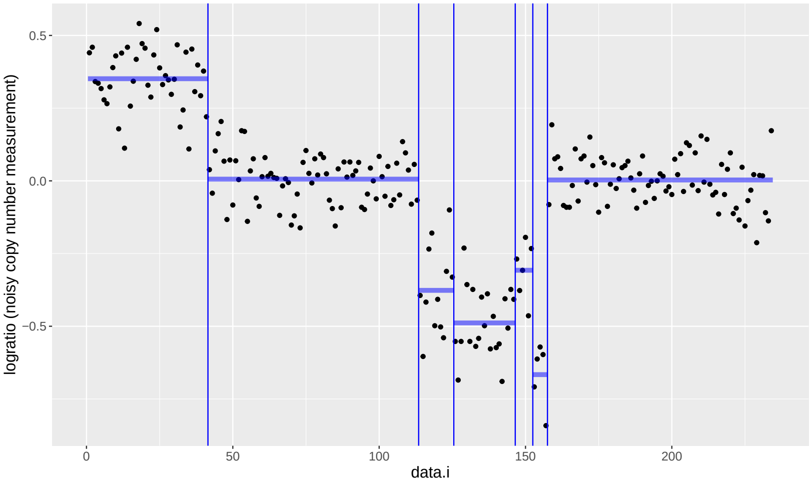
Find five segments



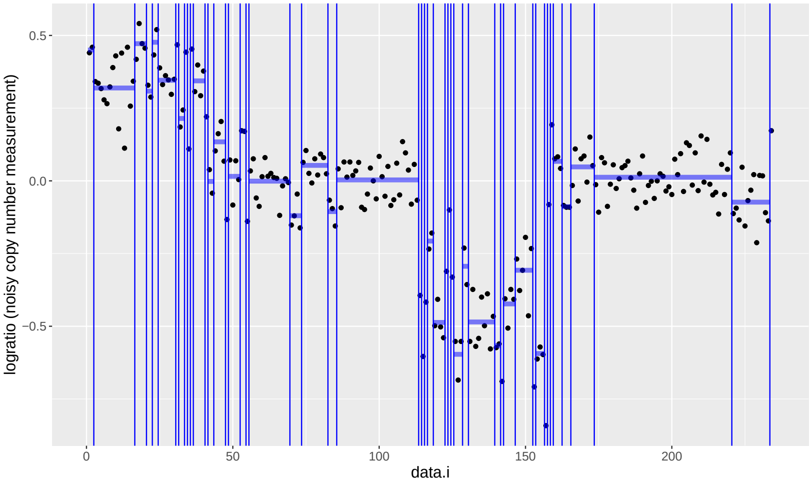
Find six segments



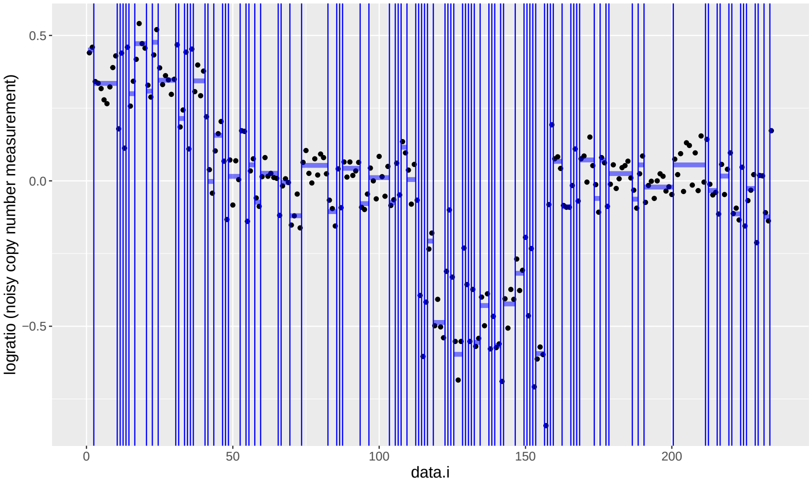
Find seven segments



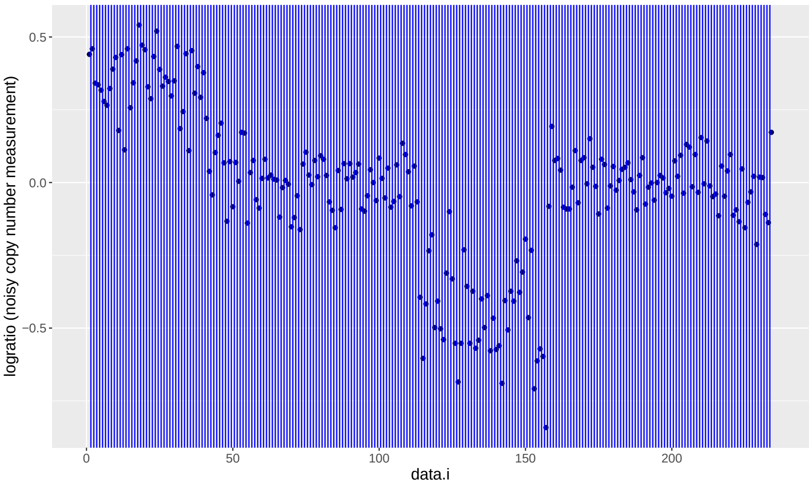
Find 50 segments



Find 100 segments



Largest model: 234 segments (changes everywhere)

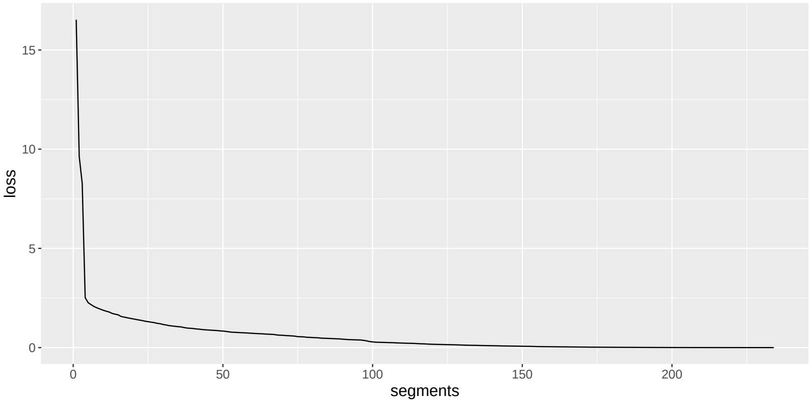


Why binary segmentation?

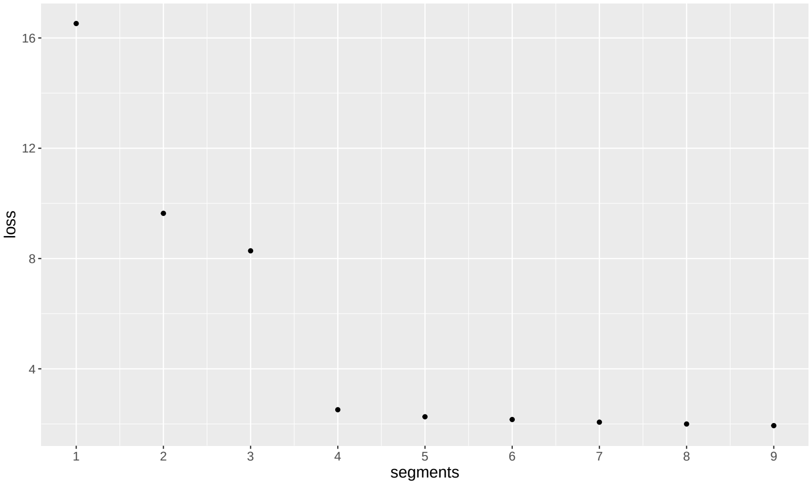
- Dynamic programming computes one optimal model for a penalty λ in $O(n \log n)$ time in sequence of n data (see pruning algorithms: PELT—Killick 2012, FPOP—Maidstone 2017, DUST—Truong and Runge 2024).
- Binary segmentation (Scott and Knott 1974) is a simple baseline algorithm.
- In theory, binary segmentation is extremely fast: heuristic sequence of models with $1, \dots, K$ segments computed in $O(n \log K)$ time (typical/best case) or $O(nK)$ time (worst case).
- Complete sequence of n models for n data in only $O(n \log n)$ time!
- Reviewers of my papers about dynamic programming want to see comparisons with binary segmentation: is there any difference in speed/accuracy?
- If we run existing code for binary segmentation: does it achieve the expected time complexity? Why or why not?

Error/loss function visualization

- Let $m^{(k)} \in \mathbb{R}^n$ be the mean vector with k segments.
- Error for k segments is defined as sum of squared difference between data x and mean $m^{(k)}$ vectors, $E_k = \sum_{i=1}^{n} (x_i - m_i^{(k)})^2$
- As in previous clustering models, kink in the error curve can be used as a simple model selection criterion.



