Algorithmics and GPU

Variance multiple change points detection

Prepared by :

Ennatiqi Mohamed Vhiny Mombo Ayoub Soukri

Evry University
Paris-Saclay university
M2 Data Science
Academic year :2021/2022

Abstract

In this project, we are considering the problem of variance multiple change points detection, this problem has many applications espacially in finance, finantial computations and algorithms are based in the fisrt place on the quantification of variance. Detecting the points in which the variance is changing will be very helpful for this kind of applications. In this report, we are comparing three different algorithms, Binary segmentation, Optimal partitionning-PELT and a Naïve approach. The comparison is in terms of complexity and time execution, and to do so we are implementing the solutions using R and $C++$.

Keywords :change points, variance, PELT, binary segmentation, naive approach

1 Introduction

For the project of algorithmics and GPU, our choice was to build and compare many algorithms in terms of complexity and precisions in detecting variance change points in a timeseries data. This choice was based on the fact that timeseries data are the ingredients of any data scientist working in financial field, so it will be verry important for us to be able to deal with this kind of problems. As we know, volatility is the degree of variation in the log-returns of a treading asset (or a time series in general), it is measured by the strandard deviation.

Our aim is to compare three algorithms, the first is based on a naîve approach (simple and far from being optimal), then the second is based on a recursion approach and a segmentation of the data sequence, it is called binary segmentation. The third one is the optimal partitionning and its variation PELT algorithm.

To do so, we undestood the theoritical concepts behind each of these algorithms, an then we will be coding theim in R, then comparing the C++ (Rcpp) version with R version, and finally we will set a benchmarking based on the performances of the algorithms and the results of integrated R packages for multiple change points detection.

2 Naive approach

2.1 Generalities

The naive algorithm in question has a basic and approximative approach ,although it lacks in accuracy ,yet it can be really useful for analytical uses , it aims to create sub-sequences or better known as windows that move alongside the dataset calculating its parameters , it is used in many other ways and for various purposes (Mean , variance , regression..) , so in our case we are only concerned with the variance in order to detect multiple change points.

More precisely , we'll introduce a double window with a given width (W) , this window will calculate the variance between its sub-windows as it moves forwards in the dataset and doing the same procedure in each step returning the values which will be used later on to detect the change as we'll have distribution of the behaviour related to our variance .

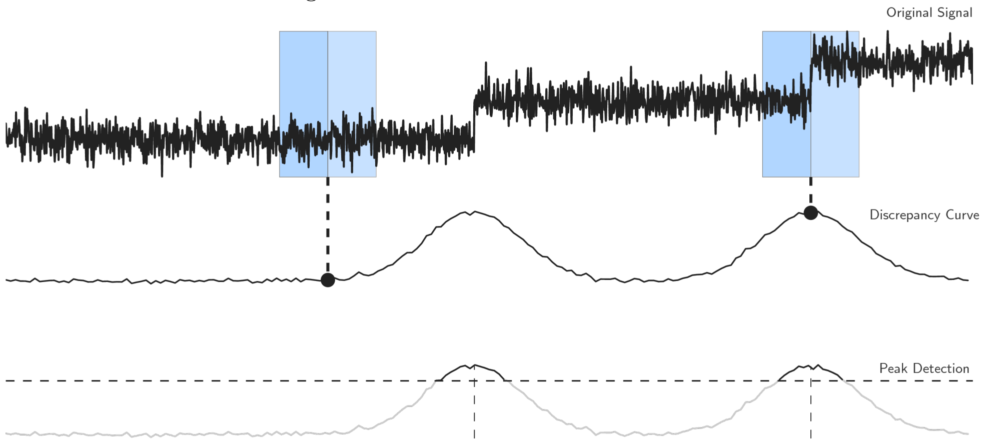
There are generally many ways to study the behaviour of the variance given through the sliding window ,in our case we'll consider 2 of which we'll choose one to work with but we'll explain both.

First let us introduce our sliding double window as the following :$S(k, w)$

For a given window width (w) ,we have a sliding double-window rolling through $[k + 1, k + w]$ with a one step progress, where k is its index

Once we set up our window , we'll have to define a statistical method to calculate the Discrepancy between the sub-windows , where the changepoints are naturally its maximal values or better known as Peaks as Figure 1 describes them :

FIGURE 1 – Sliding Double-Window and The Parameters Function



A simple intuitive way to get the discrepancy is through calculating the empirical variance of each sub-window and substracting one from the other , keeping its absolute value :

$$\{S_D(k, w)_{w \leq k \leq n-w} \mid S_D(k, w) = \mid \sigma(k, w) - \sigma(k - w, w) \mid\}$$

