tremors (Popescu and Aiordăchioaie, 2017; Xie et al., 2019); industrial processes monitoring (Pouliezos and Stavrakakis, 2013); detecting adverse health events (Clifford et al., 2015); and monitoring the structural integrity of aeroplanes (Alvarez-Montoya et al., 2020; Basseville et al., 2007).

The online setting raises computational challenges that are not present in offline changepoint detection. A procedure needs to be sequential, in the sense that one should process the observations as they become available, and at each iteration one should make a decision whether to flag a changepoint based on the information to date. The procedure also needs to be able to run on a finite state machine for an indefinite amount of iterations, *i.e.* be constant in memory. Finally, a procedure should ideally be able to process observations, at least on average, as quickly as they arrive. Many online changepoint application settings have high frequency observations, and some also have limited computational resources. For example, the observations from ECG data in the 2015 PhysioNet challenge (Clifford et al., 2015) are sampled at 240Hz, while methods for detecting gamma ray bursts (Fuschino et al., 2019) need to process high-frequency observations and be able to be run on small computers on board micro-satellites. A challenge with online changepoint algorithms is to meet such computational constraints whilst still having close to optimal statistical properties. This paper considers the univariate change in mean problem within precisely this setting.

Current online changepoint methods with a linear computational cost include the method of Page (1955) that assumes knowledge both of the pre-change and post-change mean; or moving window methods such as MOSUM. Assuming the pre-change mean is known is reasonable in many applications as there will be substantial data to estimate this mean (though see discussion in Gösmann et al., 2019). However this approach can lose power if the assumed size of the change is wrong. For example, this method can have almost no power to detect changes that are less than half the size of the assumed change. Similarly, moving window methods can perform poorly if the window size is inappropriate for the size of the change. For example, a small window size will result in little power at detecting small changes, whilst too large a window will lead to delays in detecting larger changes. See Section 2.1 for an example of these issues.

An alternative approach, with more robust statistical properties is to, e.g., apply a moving window but consider all possible window sizes. In the known pre-change mean setting this is known as the Page-cusum approach (Kirch et al., 2018) and is the approach of Yu et al. (2020) for the case of an unknown pre-change mean. The theoretical results in Yu et al. (2020) demonstrate the excellent statistical properties of such a method. However current exact implementations of this idea have a computational cost per iteration that is linear in the number of observations, and thus have an overall quadratic computational cost.

Yu et al. (2020) comment on the challenge of developing faster algorithms with good statistical guarantees: "we are not aware of nor expect to see any theoretically-justified methods with linear order computational costs". This paper presents such an algorithm, which we call Functional Online CuSUM (FOCuS). We develop FOCuS for detecting changes in mean in univariate data, and it can be applied to settings where either the pre-change mean is known or unknown. FOCuS recursively updates a piecewise quadratic, whose maximum is the test statistic for a change. We show that the amortized cost for solving the recursion is constant per iteration. Maximising the function has a computational cost proportional to the number of components in the piecewise quadratic, and the average number of components increases with the logarithm of the number of observations. We can obtain algorithms with a constant computation per iteration by, when necessary, restricting the maximisation to a fixed number of components. We develop one such approximate version of FOCuS and show empirically that it has almost identical statistical performance to an exact implementation. In the unknown pre-change mean case FOCuS implements the method of Yu et al. (2020), and our exact implementation of FOCuS can analyse 1 million observations in less than a second on a common personal computer.

Much research on online changepoint methods has looked at how to implement methods so that they have well characterised performance under the null hypothesis of no change. There are two distinct criteria for quantifying a method's behaviour under the null, one is the average run length (Reynolds, 1975) which is the expected number of observations until we detect a change. The other is a significance level – the probability of ever detecting a change if the method is run on infinitely long data. In practice these two criteria affect the choice of threshold for a detection method. If we wish to control the significance level then we need a threshold that increases with the number of observations (see e.g. Kirch and Kamgaing, 2015), whereas if we wish to control the average run length we can use a fixed threshold. The FOCuS algorithm can be used with either approach – but for simplicity we will only use a fixed threshold in this paper.