Error/loss function zoom



Learning algorithm

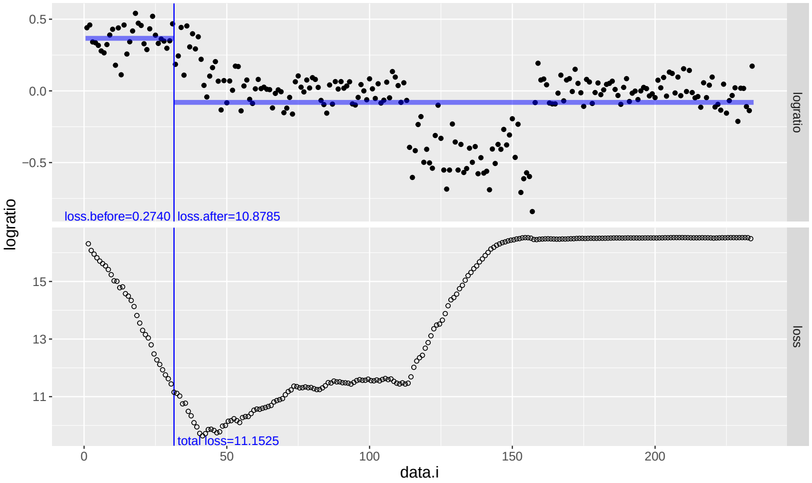
- Start with one segment, then repeat:
- Compute loss of each possible split.
- Choose split which results in largest loss decrease.
- If $s = \sum_{i=1}^n x_i$ is the sum over n data points, then the mean is s/n and the square loss (from 1 to n) is

$$L_{1,n} = \sum_{i=1}^n (x_i - s/n)^2 = \sum_{i=1}^n [x_i^2] - 2(s/n)s + n(s/n)^2$$

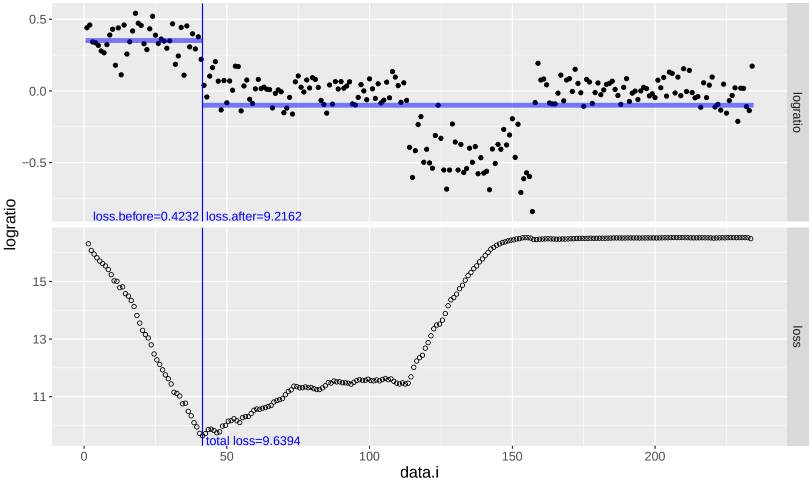
- Given cumulative sums, and a split point t, we can compute square loss from 1 to t, $L_{1,t}$, and from $t+1$ to n, $L_{t+1,n}$, in constant $O(1)$ time.
- We can minimize over all change-points t in linear $O(n)$ time,