Another approach to consider and which we'll base our algorithm on , is to perform a fisher's test on the sub-population of our double window :

$$\{S_F(k,w)_{w\leq k\leq n-w} \mid S_F(k,w) = F_{test}((k,w),(k-w,w))\}$$

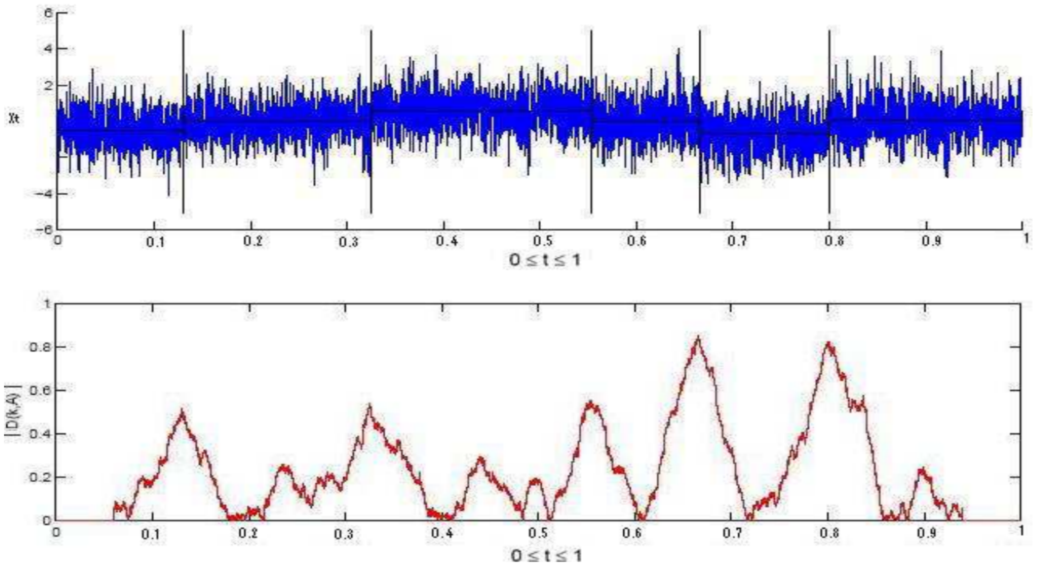
In both cases , our interest is the maximum points of these functions in order to find the change points :

In order to get these maximal values (Peaks), we have to set a threshold value (l) such that :

$$\underset{k\in[w,n-w]}{max} S_D(k,w) = \mid S_D(k,w) \mid > l$$

$$\underset{k\in[w,n-w]}{max} S_F(k,w) = S_F(k,w) > l$$

Figure 2 – Dataset and The S function



This is the algorithm we'll apply first to get our vector of potential change points :

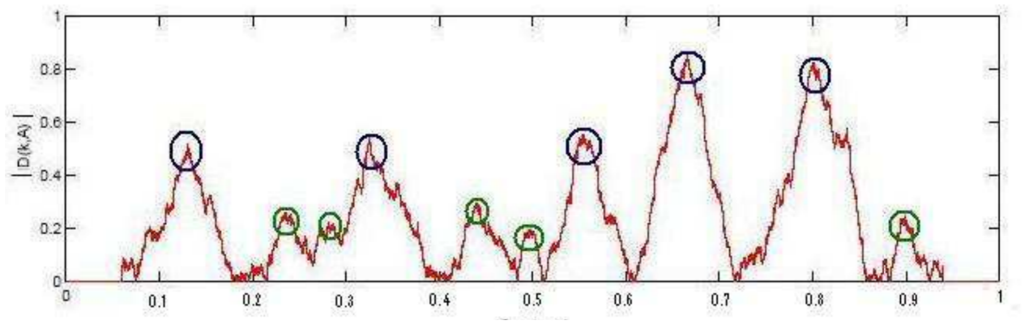
Algorithm 1 Naive Approach

```
Data: X,w,l
Result: Points de changement
$\tau = 0 \;\; \tau_k = argmax S_F(k,w)$
while $S_F(k,w) > l$ do
    $k = k+1$
    $S_F(\tau,w) = 0$ for $k \in (\tau - w, \tau + w)$
    $\tau = argmax S_F(k,w)$
    Incrementation of the change point counter
end
Vector=$\{\tau_1 ...., \tau_{max}\}$
```

We got our vector containing potential change points , but there is an issue , not all these points represent the real change points and as we can see in Figure 4 , only the blue points are the real change points , the rest aren't.

So in order to remove the wrong change points , we start by running tests in our vector of potential change points through two successive change points and whether the gap between them is noticeable , otherwise we just remove them and keep the one we start at , we keep doing so until we move to the next point or we can even set a value to this gap (for example 2*Window's length)

Figure 3 – Real change points vs False change points



2.2 Complexity

We can divide the algorithm into 3 parts to make the complexity analysis easier , first the sliding window that performs the F-test as $\mathcal{O}(n)$, then we have the second part of getting a vector of the maximal values (Potential changepoints) which is also of a complexity of $\mathcal{O}(n)$, then we have the last step of getting the real change points and getting rid of the false ones which is also of a complexity of $\mathcal{O}(n)$

3 Binary segmentation

3.1 Generalities

The second algorithm is the binary segmentation, this method is based on a sequential approach, in which we implement in a recursive way a test of hypothesis, it's widely used especialy for change point detection (mean and variance).