The outline of the paper is as follows. In Section 2 we consider the challenge of detecting a univariate change in mean when the pre-change mean is assumed known. We present the FOCuS algorithm which can be viewed as implementing the procedure of Page (1955) simultaneously for all possible size of change. Our main theoretical result shows that FOCuS achieves this with an average computational cost per iteration that is logarithmic in the number of data points. Our bound on the average per iteration cost is tight, and processing the one-millionth observation roughly equates to the cost of evaluating 15 quadratics. In Section 3 we then introduce two extensions of the FOCuS algorithm. The first extension is for the case where the pre-change mean is unknown. This algorithm can be viewed as implementing the statistical tests of Yu et al. (2020). Interestingly, whilst their algorithms are either exact but with a linear cost per iteration, or approximate with a cost that is logarithmic in the number of iterations, the FOCuS algorithm is both exact and has an average cost that is logarithmic per iteration. We do not present any statistical theory for FOCuS, as this is covered in Yu et al. (2020). Second we show how to extend FOCuS to the scenario of detecting changes in the presence of outliers. In Section 4 we show a monitoring application for FOCuS on some AWS Clowdwatch server instances. Finally, the paper concludes with a discussion. Software implementing FOCuS and the code for our simulation study is available at `https://github.com/gtromano/FOCuS`.

2. Known pre-change mean

2.1 Problem Set-up and Background

Consider the problem of detecting a change in mean in univariate data. We will let x_t denote the data at time t, for $t = 1, 2, \ldots$. We are interested in online detection, that is after observing each new data point we wish to decide whether or not to flag that a change has occurred. We first assume that the pre-change mean is known. Often the methods below are implemented in practice using a plug-in estimator for the pre-change mean that is calculated from training data.

A common approach to this problem (see Kirch et al., 2018) is to use a cumulative sum of score statistics, also know as a CUSUM based procedure. Assume we model our data as coming from a parametric model with density $f(x;\mu)$ and denote the pre-change mean as μ_0. Define the score statistic of an observation x as

$$H(x,\mu) = \frac{\partial \log f(x;\mu)}{\partial \mu}.$$

Then if there is no change prior to time n

$$\mathrm{E}\left(H(X_i,\mu_0)\right) = 0, \text{ for } i = 1,\dots,n.$$

Thus evidence of a change prior to time n can be obtained by monitoring the absolute values of partial sums of these score statistics, which we denote as

$$S(s,n) = \sum_{i=s+1}^{n} H(x_i,\mu_0).$$

The idea is that these partial sums should be close to 0 if there is no change, and conversely diverge from zero if there is a change.

For ease of presentation, and to make ideas concrete, in the following we will consider the case where we have a Gaussian model for the data. In this case $H(x,\mu) = (x-\mu)$. Also as we are assuming μ_0 is known, then without loss of generality we can set $\mu_0 = 0$.

There have been a number of different choices of partial sums that we can monitor. For detecting a change after observing x_n, Kirch et al. (2018) highlight the following statistics:

$$\text{CUSUM} \qquad C(n) = \frac{1}{\sqrt{n}}|S(0,n)|; \tag{1}$$

$$\text{MOSUM} \qquad M_w(n) = \frac{1}{\sqrt{w}}|S(n-w,n)|; \tag{2}$$

$$\text{mMOSUM} \qquad M_k^{(\mathrm{m})}(n) = \frac{1}{\sqrt{nk}}|S(n-\lfloor kn \rfloor,n)|; \tag{3}$$

$$\text{Page-CUSUM} \qquad P(n) = \max_{0\le w<n} \frac{1}{\sqrt{w}}|S(n-w,n)|. \tag{4}$$

The scale factor in each case is to normalise the cumulative sum $S(\cdot,\cdot)$, with the aim of standardizing its variance. In each case we would compare the statistic at time n with some appropriate threshold, and detect a change prior to n if the statistic is above the threshold. As discussed in the introduction the choice of threshold impacts the properties of the test under the null. It is possible to choose thresholds that are constant or that increase as n increases (Kirch et al., 2018), but for simplicity we will use constant thresholds throughout.