$$\min_{t \in \{1,\dots,n-1\}} L_{1,t} + L_{t+1,n}$$

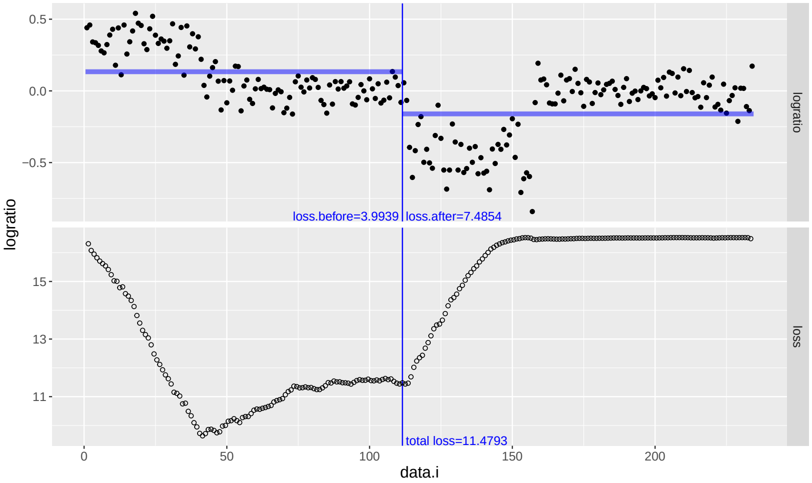
First step of binary segmentation



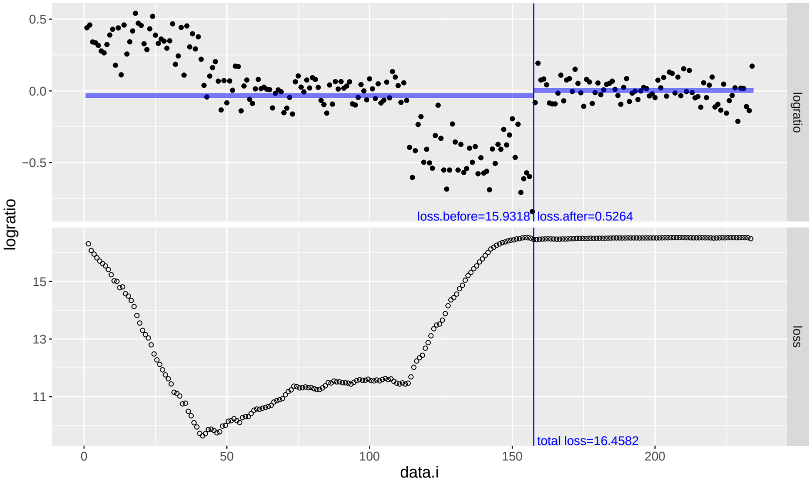
First step of binary segmentation



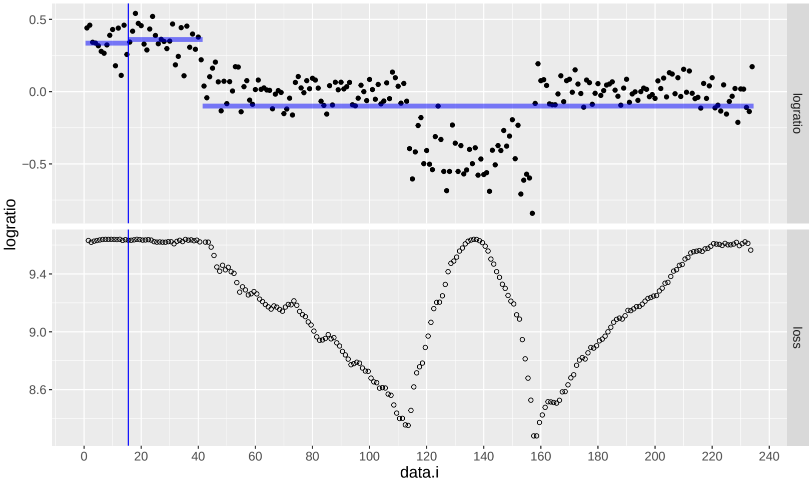
First step of binary segmentation



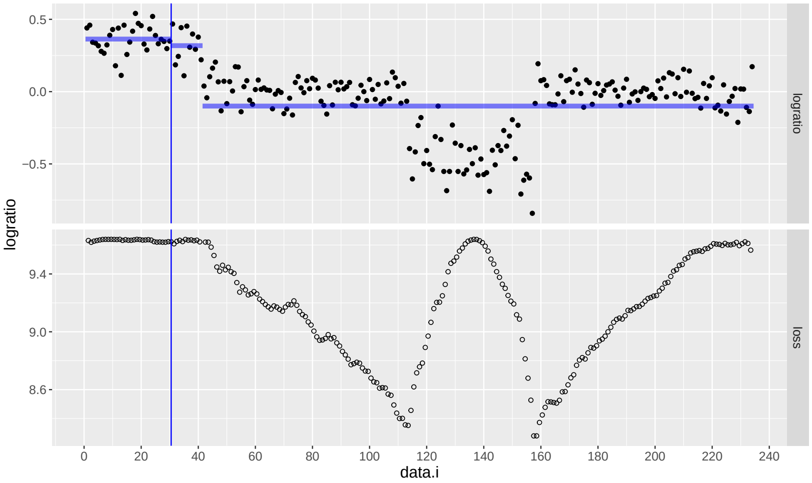
First step of binary segmentation



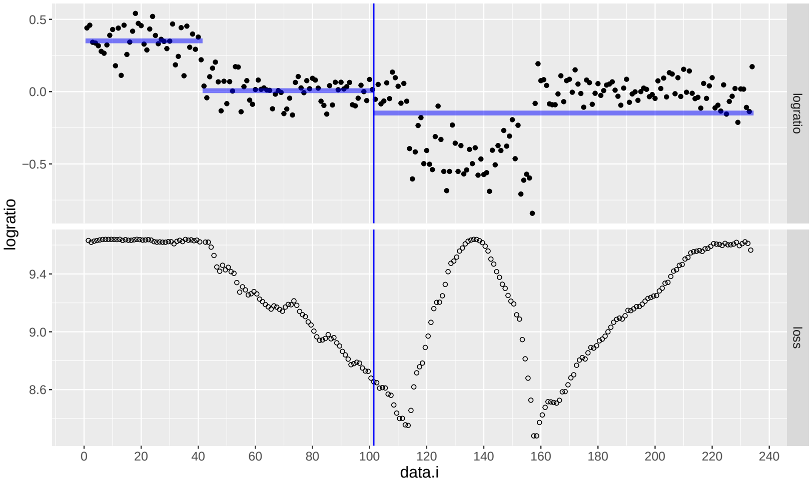
Second step of binary segmentation



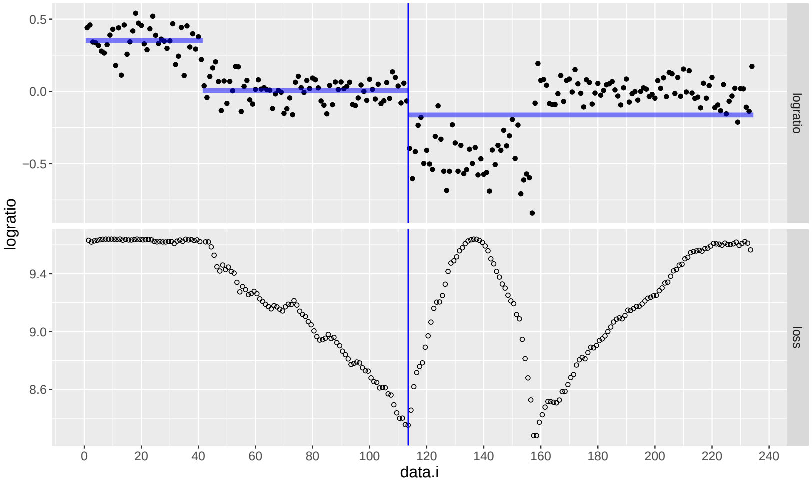
Second step of binary segmentation



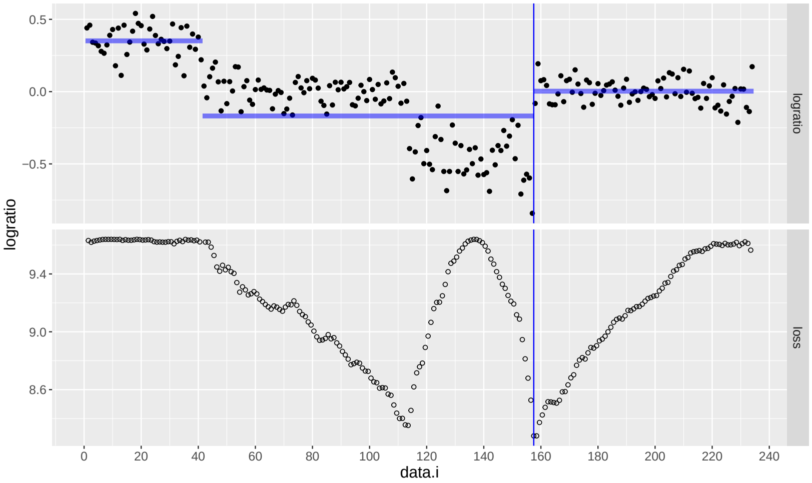
Second step of binary segmentation



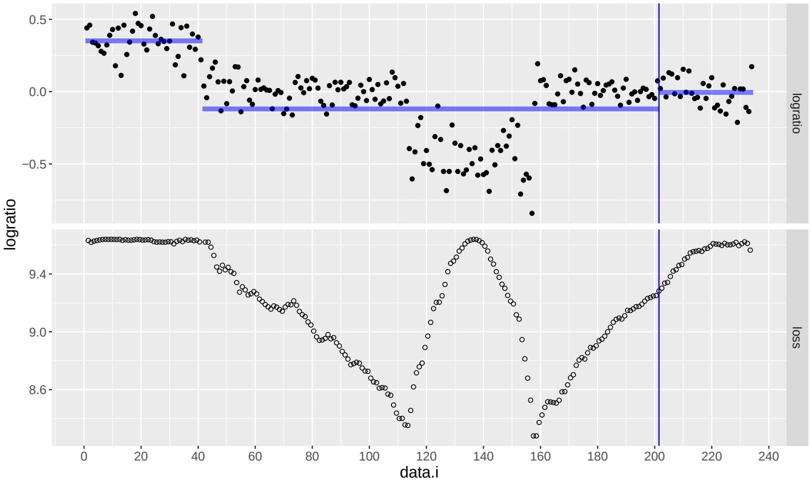
Second step of binary segmentation



Second step of binary segmentation

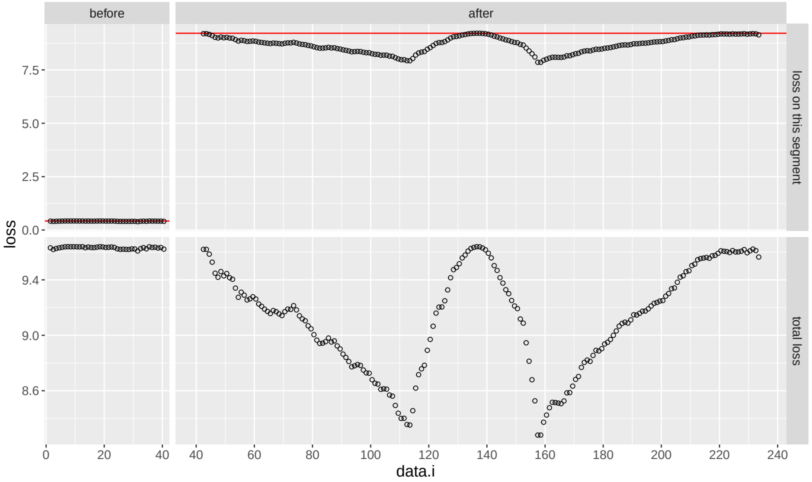


Second step of binary segmentation



Efficient loss computation

- Minimization can be performed by choosing the split with loss (black point) which maximizes the decrease in loss with respect to previous model (red, with no split).



Learning algorithm, implementation details

First compute the vectors of cumulative sums of data and squares, $y_1 = z_1 = 0, y_t = \sum_{i=1}^{t-1} x_i, z_t = \sum_{i=1}^{t-1} x_i^2$, for all $t \in \{2, \dots, n+1\}$.

Assume there is some set $\mathcal{S}$ of segments that could be split, each $(j, e) \in \mathcal{S}$ is a segment start j and end e (both in $1, \dots, n$).

Then the next segment to split (j, e), and best split point t, are defined by the best loss decrease,

$$\max_{(j,e)\in\mathcal{S}} \underbrace{L_{j,e}}_{\text{loss before split}} - \min_{t\in\{j,\dots,e-1\}} \underbrace{L_{j,t} + L_{t+1,e}}_{\text{loss after split}}$$

Use the cumsum trick to compute loss in constant time,

$$\sum_{i=j}^{t} x_i = y_{t+1} - y_j.$$

$$L_{j,t} = z_{t+1} - z_j - (y_{t+1} - y_j)^2/(t - j + 1).$$

Learning algorithm, recursion/pseudo-code

Notation. Let $D_{j,e}(t) = L_{j,t} + L_{t+1,e} - L_{j,e}$ be the loss difference after splitting segment (j, e) at t, and let $f(j, e) = \min, \arg\min_{t \in \{j,\dots,e-1\}} D_{j,e}(t)$ be the best loss difference/split on segment (j, e).

Initialization. Let $\mathcal{L}_1 = L_{1,n}$ be the loss with one segment, and let $\mathcal{S}_1 = \{(1, n, f(1, n))\}$ be the initial segment to split.

Recursion. For all $k \in \{2, \dots, K\}$: (max segments $K \leq n$)

- $j_k^*, e_k^*, d_k^*, t_k^* = \arg\min_{(j,e,d,t) \in \mathcal{S}_{k-1}} d$ (best segment to split)
- $\mathcal{L}_k = \mathcal{L}_{k-1} + d_k^*$, (loss)
- $\mathcal{N}_k = \{(j_k^*, t_k^*), (t_k^* + 1, e_k^*)\}$ (new segments)
- $\mathcal{V}_k = \{(j, e, f(j, e)) \mid (j, e) \in \mathcal{N}_k, j < e\}$ (splittable segments)
- $\mathcal{S}_k = [\mathcal{S}_{k-1} \setminus (j_k^*, e_k^*, d_k^*, t_k^*)] \cup \mathcal{V}_k$ (segments to search)

Modification for min segment length

Sometimes there is prior knowledge that there should be no segments with fewer data points than $\ell \in \{1, 2, \dots\}$, and in that case there is a simple modification:

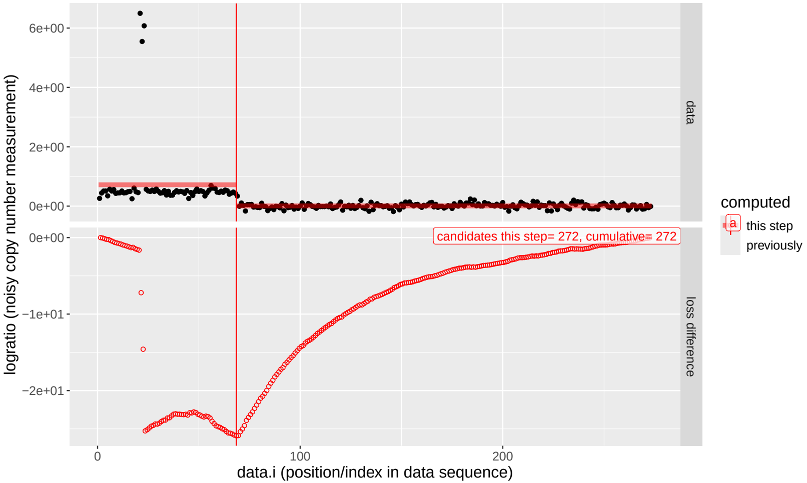
- $f(j, e) = \min, \arg\min_{t \in \{j+\ell-1, \dots, e-\ell\}} D_{j,e}(t)$ (best split)
- $\mathcal{V}_k = \{(j, e, f(j, e)) \mid (j, e) \in \mathcal{N}_k,\ e - j + 1 \geq 2\ell\}\}$ (splittable segments)

Introduction

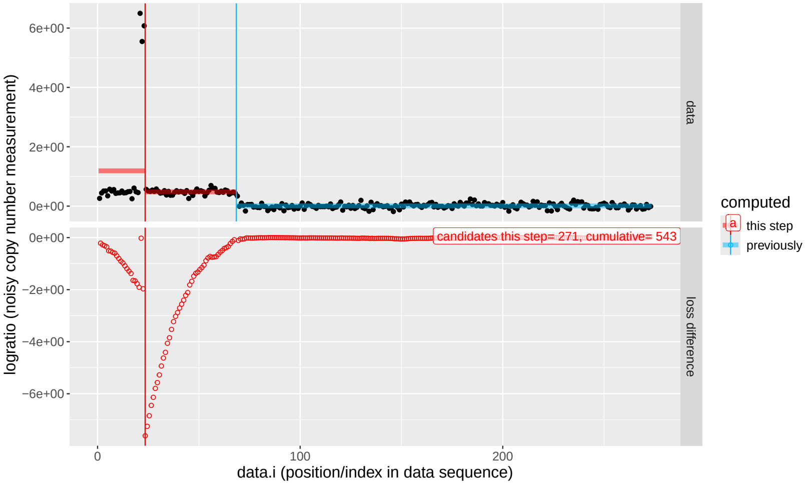
Finite Sample Complexity Analysis of Binary Segmentation (arXiv:2410.08654)

Comparing binsegRcpp with other implementations of binary segmentation (under review at Journal of Statistical Software)