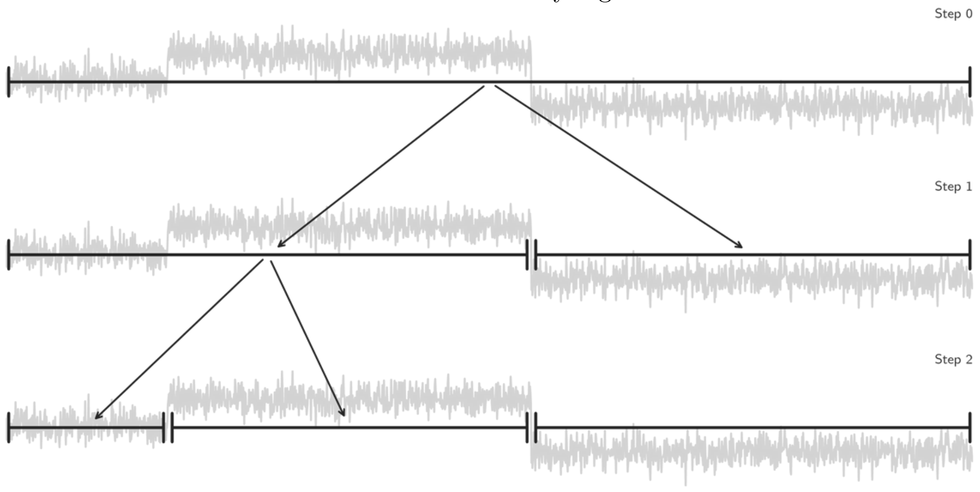
The first step is to detect one change point complete input Y, then Y is splited around this first detected change point, then the operation is repeated on the two resulting sub-vectors.

The advantage of binary segmentation includes low complexity $(O(C \log n))$, where n is the number of samples and C the complexity of calling the considered cost function on one sub-signal, the fact that it can extend any single change point detection method to detect multiple changes points and that it can work whether the number of regimes is known beforehand or not.

In this project, we were based on the article of Carla INCLAN and George C. TIAO and the article of Killick, R., Fearnhead, P. and Eckley, I.A. The first article discussed how we can apply the CSS (Cumulative Sum of Squares) to detect change points, and the second was about the use of a penality and a cost function to implement the same method, but the approach is the same : the fast signal segmentation.

The classical binary segmentation try in general to find a set of change points in variance, this set of points is found based on an iterative operation which consists on searching the points one by one :

FIGURE 4 – Illustration of binary segmentation idea



The algorithm is using the statistic of cumulated sum of squares Dk defined by :

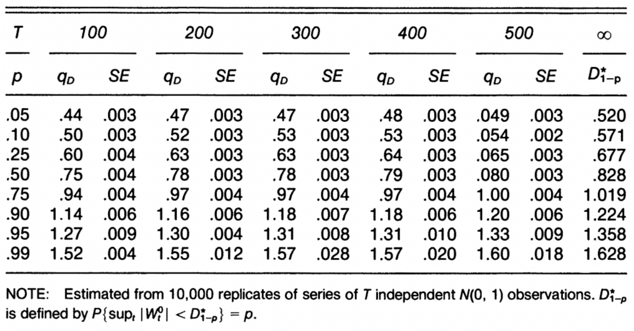
$$D_k = \frac{C_k}{C_T} - \frac{k}{T}$$

with $C_k = \sum_{i=1}^{k} y_i^2$, $k = 1...T$ and $D_0 = D_T = 0$

The plot of D_k will go out of certain critical boundaries calculated by the asymptotic behaviour of the distribution D_k assuming a constant variance. It is important to note that D_K is behaving like a brownian bridge.

The value of the critical boundaries is given in the table below :

FIGURE 5 – Empirical and asymptotic quantiles of $max\sqrt{T/2} \mid D_k \mid$



T	100		200		300		400		500		∞
p	q_D	SE	q_D	SE	q_D	SE	q_D	SE	q_D	SE	D^*_{1-p}
.05	.44	.003	.47	.003	.47	.003	.48	.003	.049	.003	.520
.10	.50	.003	.52	.003	.53	.003	.53	.003	.054	.002	.571
.25	.60	.004	.63	.003	.63	.003	.64	.003	.065	.003	.677
.50	.75	.004	.78	.003	.78	.003	.79	.003	.080	.003	.828
.75	.94	.004	.97	.004	.97	.004	.97	.004	1.00	.004	1.019
.90	1.14	.006	1.16	.006	1.18	.007	1.18	.006	1.20	.006	1.224
.95	1.27	.009	1.30	.004	1.31	.008	1.31	.010	1.33	.009	1.358
.99	1.52	.004	1.55	.012	1.57	.028	1.57	.020	1.60	.018	1.628

NOTE: Estimated from 10,000 replicates of series of T independent $N(0, 1)$ observations. D^*_{1-p} is defined by $P\{\sup_t |W_t^0| < D^*_{1-p}\} = p$.

It is important also to notice that if the data is supposed to be normally distributed with mean 0 and variances σ_i^2 with $i = 1...T$, the raltioship between the log-likelihood and the metric D_k, and it is given by the formula :

$$LR_{0,1} = -max_k(-\frac{k}{2}log(1 + \frac{T}{k}D_k) - \frac{T-k}{2}log(1 - \frac{T}{T-k}D_k))$$

$LR0,1$ is the log-likelihood ratio between getting of not a change of variance.

3.2 Implementation in R

Algorithm 2 BS

```
Data: s, e, Y, penality
Result: List of all variance change points
Stop condition : $e - s = 1$
if e-s ==1 then
    Return(empty list)
else
    stat$_{s,...,e}$ = $css$(Y$_{s,...,e}$)
    changePoint = Argmax$_{k \in s,...e}$($stat_{s,...,e}$) + $s$
    if max(css)$\sqrt{\frac{e-s+1}{2}}$ > penality then
        return(BS(s,changePoint-1,Y,penality), changePoint, BS(changePoint+1,e,Y,penality))
    else
        return(empty list)
    end
end
```

Where the css function is just the values of D_k and the penality is the value given in the table by the asymptotic distribution of D_k.