The standard CUSUM statistic uses the partial sum of score statistics to time n. For both the MOSUM procedure (Eiauer and Hackl (1978), Chu et al. (1995)) and mMOSUM procedure (originating in Chen and Tian, 2010) we need to specify a tuning parameter. The MOSUM method uses the partial sum over a window of the most recent $w > 0$ observations, whilst the mMOSUM fixes some proportion $0 < k < 1$ and uses the partial sum over the most recent proportion k of the observations. All three of these statistics are online, in that there is again only an $O(1)$ update of the statistics as we process each new data point. The Page-CUSUM maximises over all possible partial sums ending at time n.

To illustrate the difference between CUSUM, MOSUM and Page-CUSUM we implement these methods to detect a change at time t to some mean μ_1 for different values of t and μ_1, and compare the detection delay of each statistic. Results are shown in Figure 1. In Figure 1a, we simulated data with a change after 1000 observations and the size of change is chosen to give high power for the window size of the MOSUM procedure. MOSUM and Page-CUSUM tend to detect a change quickly. In the second example, shown in Figure 1b, we reduce the magnitude of the change. Here the MOSUM test loses power substantially and has a much larger detection delay than Page-CUSUM. In our final example, Figure 1c, we have the same size of change as the first example, however the change now occurs after 8,000 observations. In this case we see that the CUSUM statistic behaves poorly. This is because the CUSUM statistic has to average the signal from data after the change with all the data prior to change, and this reduces the power of the test statistic. This is particularly the case when there is substantial data prior to the change. Both MOSUM and Page-CUSUM perform as in the first example. Whilst we do not show the performance of mMOSUM in these examples, it shares a similar sensitivity to the choice of window proportion, k, as the MOSUM does for window size.

The Page-CUSUM approach tries to avoid the issues with choosing a window size within the MOSUM method, and is equivalent to maximising the MOSUM statistic over w. However current implementations of Page-CUSUM are not online. Indeed, fact the computational cost of calculating $\max_{0\leq s<n} \; |S(s,n)|$ increases linearly with n, resulting in an $O(n^2)$ computational complexity.

An alternative approach (Page, 1954, 1955) to detecting the change is based on sequentially applying a likelihood ratio test under an assumed value for the post-change mean, μ_1. Under our Gaussian model with pre-change being 0, we have that the contribution to the likelihood-ratio statistic from a single data point, x_t is

$$LR(x_t, \mu_1) = \mu_1 \left(\frac{\mu_1}{2} - x_t \right)$$

At time n, the test-statistic for a change at time s is the sum of these terms from $t = s+1, \dots, n$. As we do not know the time of the change, we maximise over s:

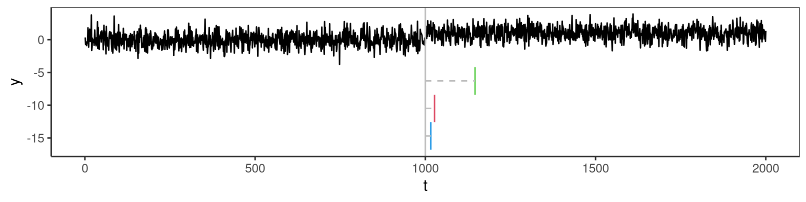
$$\mathcal{Q}_{n,\mu_1} = \max_{0\leq s<n} \sum_{t=s+1}^{n} \mu_1 \left(\frac{\mu_1}{2} - x_t \right).$$

We will call this statistic the *sequential-Page statistic.*

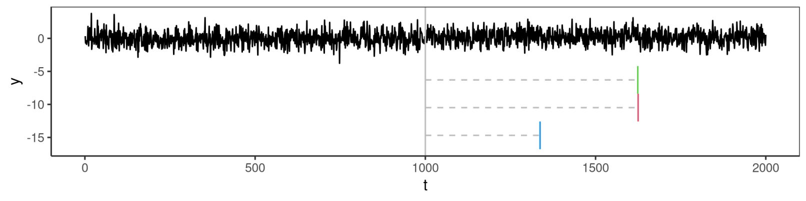
Whilst $\mathcal{Q}_{n,\mu_1}$ involves a sum over n terms, Page (1954) showed that we can calculate $\mathcal{Q}_{n,\mu_1}$ recursively in constant time as:

$$\mathcal{Q}_{n,\mu_1} = \max \left\{ 0,\ \mathcal{Q}_{n-1,\mu_1} + \mu_1 \left(\frac{\mu_1}{2} - x_t \right) \right\}. \tag{5}$$