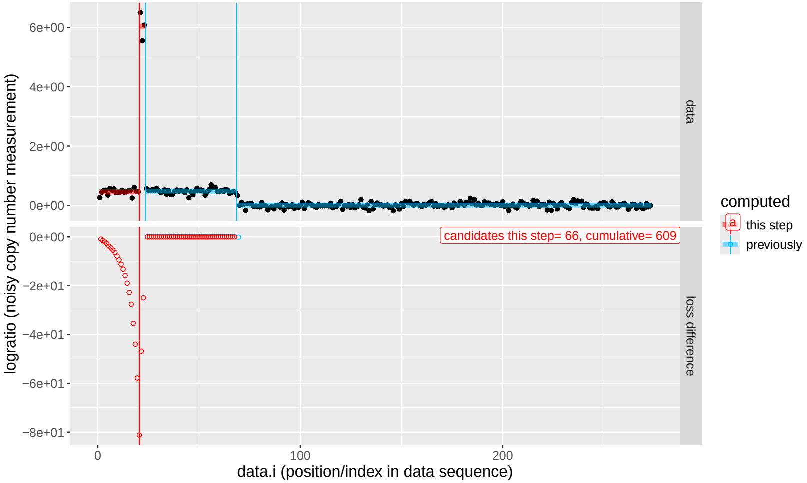
Visualization of computations at each iteration



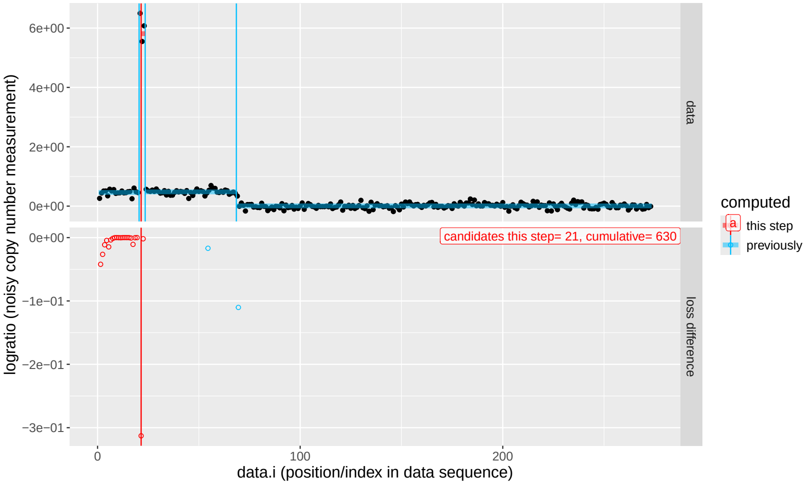
Visualization of computations at each iteration



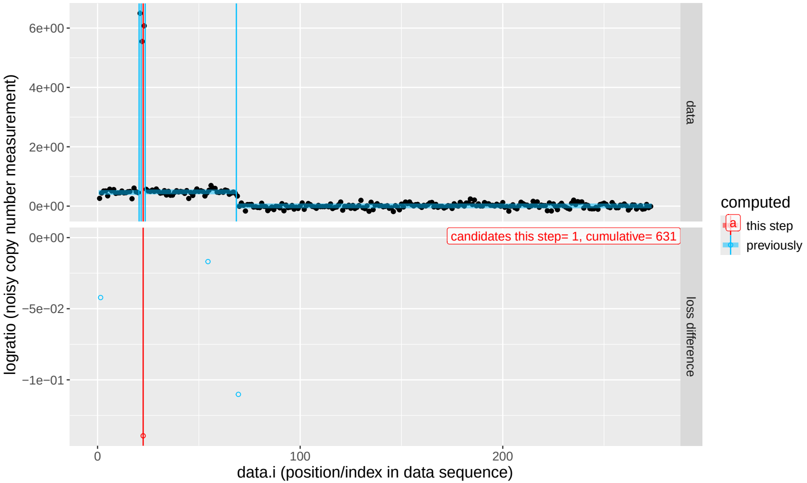
Visualization of computations at each iteration



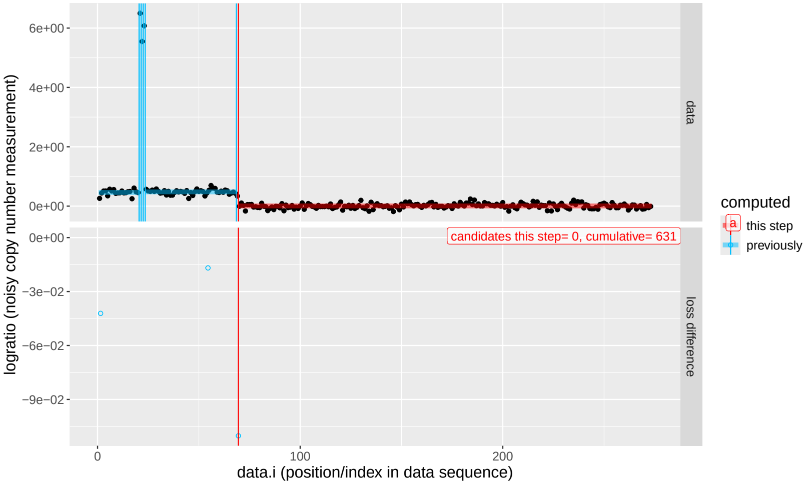
Visualization of computations at each iteration



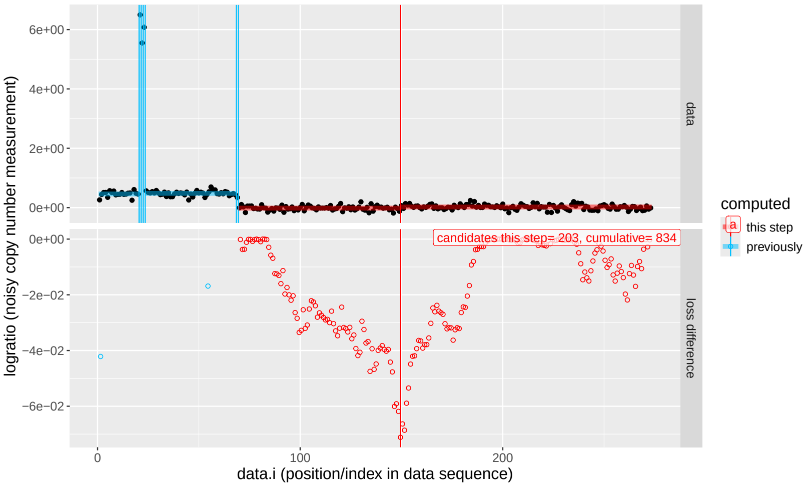
Visualization of computations at each iteration



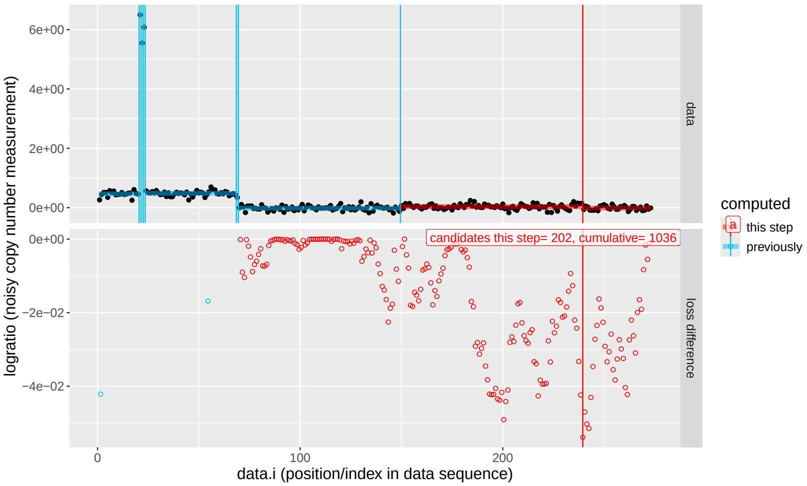
Visualization of computations at each iteration



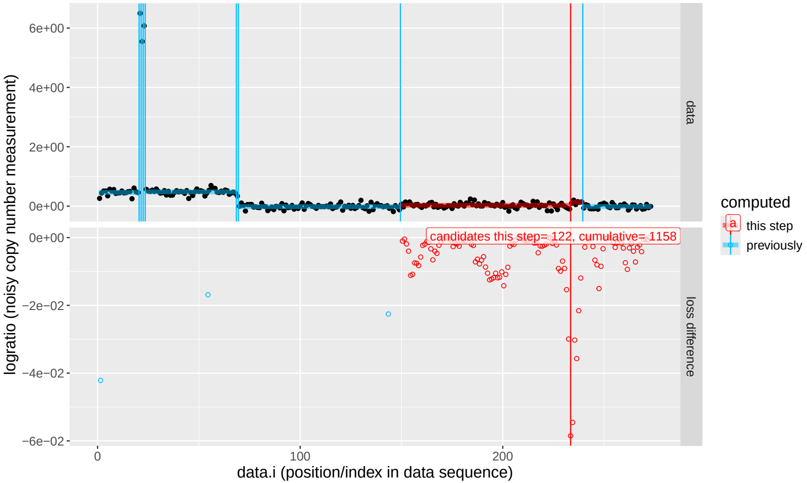
Visualization of computations at each iteration



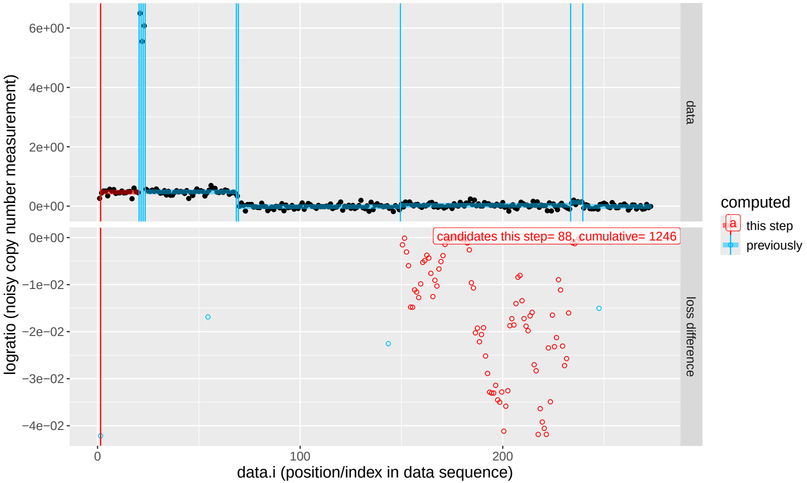
Visualization of computations at each iteration