To generate the data, we used the package *gfpop*, Y is a vector of normal distributed random variable, with mean zero and different variances. The position in which the variance will be changed is fixed in advanced so that we could compare the true positions with the detected change points.

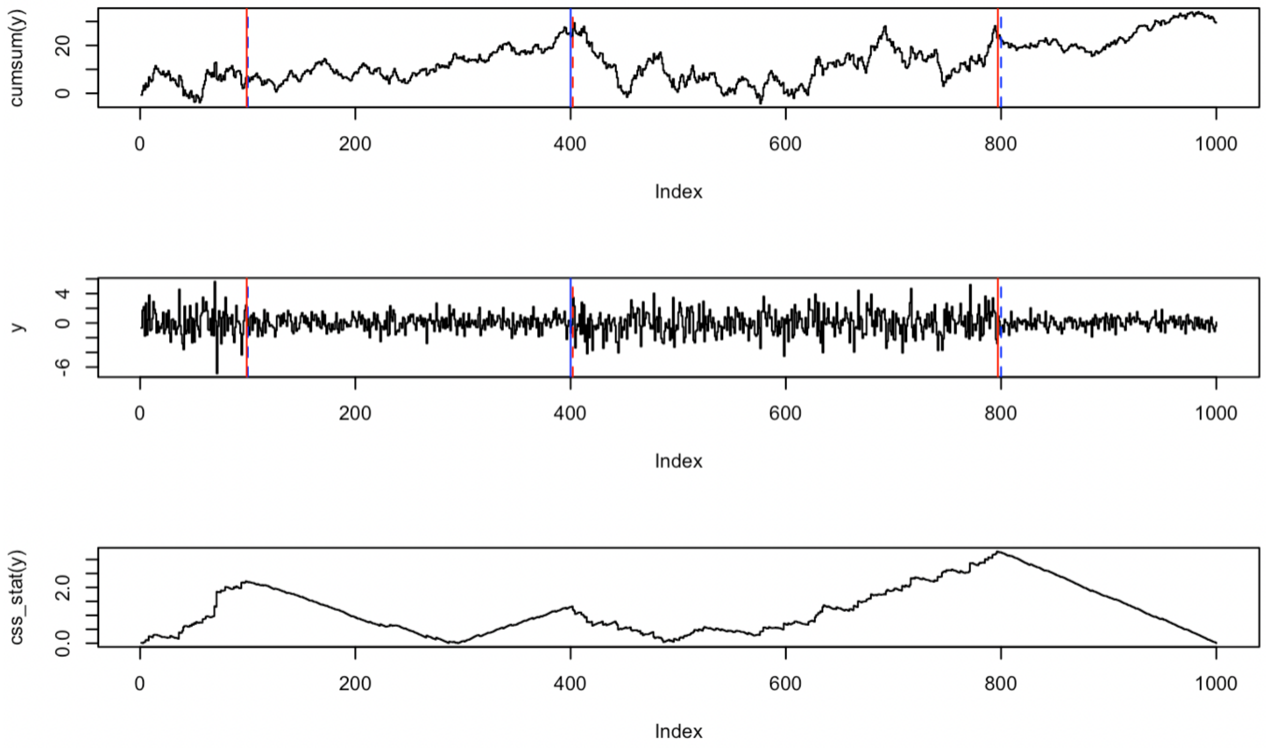


FIGURE 6 – Red : true change points, blue : detected change points

3.3 Complexity of the algorithm

As seen in the generalities, the complexity of the binary segmentation is basically of the order $\mathcal{O}(n \log n)$.
The plot of the complexity compared with the plot of the function $T(n) = n \log n$ are proportional, in fact, the constant of the scale depend on the use of note of Rcpp in the implementation, and also on the complexity (even if it is constant) of the elementary instruction of the code, and we can clearly see that in the figures of the next section.

If we consider the recursive algorithm as a tree in which we divide the process in each level, The top level has cost cn, The next level down has 2 subproblems, each contributing cost $cn/2$, The next level has 4 subproblems, each contributing cost $cn/4$. Each time we go down one level, the number of subproblems doubles but the cost per subproblem halves. Therefore, cost per level stays the same. The height of this recursion tree is $log(n)$ and there are $log(n) + 1$ levels.

Total cost is sum of costs at each level of the tree. Since we have $log(n) + 1$ levels, each costing cn, the total cost is $cn \log(n) + cn$. Ignoring lower order term n, we can say the complexity is

$\mathcal{O}(n \log n)$.