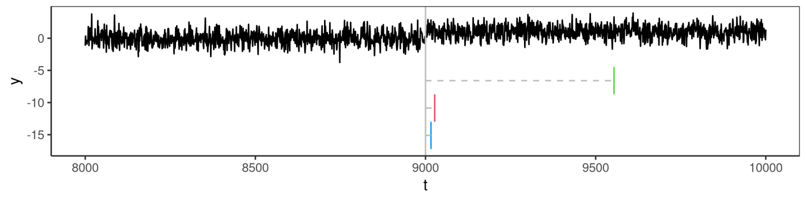
One issue with the sequential-Page statistic is the need to specify μ_1, and a poor choice of μ_1 can substantially reduce the power to detect a change. This is similar to the choice of



(a)



(b)



(c)

Figure 1: Detection delays of CUSUM (in green), MOSUM (in red, with $w = 50$) and Page-CUSUM (in blue) on three sequences. The sequences were generated in the following way: (a) a sequence of 2000 observations with a change of size 1 at 1000; (b) similar to (a) but with a change in the mean of 0.2; (c) similar (a) again, but with an additional 8×10^3 observations at the start of the sequence. Penalties were tuned accordingly to the simulation study in Section 2.3. The solid grey line refers to the true changepoint location, the dashed segments to the detection delays of the super-mentioned procedures.

window size for MOSUM. To partially overcome this, in both cases we can implement the methods multiple times, for a grid of either window sizes or values of μ_1. Obviously, this comes with an increased computational cost.

2.2 FOCuS^0: solving the Page recursion for all μ_1

Our idea is to solve the sequential-Page recursion simultaneously for all values of the post-change mean. To this end we re-write (5) in terms of a recursion for a function $Q_n(\mu)$ of the post-change mean $\mu_1 = \mu$. We then have $Q_0(\mu) = 0$ and for $n = 1, \dots,$

$$Q_n(\mu) = \max \left\{ 0,\ Q_{n-1}(\mu) + \mu \left(x_n - \frac{\mu}{2} \right) \right\}. \qquad (6)$$

We would then use $\max_\mu Q_n(\mu)$ as our test statistic. It is straightforward to see that for any μ_1, $Q_n(\mu_1) = \mathcal{Q}_{n,\mu_1}$. Thus if we can efficiently calculate $Q_n(\mu)$ then our test statistic is equivalent to the maximum value of the sequential-Page statistic over all possible choices of post-change mean. Furthermore, the following proposition shows that this test statistic is

Algorithm 1: FOCuS^0 (one iteration)

```
   Data: $x_n$ the data at time $n$; $Q_{n-1}(\mu)$ the cost function from the previous
         iteration.
   Input: $\lambda > 0$
1  $Q_n(\mu) \leftarrow \max\left\{0,\ Q_{n-1}(\mu) + \mu\left(x_n - \frac{\mu}{2}\right)\right\}$ ;  // Algorithm 2 :  amortized $O(1)$
2  $\mathcal{Q}_n \leftarrow \max_\mu Q_n(\mu)$ ;                        // Theorem 4 :  average $O(\log(n))$
3  if $\mathcal{Q}_n \geq \lambda$ then
4      return $n$ as a stopping point;
5  end
6  return $Q_n(\mu)$ for the next iteration.
```

equivalent to the Page-CUSUM statistic (4), or equivalently the maximum of the MOSUM statistic (2) over all possible windows.

Proposition 1 *The maximum of $Q_n(\mu)$ satisfies*

$$\max_\mu Q_n(\mu) = \frac{1}{2}P(n)^2 = \frac{1}{2}\max_w M_w(n)^2,$$

where $P(n)$ is the Page-CUSUM statistic and $M_w(n)$ is the MOSUM statistic with window size w.

The proof of this can be found in Appendix A.