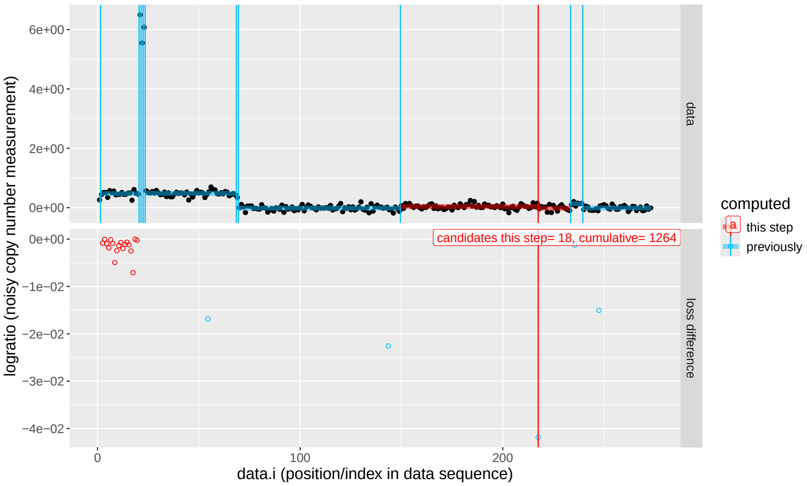
Visualization of computations at each iteration



Visualization of computations at each iteration



Visualization of computations at each iteration



Complexity analysis

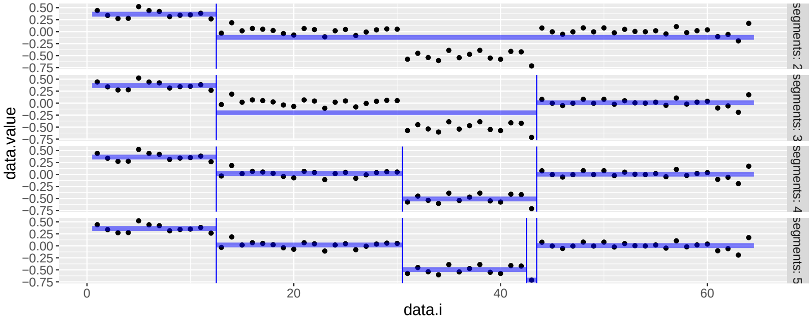
- Assume n data and K segments.
- Computing best loss decrease and split point for a segment with t data takes $O(t)$ time (for square loss).
- Keep a list of segments which could be split, sorted by loss decrease values.
- Best case is when segments get cut in half each time, $O(n \log K)$ time. (minimize number of possible splits for which we have to recompute loss)
- Worst case is when splits are very unequal $(1, t-1)$, $O(nK)$ time. (maximize number of possible splits for which we have to recompute loss)

Detailed complexity analysis

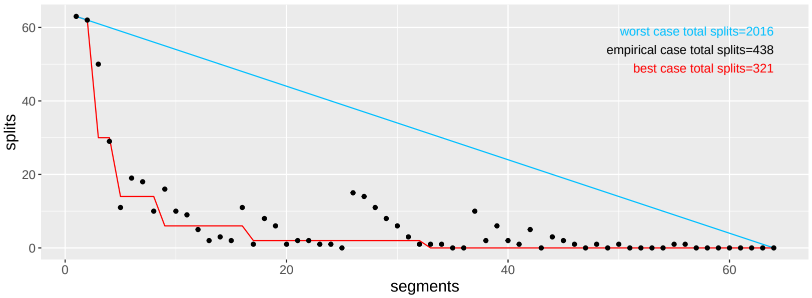
- Let $n = 2^J$ for some depth parameter $J \in \{1, 2, \dots\}$.
- For example $J = 6 \Rightarrow n = 64$ data.
- For any tree depth $j \in \{1, \dots, J+1\}$ if we do $K = 2^{j-1}$ iterations then how many split cost values to compute?
- Best case: $nj - 2^j + 1 = n(1 + \log_2 K) - 2K + 1 \Rightarrow O(n \log K)$.
- Worst case: $nK - K(1+K)/2 \leq nK \Rightarrow O(nK)$.

j	K	best	total	worst	total
1	1	$n-1=63$	$n-1=63$	$n-1=63$	$n-1=63$
2	2	$n-2=62$	$2n-3=125$	$n-2=62$	$2n-3=125$
	3	$n/2-2=30$		$n-3=61$	$3n-6=186$
3	4	$n/2-2=30$	$3n-7=185$	$n-4=60$	$4n-10=246$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
4	8	$n/4-2=14$	$4n-15=241$	$n-8=56$	$8n-36=476$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
7	64	$n/32-2=0$	$7n-127=321$	$n-64=0$	$64n-2080=2016$