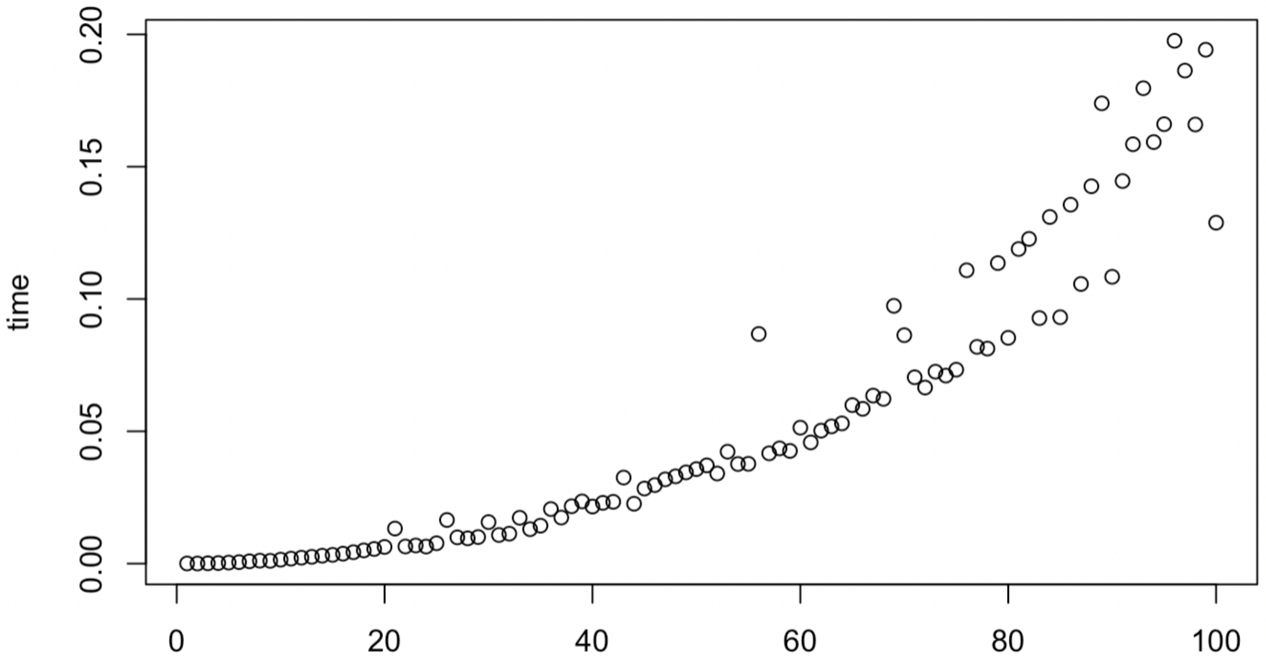


FIGURE 7 – complexity simulatione n=10000

3.4 Implementation using RCPP

Always in order to optimize our code, we use the RCPP package to write our functions in C++ language and integrate it with R code. First we wrote all functions written already using R in C++, then we use the Rcpp package to integrate the functions in the code.

First we did a simulation with $n = 10000$ and 3 change points, the results are surprising, the difference in time is large :

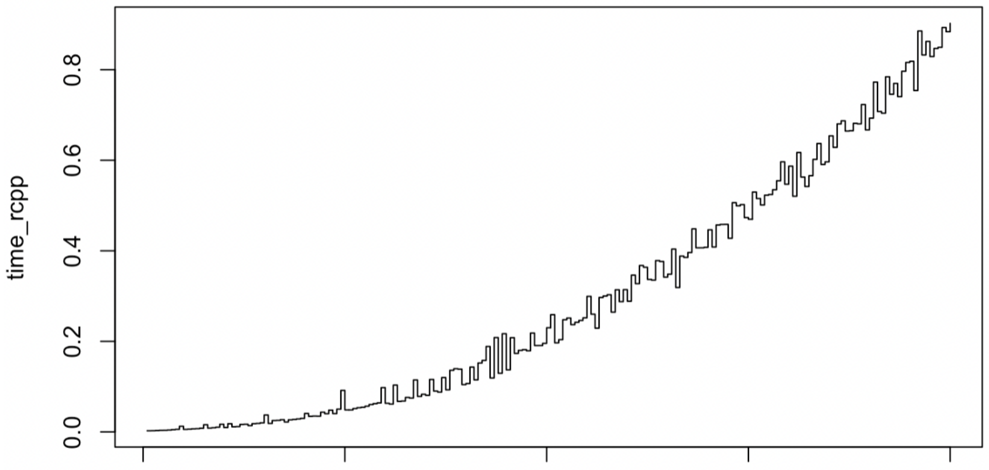


FIGURE 8 – time execution with Rcpp code with n=500 : 10000

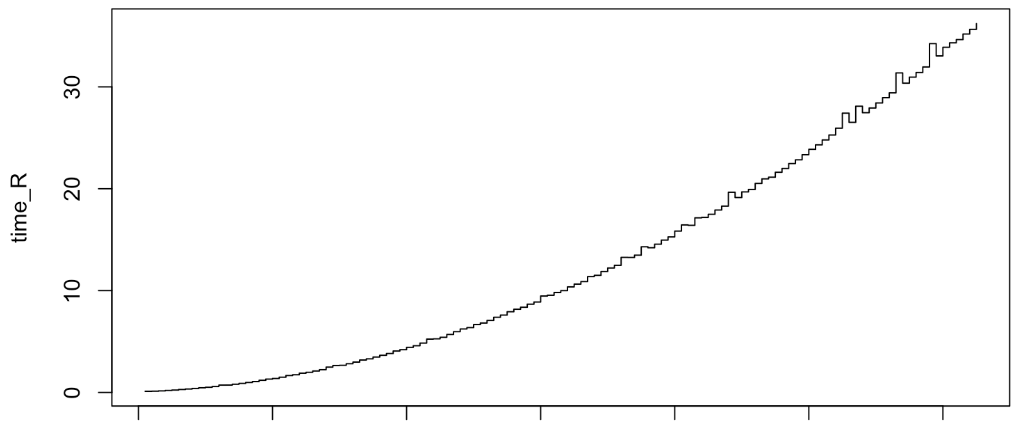
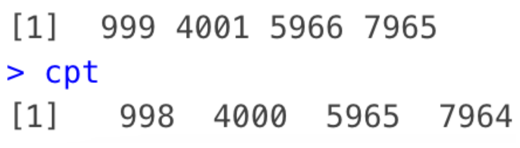