A description of the resulting algorithm for online changepoint detection is given in Algorithm 1. We call this the Functional Online CuSUM (FOCuS) algorithm. To be able to distinguish this version, that assumes a known pre-change mean, we call Algorithm 1 FOCuS^0. The FOCuS^0 algorithm is only useful if it is computationally efficient, and in particular if we can implement Steps 1 and 2 efficiently. These steps correspond to solving the recursion in (6) to obtain $Q_n(\mu)$ from $Q_{n-1}(\mu)$ and then maximising $Q_n(\mu)$. Below, we describe each of these steps in turn, and present results on their average computational cost.

2.2.1 Step 1: Updating the intervals and quadratics

For Step 1 of Algorithm 1 we propose to update the value of $Q_n(\mu)$ separately for $\mu > 0$ and $\mu < 0$. These can be updated in an identical manner, so we will only describe the update for $\mu > 0$. We will use the fact that (6) maps piecewise quadratics to piecewise quadratics, and hence $Q_n(\mu)$ will be piecewise quadratic (see Maidstone et al., 2017, for a similar idea) and can be stored as a list of ordered intervals of μ together with the coefficients of the quadratic for $Q_n(\mu)$ on that interval. Let $S_t = \sum_{j=1}^t x_j$ be the sum of the first t data points. The quadratic introduced at iteration τ will be of the form

$$\mu\left(\sum_{t=\tau+1}^{n} x_t - (n-\tau)\frac{\mu}{2}\right) = \mu\left((S_n - S_\tau) - (n-\tau)\frac{\mu}{2}\right).$$

Thus, if at time n we know n and S_n, its coefficients can be calculated if we store τ and S_τ. This information stored for the quadratic does not need to be updated at each iteration.

As a result, to update $Q_n(\mu)$ we need only add a quadratic, update the interval associated with currently stored quadratics, and remove quadratics that are not longer optimal for any μ (i.e. whose associated interval is the empty set). Intervals only change due to comparisons between stored quadratics and the new quadratic. The new quadratic is a constant at 0. A key observation, simplifying the functional update, is that the difference between a quadratic introduced at iteration τ and n gives that the new quadratic is better on the interval

$$\left[2 \sum_{t=\tau+1}^{n} \frac{x_t}{(n-\tau)}\ ,\ +\infty\right). \tag{7}$$

Considering all τ we get get that the new quadratic is better than all others on

$$\left[2 \max_{\tau} \sum_{t=\tau+1}^{n} \frac{x_t}{(n-\tau)}\ ,\ +\infty\right).$$

Therefore to update $Q_n(\mu)$ we essentially need to recover

$$\max_{\tau} \sum_{t=\tau+1}^{n} \frac{x_t}{(n-\tau)}. \tag{8}$$

Furthermore this argument shows that the lower bound for any existing quadratics is either unchanged, if it is lower than this value, or that quadratic can be removed.

One can show that the set of changes in $Q_n(\mu)$ are part of the convex hull of the 2D points $\{(1, S_1), (2, S_2), \cdots, (n, S_n)\}$, see Lemma 7 in the Appendix. In fact, our algorithm for updating the list of quadratics is based on Melkman's Algorithm (Melkman, 1987) for calculating the the convex hull of a set of points, and is given in Algorithm 2. A graphical representation of 4 iterations of the procedure is found in Figure 2. As described above, $Q_n(\mu)$ is defined by k quadratics, with associated triples denotes as (τ_i, s_i, l_i) for $i = 1, \ldots, k$. These are ordered so that $0 = l_1 < \cdots < l_k$. The idea is that at time $n+1$ we need to find the value of l such that $Q_n(l) = 0$, by solving (8). We can do this by considering each quadratic in turn, starting with kth quadratic and stepping through them in decreasing order. If we are considering the ith quadratic we check whether $Q_n(l_i) < 0$ or not. If it is we remove the quadratic and move to the $(i-1)$th quadratic (or stop if $i = 1$). If $Q_n(l_i) > 0$ then $l > l_i$ and we find l as the positive value of μ such that the ith quadratic is equal to 0.

We can show that Algorithm 2 has an amortized per-iteration cost that is $O(1)$. The intuition is that each quadratic is added once and removed once, and otherwise unchanged. Thus the average per-iteration cost is essentially the cost of adding and of removing a quadratic.

Theorem 2 *The worst case complexity of Algorithm 2 for any data $x_1, \ldots, x_T$ is $O(T)$ and its amortized complexity per iteration is $O(1)$.*

The proof of this theorem, see below, provides an interesting insight into the behaviour of the pruning step within Melkman's Algorithm.