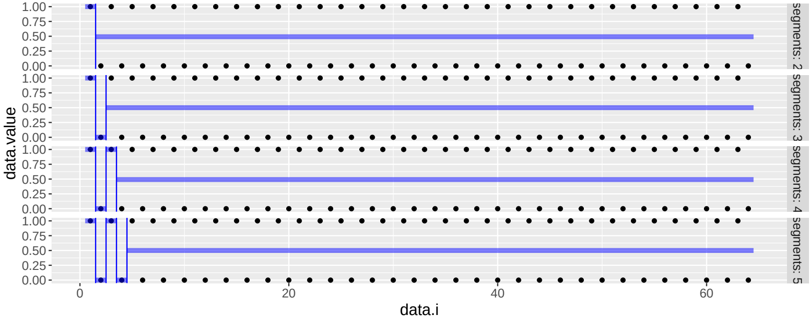
Real data time complexity analysis

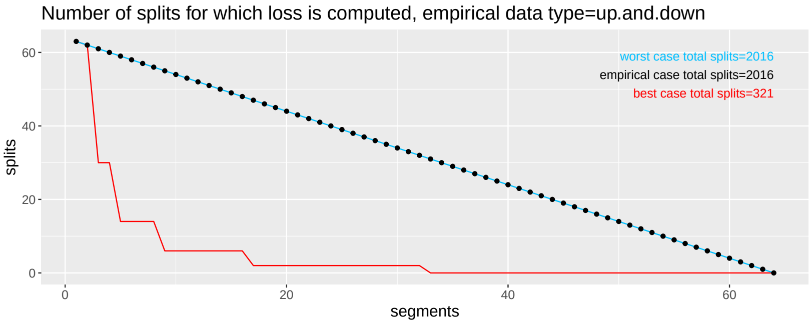


Number of splits for which loss is computed, empirical data type=real

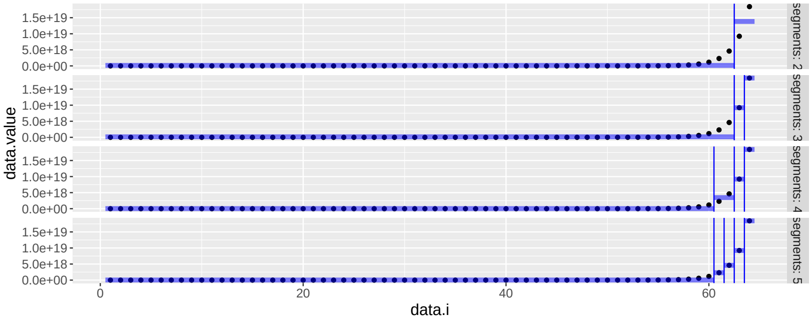


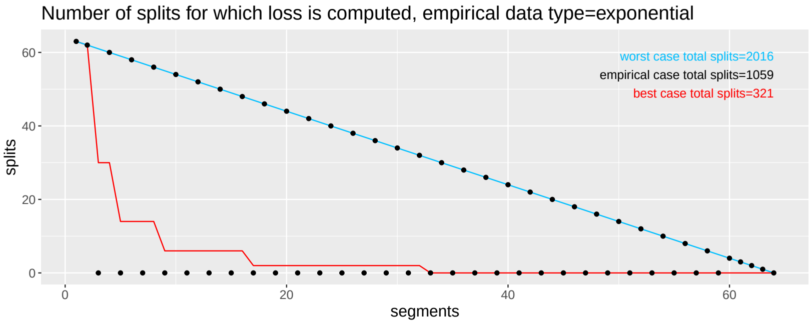
Synthetic data time complexity analysis



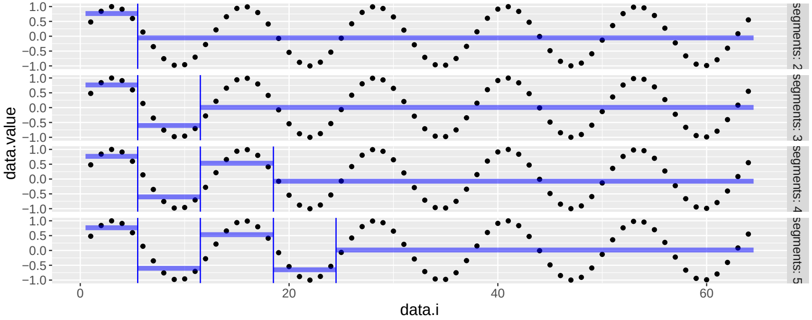


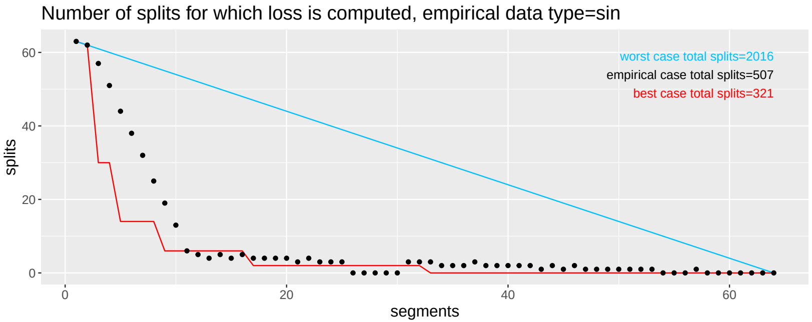
Synthetic data time complexity analysis



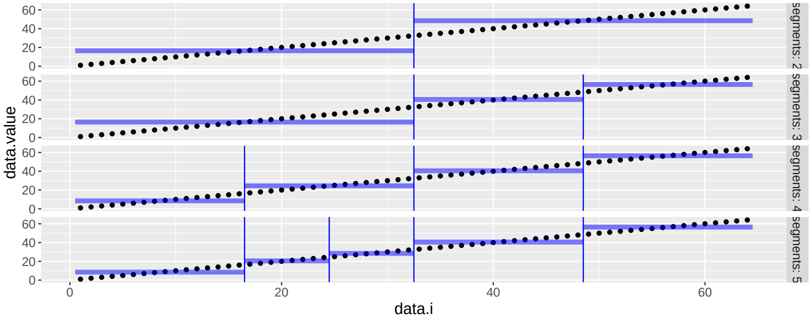


Synthetic data time complexity analysis

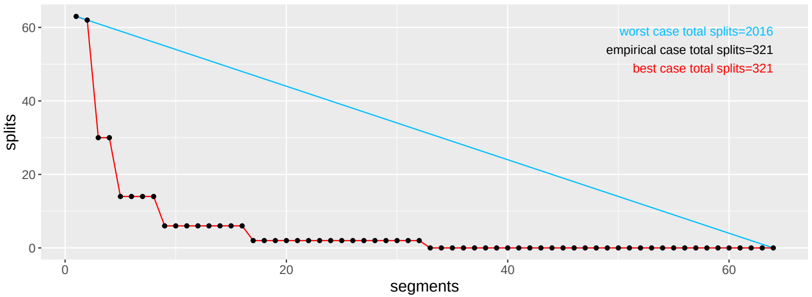




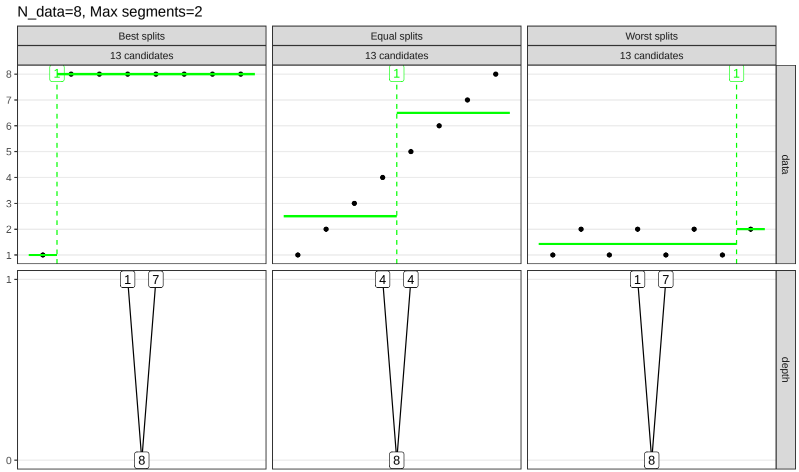
Synthetic data time complexity analysis



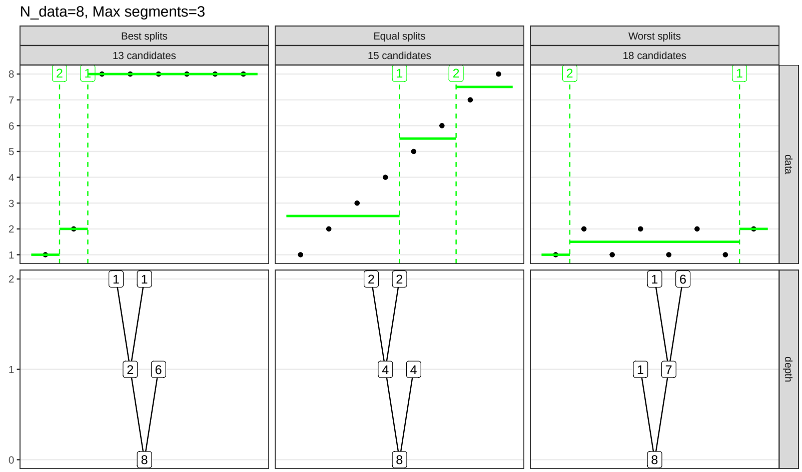
Number of splits for which loss is computed, empirical data type=linear



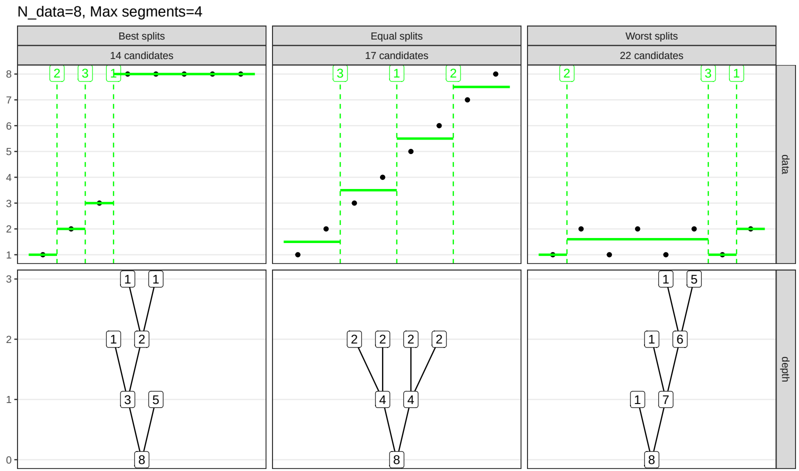
Synthetic data time complexity, vary max segments



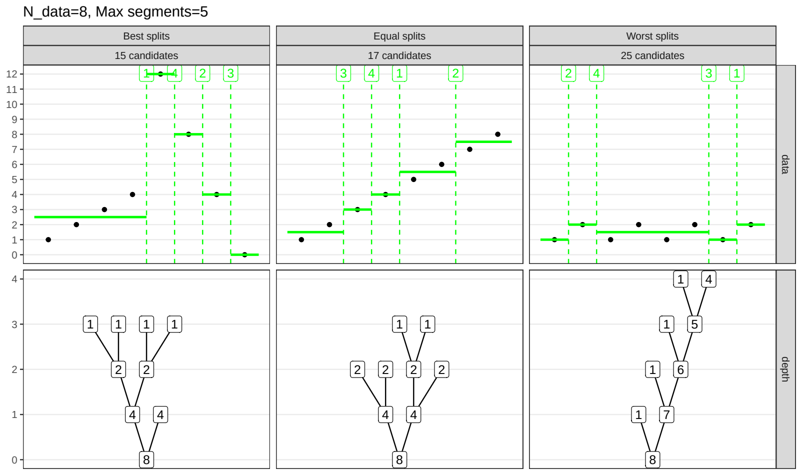
Synthetic data time complexity, vary max segments



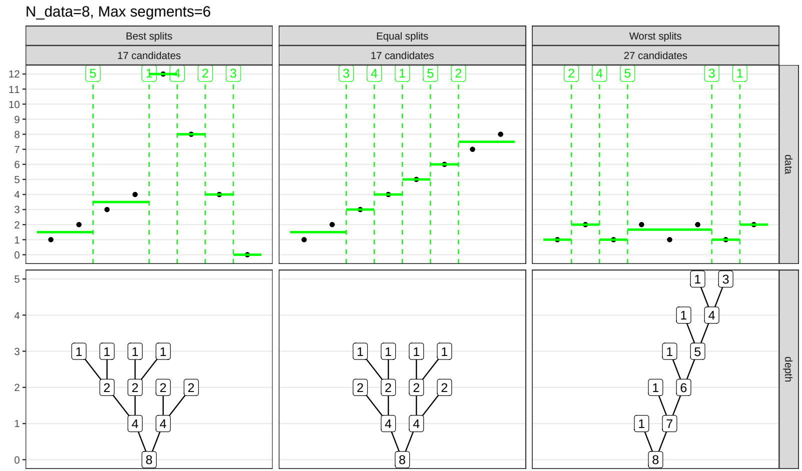
Synthetic data time complexity, vary max segments



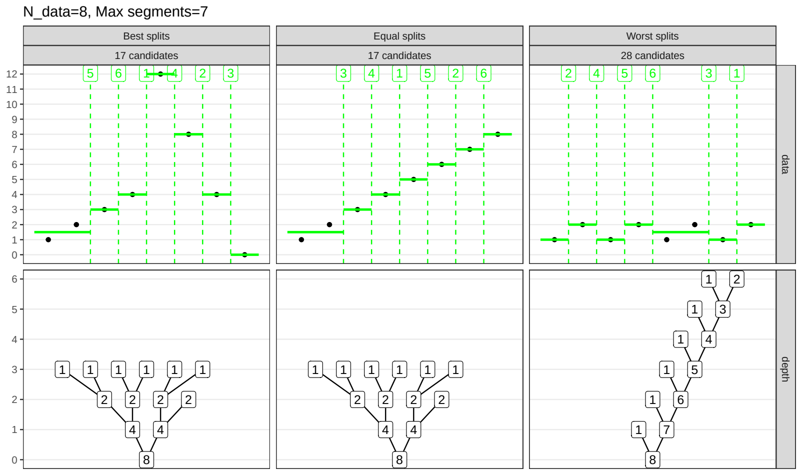
Synthetic data time complexity, vary max segments



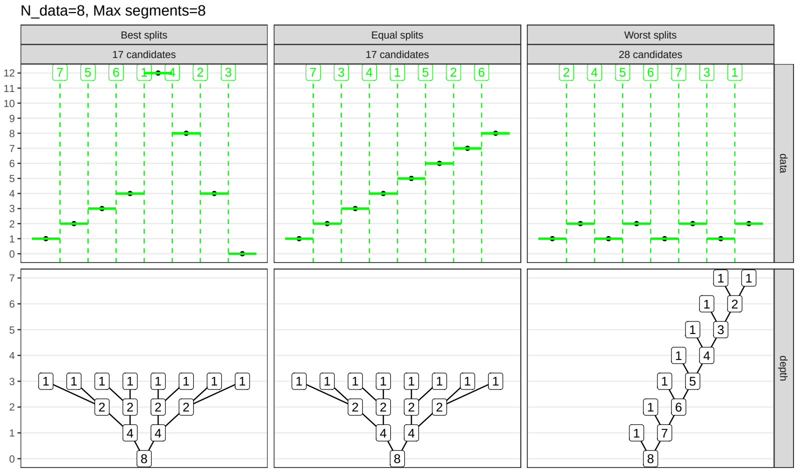
Synthetic data time complexity, vary max segments