FIGURE 9 – time execution with R code with n=500 : 10000

Another test that we did, is to compare the results of the coded version in Rcpp and R with an already integrated R package. and these are some results :



```
[1]  999 4001 5966 7965
> cpt
[1]   998  4000  5965  7964
```

FIGURE 10 – Comparaison between our BS and the package n=10000

4 Optimal partitioning - PELT

4.1 Generalities

The algorithm of Optimal partitioning and Pruned Exact Linear Time (PELT) are two exact method for the search of change point in sequences. It is based on the minization of the objective function :

$$\sum_{i=1}^{m+1} \mathcal{C}(y_{(\tau_{i-1}+1):\tau_i}) + \beta$$

where $\mathcal{C}$ is a cost function for a segment and β is a penalty to prevent the over fitting. Although many cost functions such as quadratic loss and cumulative sums can be used, in the litterature, the most common is twice the negative log likelihood. For the choice of the penalty is depend on the problem (change in variance or change in mean), but generally it is some statistics functions such as Akaike's Information Criterion (AIC) (β = 2p) and Schwarz Information Criterion (SIC (β = p log n), where p is the number of additional parameters introduced by adding a changepoint. Let consider, $\mathcal{T}_s = \{\tau_1, \tau_2, ...\tau_m, \tau_{m+1}\}$ the set of change point and $F(s)$

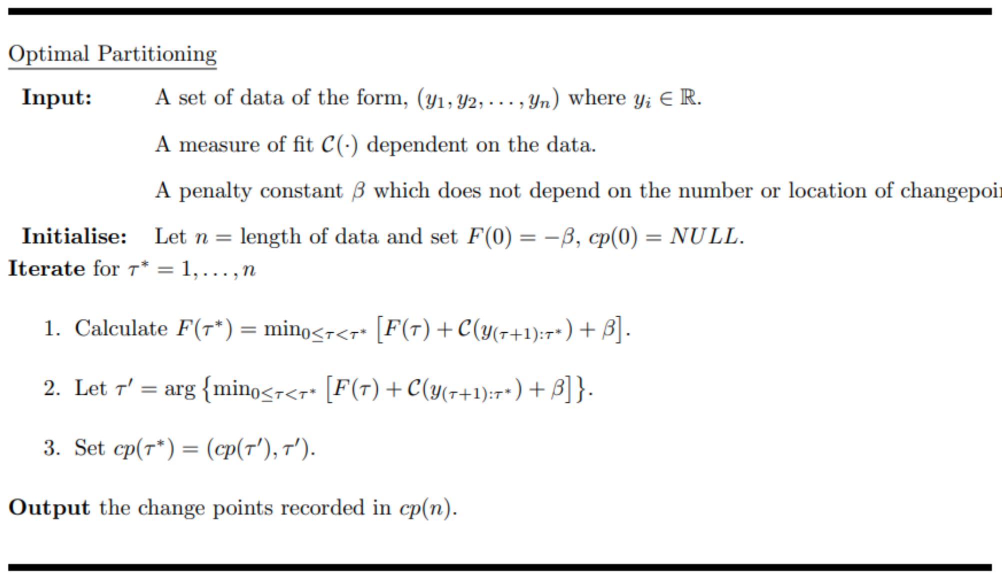
the minimization of the objective function for a sequence $y_{1:s}$ can be rewritten as :

$$F(s) = \min_{\tau \in \mathcal{T}_s} \left\{ \sum_{i=1}^{m+1} \mathcal{C}(y_{(\tau_{i-1}+1):\tau_i}) + \beta \right\} \tag{4.1}$$

$$= \min_{t} \left\{ \min_{\tau \in \mathcal{T}_t} \sum_{i=1}^{m} \left[\mathcal{C}(y_{(\tau_{i-1}+1):\tau_i}) + \beta \right] + \mathcal{C}(y_{(t+1):n}) + \beta \right\} \tag{4.2}$$

$$= \min_{t} \left\{ F(t) + \mathcal{C}(y_{(t+1):n}) + \beta \right\} \tag{4.3}$$

With a complexity of $\mathcal{O}(n^2)$ the Optimal Partitioning algorithm proposed by Yao (1984) and Jackson et al. (2005) solve this problem the following pseudo code :



```
Optimal Partitioning
Input:       A set of data of the form, $(y_1, y_2, \dots, y_n)$ where $y_i \in \mathbb{R}$.
             A measure of fit $\mathcal{C}(\cdot)$ dependent on the data.
             A penalty constant $\beta$ which does not depend on the number or location of changepoints.
Initialise:  Let $n$ = length of data and set $F(0) = -\beta$, $cp(0) = NULL$.
Iterate for $\tau^* = 1, \dots, n$
    1. Calculate $F(\tau^*) = \min_{0 \le \tau < \tau^*} \left[ F(\tau) + \mathcal{C}(y_{(\tau+1):\tau^*}) + \beta \right]$.
    2. Let $\tau' = \arg \left\{ \min_{0 \le \tau < \tau^*} \left[ F(\tau) + \mathcal{C}(y_{(\tau+1):\tau^*}) + \beta \right] \right\}$.
    3. Set $cp(\tau^*) = (cp(\tau'), \tau')$.
Output the change points recorded in $cp(n)$.
```