Proof: At iteration n, let k_n be the number of quadratics input, and let c_n be the number of times the algorithm repeats the while loop in Steps 4 to 6. Let C_1 be the cost

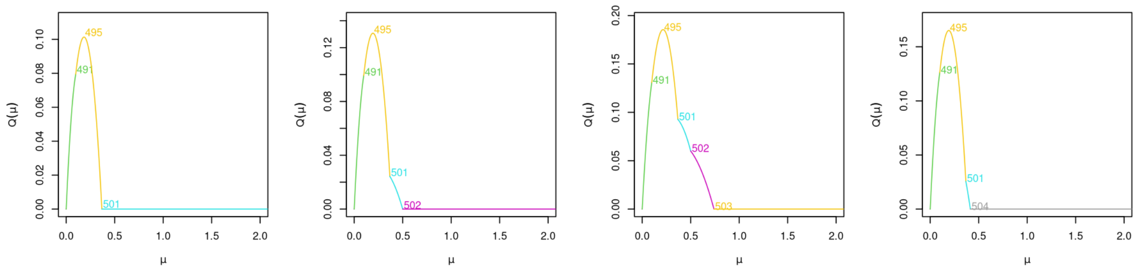


Figure 2: A graphical representation of FOCuS cost function across 4 iterations from time 500 to 504 (left to right). Labels report the iterations where the quadratics were added. The line being the newly added quadratic (hence yet to be updated). We notice how the quadratic introduced at time 502, in purple, is pruned after just two iterations, being no longer optimal.

Algorithm 2: Algorithm for $\max\{0,\ Q_{n-1}(\mu) + \mu(x_n - \mu/2\}$ for $\mu > 0$

```
Data: $Q_n^+(\mu) = \mathrm{Q}$ an ordered set of triples $\{q_i = (\tau_i,\ s_i,\ l_i)\ \forall\ i = 1, \dots,\ k\}$,
      $x_n$ and $S_{n-1}$
1  $S_n \leftarrow S_{n-1} + x_n$ ;                                   // update cumulative sum
2  $q_{k+1} \leftarrow (\tau_{k+1} = n,\ s_{k+1} = S_n,\ l_{k+1} = \infty)$ ;          // new quadratic
3  $i \leftarrow k$;
4  while $2(s_{k+1} - s_i) - (\tau_{k+1} - \tau_i)l_i \leq 0$  and $i \geq 1$ do
5      $i \leftarrow i - 1$;
6  end
7  $l_{k+1} \leftarrow 2(s_{k+1} - s_i)/(\tau_{k+1} - \tau_i)$;                // update new quadratic
8  if $i \neq k$ then
9      $\mathrm{Q} \leftarrow \mathrm{Q} \setminus \{q_{i+1},\ \dots, q_k\}$;                // pruning old quadratic
10 end
11 return $\{\mathrm{Q}, q_{k+1}\}$,   $S_n$
```

of steps 1 to 3 and 7 to 11, and C_2 be the cost of one set of steps 4 to 6. Then the computational cost of one iteration of Algorithm 2 is $C_1 + c_n \times C_2$.

The key observation is that $k_{n+1} = k_n - (c_n - 1) + 1$. That is if we repeat Steps 4 to 6 c_n times then we will remove $c_n - 1$ quadratic in Step 9 and add one quadratic in Step 11. Furthermore $k_1 = 0$ and k_{T+1} is the number of quadratics for $Q_T(\mu)$. Thus the total computational cost is

$$\sum_{n=1}^{T}(C_1 + c_n C_2) = C_1 T + C_2 \sum_{n=1}^{T} c_n = C_1 T + C_2 \sum_{n=1}^{T}(2 + k_n - k_{n+1}).$$

Due to the cancellations in the telescoping sum and the fact that $k_1 = 0$ we have

$$\sum_{n=1}^{T} c_n = 2T - k_{T+1} \leq 2T$$

Thus the theorem holds □

As the overall computational cost is linear in T, the expected cost per iteration must be constant. The proof gives a form for the overhead in terms of the operations in Algorithm 2. In practice this cost is observed to be negligible relative to the cost of step 2 of Algorithm 1, namely that of maximising $Q_n(\mu)$.