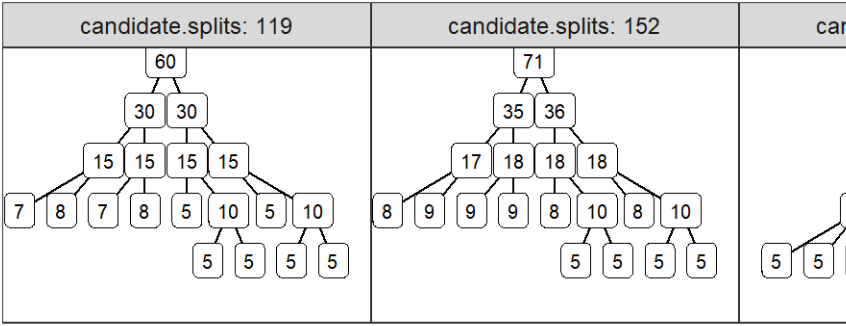
Synthetic data time complexity, vary max segments



Synthetic data time complexity, vary max segments

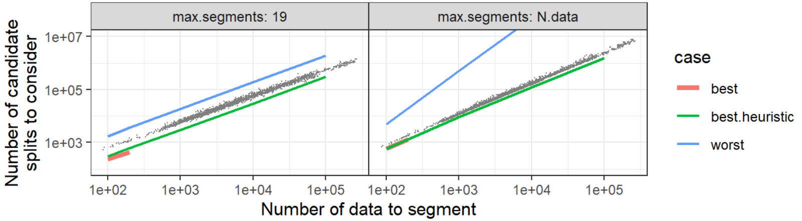


Optimal binary trees



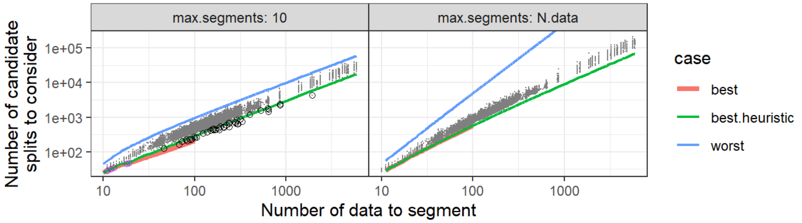
- $n \in \{60, 71, 72, 80\}$ data, min segment length $m = 5$, and number of splits/iterations $K = 9$.
- Smaller $n \in \{60, 71\}$ values result in a balanced first split,
- Larger $n \in \{72, 80\}$ values result in an unbalanced first split (one small child with no splits, one large child with all remaining splits).
- Hocking arXiv:2410.08654 Finite Sample Complexity Analysis

Empirical number of candidates (Poisson loss)



- 2752 real genomic count data sets from McGill benchmark of data size *n* from 87 to 263169, using binary segmentation with the Poisson loss.
- Number of candidate splits to consider in real data (grey dots) achieves the asymptotic best case, $O(n \log n)$.

Empirical number of candidates (Square loss)



- 13721 real genomic data sets from neuroblastoma benchmark of data size n from 11 to 5937, using binary segmentation with the square loss.
- Number of candidate splits achieves the asymptotic best case, $O(n \log n)$, and in 45 instances (black circles) requires fewer candidate splits than predicted by the best case heuristic.
- For max segments $= 10$, we used dynamic programming to compute the best case number of splits for all data sizes between 11 and 100 (orange line), and the heuristic (green line) was exact for only 5 data sizes $n \in \{11, 12, 13, 18, 19\}$ (purple circles).

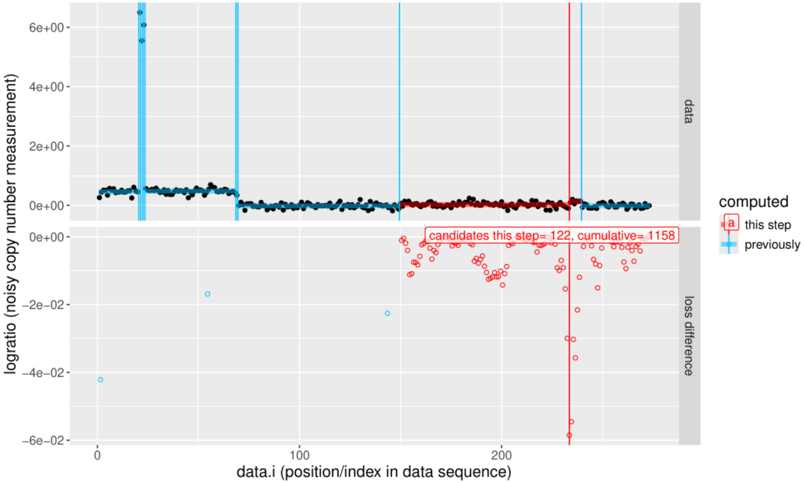
Introduction

Finite Sample Complexity Analysis of Binary Segmentation (arXiv:2410.08654)

Comparing binsegRcpp with other implementations of binary segmentation (under review at Journal of Statistical Software)

Two key steps in binary segmentation algorithm

- Compute best split on a segment (red).
- Find best segment to split (blue).



Analysis of insert time and storage (STL list)

To store previously computed best loss/split for each segment, use C++ Standard template library list as baseline.

- If list has p items then insert takes $O(1)$ time.
- Find best segment to split: $O(p)$ time.
- Total $O(K^2)$ time for K iterations (equal splits), larger than $O(n \log K)$ time for computing candidate loss values.

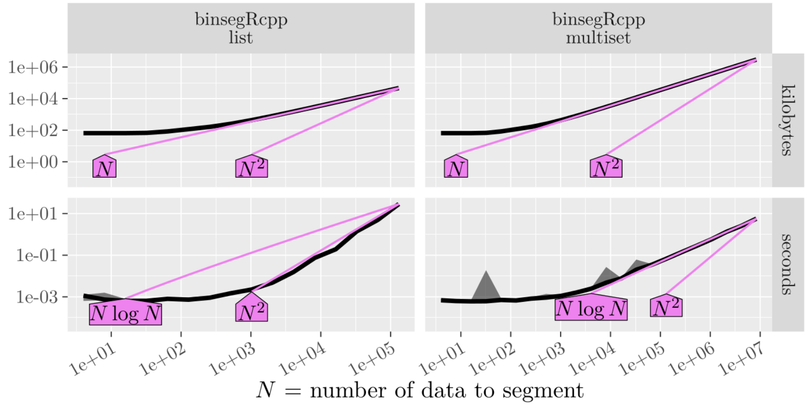
splits K	items in list (equal splits)	items in list (unequal splits)
1	1	1
2	2	1
3	3	1
$\vdots$	$\vdots$	$\vdots$
32	32	1
33	31	1
34	30	1
$\vdots$	$\vdots$	$\vdots$
62	2	1
63	1	1

Analysis of insert time and storage (STL multiset)

To achieve asymptotic best case timings, use a C++ Standard template library multiset, keyed by loss decrease. If multiset has p items then insert takes $O(\log p)$ time. Below: inserts column shows p for each insert, and size column shows p after inserts. Total $-\log(K-1) + \sum_{p=1}^{K-1} 2\log p \in O(K \log K)$ time over all inserts, smaller than $O(n \log K)$ time for split computation.

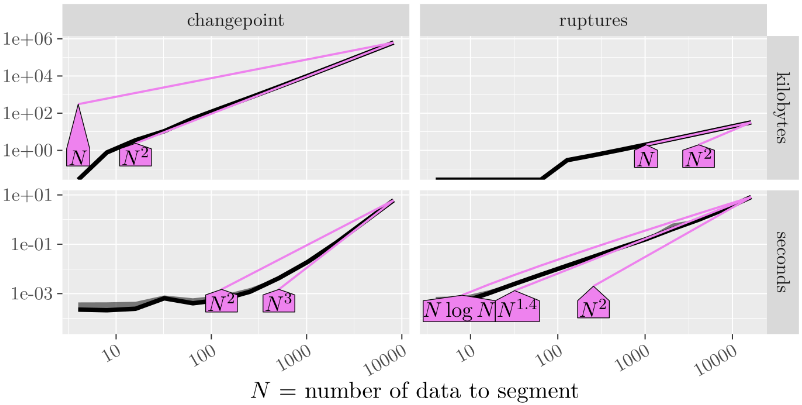
	equal splits		unequal splits	
iteration K	inserts	size	inserts	size
1	0	1	0	1
2	0,1	2	0	1
3	1,2	3	0	1
⋮	⋮	⋮	⋮	⋮
32	30,31	32	0	1
33		31	0	1
34		30	0	1
⋮	⋮	⋮	⋮	⋮
62		2	0	1
63		1	0	1

Timings (black) with asymptotic references (violet)



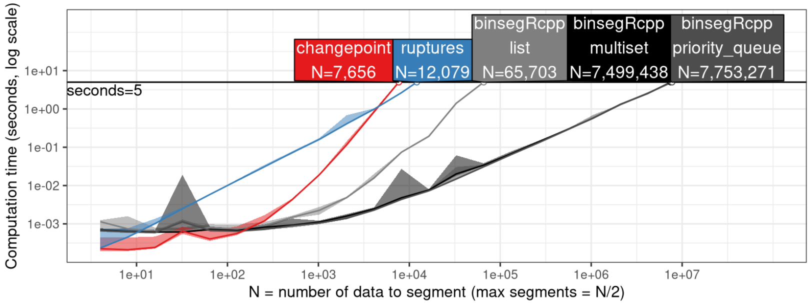
- Number of segments grows with data size, $K = O(N)$.
- Best case data, square loss.
- Linear space, $O(N)$.
- binsegRcpp multiset is $O(N \log N)$ time, asymptotically faster than binsegRcpp list, $O(N^2)$ time.