FIGURE 11 – Optimal Partioning

In addition to having a greater computation complexity than Binary Segmentation, this algorithm is less accurate. In order to reduce the complexity, Killick R. & Al[1] introduce the idea of pruning by showing that for there exists real K, if $F(t) + \mathcal{C}(y_{(t+1):s}) + \beta \geq F(s)$ holds for any $T > s$, t $(\leq s)$ can never be the optimal last changepoint prior T. This condition reduce the set of changepoints candidate to those verifying the contrary of this inequality for the next generation.It can be implemented into the OP method and the pseudo-code through the pseudo code(figure 12)

Thus, OP and PELT give the same estimation but PELT decrease the computation complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. We apply these algorithms on data a sequence of simulated data(figure 13)

```
PELT Method
Input:      A set of data of the form, $(y_1, y_2, \dots, y_n)$ where $y_i \in \mathbb{R}$.
            A measure of fit $\mathcal{C}(.)$ dependent on the data.
            A penalty constant $\beta$ which does not depend on the number or location of changepoints.
            A constant $K$ that satisfies equation 4.
Initialise: Let $n$ = length of data and set $F(0) = -\beta$, $cp(0) = NULL$, $R_1 = \{0\}$.
Iterate for $\tau^* = 1, \dots, n$
    1. Calculate $F(\tau^*) = \min_{\tau \in R_{\tau^*}} \left[F(\tau) + \mathcal{C}(y_{(\tau+1):\tau^*}) + \beta\right]$.
    2. Let $\tau^1 = \arg\left\{\min_{\tau \in R_{\tau^*}} \left[F(\tau) + \mathcal{C}(y_{(\tau+1):\tau^*}) + \beta\right]\right\}$.
    3. Set $cp(\tau^*) = [cp(\tau^1), \tau^1]$.
    4. Set $R_{\tau^*+1} = \{\tau \in R_{\tau^*} \cup \{\tau^*\} : F(\tau) + \mathcal{C}(y_{\tau+1:\tau^*}) + K \le F(\tau^*)\}$.
Output the change points recorded in $cp(n)$.
```

FIGURE 12 – PELT

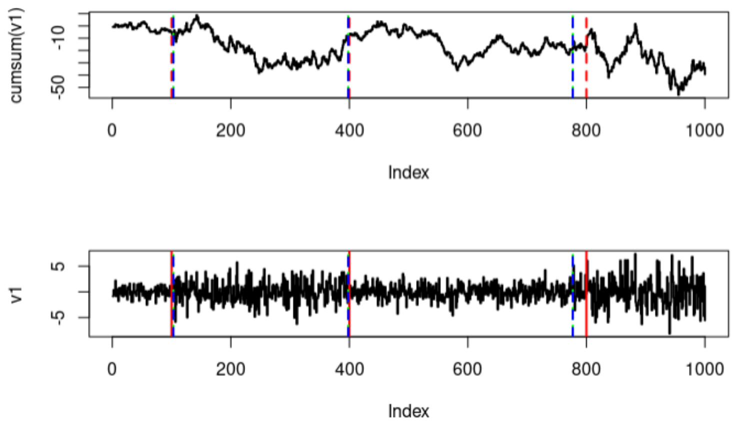


FIGURE 13 – Change point estimated (*green* is for OP and *blue* is PELT) vs true data (Red)

4.2 Complexity of the algorithms

We have seen that OP is a quadratic complexity algorithm whereas its optimized version PELT is linear theoritically. In this section, we are trying to show it pratically through our implementation in R and C++. As said early, for the C++ version, we used the RCPP package to interface it.

For the experiment we did a simulation with a sequence of 5000 samples, it shows that, the C++ version is far quicker than the R one in term of execution time, that demonstrates the drop in complexity between OP and PELT.

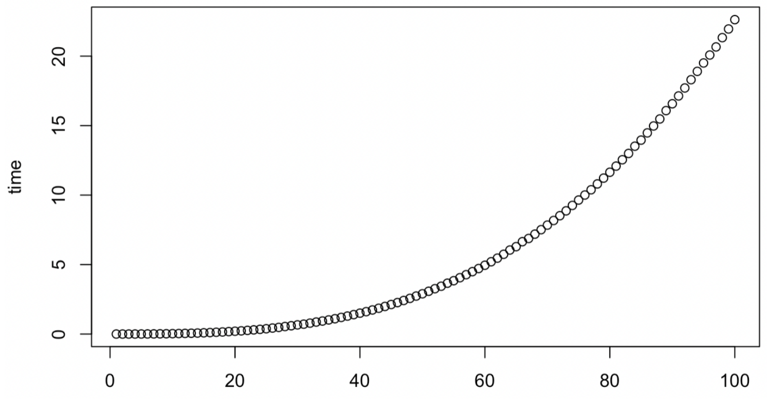


FIGURE 14 – execution time vs length of the sequence (OP)