Timings (black) with asymptotic references (violet)



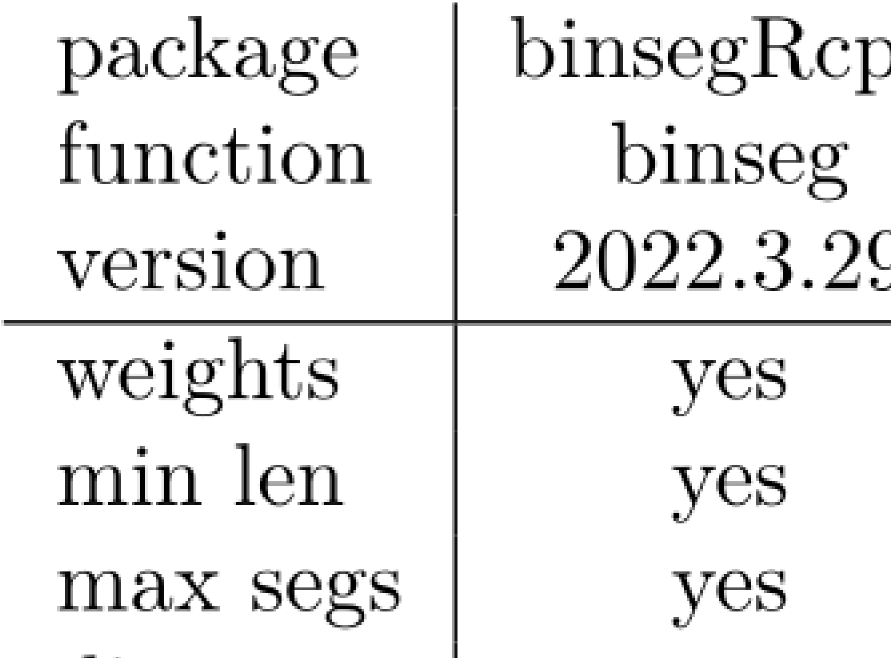
- Number of segments grows with data size, $K = O(N)$.
- Best case data, square loss.
- change-point is $O(N^2)$ space, $O(N^3)$ time.
- ruptures is $O(N)$ space, $O(N^{1.4})$ time (unclear why)
- Not asymptotic optimal complexity.
- Sub-optimal implementation.

Timing and throughput comparisons



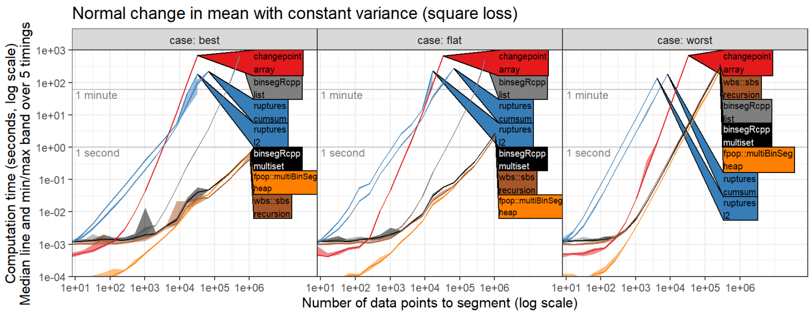
- Number of segments grows with data size, $K = O(N)$.
- Best case data, square loss.
- in 5 seconds, binsegRcpp using multiset or priority queue has 100x larger throughput than list.
- about 500x larger throughput than ruptures.
- about 1000x larger throughput than change-point.

Comparing implementations of binary segmentation



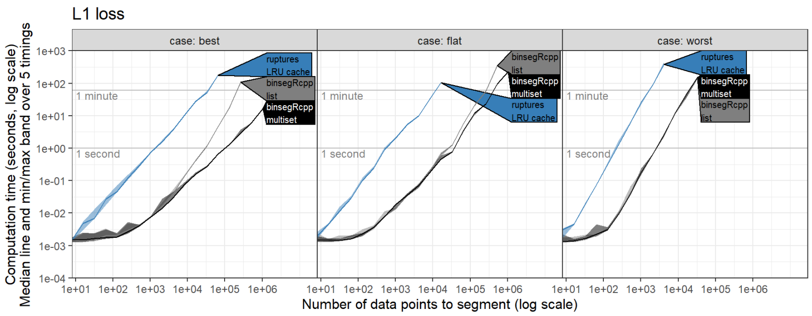
package	binsegRcp
function	binseg
version	2022.3.2
weights	yes
min len	yes
max segs	yes

Timings using square loss



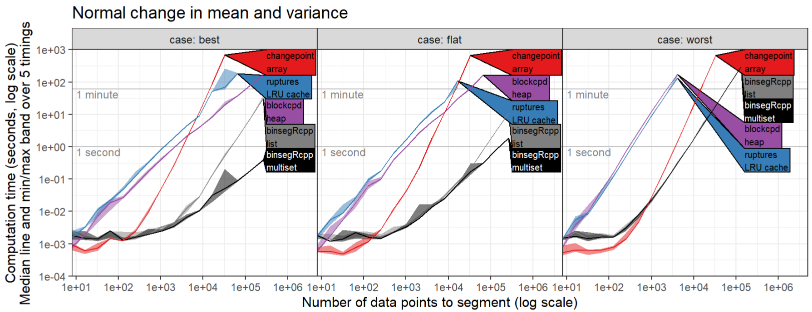
- binsegRcpp map, `wbs::sbs`, `fpop::multiBinSeg` optimal: $O(n \log n)$ time in best case.
- ruptures is between $O(n \log n)$ and $O(n^2)$ time, but with large constant factors.
- binsegRcpp list has larger slope: $O(n^2)$ time.
- change-point has even larger slope: $O(n^3)$ time.

Timings using L1 loss

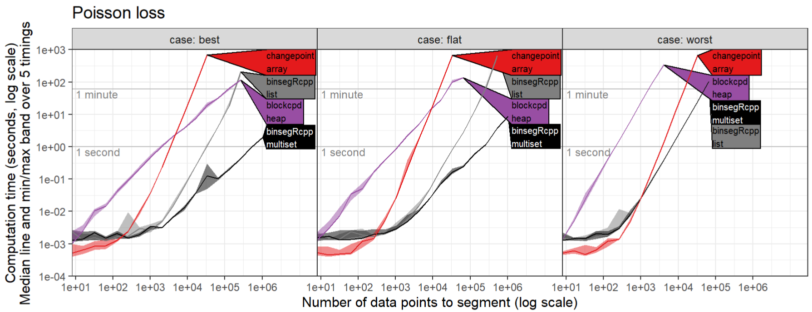


- Number of segments grows with data size, $K = O(n)$.
- Best case: binsegRcpp multiset is $O(n \log n)$, asymptotically faster than binsegRcpp list, $O(n^2)$.
- ruptures is significantly slower (100x).
- Worst case: all algorithms are quadratic, because that is the number of candidate change-points that must be considered.

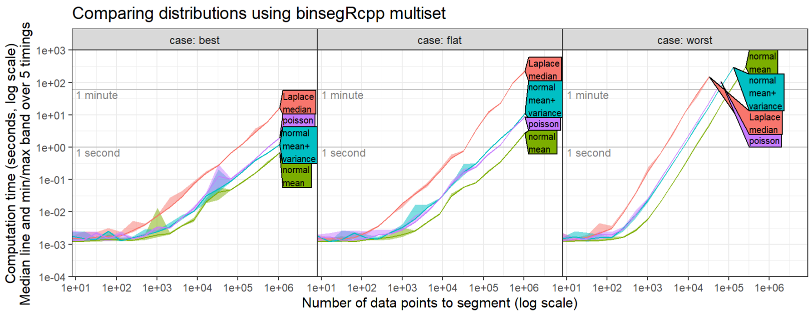
Timings using Gaussian change in mean and variance model



Timings using Poisson loss

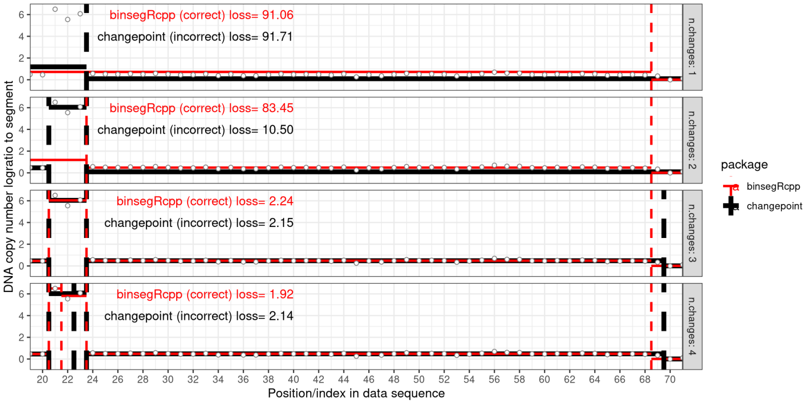


Comparing loss functions, binsegRcpp multiset



- Laplace (log-linear) is somewhat slower than others (linear).

Accuracy of library(binsegRcpp) and library(changepoint)



- Real data set for which change-point does not compute the correct model.
- Loss should be minimized at each iteration.

Conclusions

- Binary segmentation is an important algorithm for change-point detection in sequential data.
- Optimal binary trees used to determine best case number of candidates that the algorithm needs to consider.
- Empirical number of candidates in real data: same as best case, asymptotically.
- Proposed binsegRcpp R package provides C++ code which is asymptotically optimal speed, and correct.
- `install.packages("binsegRcpp")` in R.
- Future work: faster method for computing optimal binary trees? $O(n^2K^2)$ time for lower bound is slower than $O(nK)$ time for algorithm itself.

Comparison with previous algorithms from clustering

- Binary segmentation has segment/cluster-specific mean parameter, as in K-means and Gaussian mixture models. These algorithms attempt optimization of an error function which measures how well the means fit the data (but are not guaranteed to compute the globally optimal/best model).
- Binary segmentation is deterministic (different from K-means/GMM which requires random initialization). It performs a sequence of greedy minimizations (as in hierarchical clustering).
- Binary segmentation defines a sequence of split operations (from 1 segment to N segments), whereas agglomerative hierarchical clustering defines a sequence of join operations (from N clusters to 1 cluster). Data with common segment mean must be adjacent in time/space; hierarchical clustering joins may happen between any pair of data points (no space/time dimension).

Possible exam questions

- Explain in detail one similarity and one difference between binary segmentation and k-means. (gaussian mixture models, hierarchical clustering)
- For a sequence of $n = 10$ data, we need to compute the loss for each of the 9 possible splits in the first iteration of binary segmentation. What is the number of splits for which we must compute the loss in the second/third steps? (best and worst case)