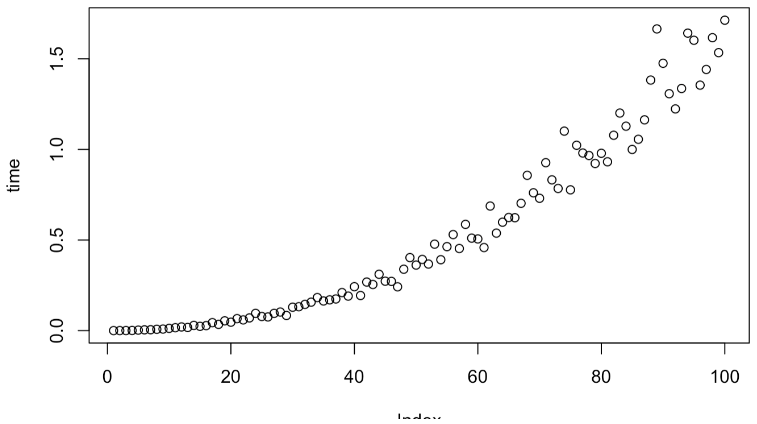
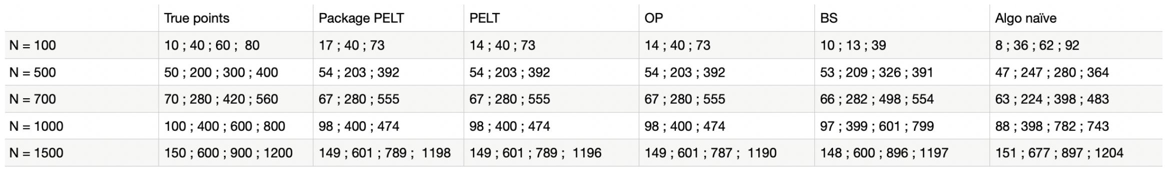


FIGURE 15 – execution time vs length of the sequence (PELT)

5 Remarques

To see how our three algorithms are performing, we tried to run the results on different values $N \in \{100, 500, 700, 1000, 1500\}$.



	True points	Package PELT	PELT	OP	BS	Algo naïve
N = 100	10 ; 40 ; 60 ; 80	17 ; 40 ; 73	14 ; 40 ; 73	14 ; 40 ; 73	10 ; 13 ; 39	8 ; 36 ; 62 ; 92
N = 500	50 ; 200 ; 300 ; 400	54 ; 203 ; 392	54 ; 203 ; 392	54 ; 203 ; 392	53 ; 209 ; 326 ; 391	47 ; 247 ; 280 ; 364
N = 700	70 ; 280 ; 420 ; 560	67 ; 280 ; 555	67 ; 280 ; 555	67 ; 280 ; 555	66 ; 282 ; 498 ; 554	63 ; 224 ; 398 ; 483
N = 1000	100 ; 400 ; 600 ; 800	98 ; 400 ; 474	98 ; 400 ; 474	98 ; 400 ; 474	97 ; 399 ; 601 ; 799	88 ; 398 ; 782 ; 743
N = 1500	150 ; 600 ; 900 ; 1200	149 ; 601 ; 789 ; 1198	149 ; 601 ; 789 ; 1196	149 ; 601 ; 787 ; 1190	148 ; 600 ; 896 ; 1197	151 ; 677 ; 897 ; 1204

FIGURE 16 – Comparaison between the three algorithms and the packages

The advantage of PELT over BS is that PELT is guaranteed to find the optimal segmentation under the chosen cost function, and as such is likely to be preferred providing sufficient computational time is available to run it.But as we can see in the table above, BS still has a good performance regarding some ranges of N and with a specific confidence interval. A way to evaluate the performance of each algorithm, could be a kind of MSE (see this formula below). However, since some points are detected, it is advised to assume that a point is correctly predicted, if a prediction index falls in a window of indexes centered in the true value. If there are 2 predicted value in this window, one of them is considerated as wrong. The size of the window depend of the length of the sequence and should be chosen wisely.

$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{\theta}_i - \theta_i)^2$, with $\hat{\theta}_i$ are the predictions and θ_i are the true changing points.

6 Conclusion

As seen in this report, we studied the complexity and the performance of three algorithms, mainly binary segmentation, OP/PELT and the naïve version, we saw how the optimization of the code using Rcpp helped us reducing the computation time by almost 30 times for BS (with $N = 10000$), 10 times for OP and more than 15 times for PELT (with $N = 1000$). And that is due to the fact that by using a language with less abstraction level, we gain in performance.

All in all, binary segmentation and the naîve approach are non exact methods, based on a boundary decision represented by the statistics F and D_k, but binary segmentation is very efficient, accurate and fast, in the other hand, we studied the exact methods : OP and PELT, but we noticed that their taking more time especially for larger time series data.

Références

[1] Optimal detection of changepoints with a linear computational cost Killick, R., Fearnhead, P. and Eckley, I.A. October 10, 2012

[2] Use of Cumulative Sums of Squares for Retrospective Detection of Changes of Variance Carla INCLAN and George C. TIAO

[3] Application of change-point analysis to the selection of representative data in creep experiments July 2020 N. Volz, S. Neumeier, I. Roslyakova

[4] Strong Approximations in Probability and Statistics, Akadémiai Kiadö, Budapest.

[5] On-line inference for multiple change points problems , Fearnhead, P. and Liu, Z. (2007).