2.2.2 Step 2 : Maximisation

To implement Step 2 of Algorithm 1 we first use the trivial observation that if $x_n > 0$ then $Q_n(\mu) < Q_{n-1}(\mu)$ for all $\mu < 0$. Thus to check if $\max_\mu Q_n(\mu) \geq \lambda$ we need only check this for $\mu > 0$. Similarly if $x_n < 0$ then we need only check for $\mu < 0$. To perform the check we just loop over all quadratics stored for either $\mu > 0$ or $\mu < 0$, and for each one check if its maximum is greater than λ. For a quadratic with stored triplet (τ, s, l) this involves checking whether

$$(S_n - s)^2 \geq 2\lambda(n - \tau). \tag{9}$$

If we flag a change at time n, then we can also output the value of τ corresponding to the quadratic whose maximum is largest, and this will be an estimate of the time of the change. The computational cost is thus proportional to number of quadratics that are stored, and can be bounded using the following result.

Theorem 3 *Let $x_1, \dots, x_T, \dots$ be a realization of the process $X_i = \mu_i + \epsilon_i$ where ϵ_i are independent, identically distributed continuous random variables with mean 0. Let the number of quadratics stored by FOCuS^0 for $\mu > 0$ at iteration T be $\#\mathcal{I}^0_{1:T}$. Then if μ_i is constant*

$$E(\#\mathcal{I}^0_{1:T}) \leq (\log(T) + 1),$$

while if μ_i has one change prior to T then

$$E(\#\mathcal{I}^0_{1:T}) \leq 2(\log(T/2) + 1).$$

The proof of this can be found in Appendix B.

By symmetry, the same result holds for the number of quadratics stored for $\mu < 0$. The conditions of the data generating mechanism are weak – as the distribution of the noise can be any continuous distribution providing the noise is independent. The theorem shows that the expected per-iteration time and memory complexity of FOCuS^0 at time T is $O(\log T)$. Furthermore, the expected per-iteration cost is essentially equal to checking (9) $\log(n) + 1$ times if there has not been a change, and for $2(\log(n/2) + 1)$ if there has been an, as yet, undetected change. A change of fixed size is detected in $O(1)$ iterations, and thus the overall computational time, for large T, will be dominated by the cost of iterations prior to the changepoint. For data of size one million, the bound on the number of quadratics is less than 15.

The FOCuS^0 algorithm is not strictly online, due to the cost per iteration not being bounded. But it is simple to introduce a minor approximation that is online. Assume we have a constraint that means we can find the maximum of at most P quadratics per iteration. A simple approximation is to introduce grid of points $\pm m_p$ for $m_p \in \mathbb{R}^+,\ p = 1, ...P$. There are then two natural approaches. One is that if we have $P + 1$ quadratics stored we pruned to P quadratics by removing the first quadratic whose interval does not contain a grid point. Alternatively we can keep all quadratics but only find the maximum of the quadratics whose interval contains a grid point. The advantage of this latter approach is that it avoids any approximations to $Q_n(\mu)$ which could propagate to future values of $Q_t(\mu)$ for $t > n$. Both these methods would dominate using the sequential-Page approach that used the same grid for μ_1 values. For example, if $\tilde{Q}_n(\mu)$ denotes the approximation to $Q_n(\mu)$ using the first approach, then we have $\tilde{Q}_n(\mu_1) = \mathcal{Q}_{n,\mu_1}$ for all μ_1 in our grid.

2.3 Simulation Study

We study the average run length and the detection delay of the FOCuS^0 procedure and its approximation (introduced at the end of Section 2.2), the sequential Page-CUSUM statistics from equation (5), and the MOSUM procedure from equation (2) in case of a pre-change mean known at 0.

The study is structured as follows: For the Page recursion we employ a geometric grid (as recommended by Chen et al., 2020). We use a 10 point grid as that is equivalent to the expected number of intervals stored in FOCuS^0 over a sequence of one hundred thousand observations. To see the potential benefits of using a finer grid, we also use a 20 point grid. We call these two approaches Page-20p, and Page-10p. We evaluate MOSUM over a set of 20 window sizes, with these sizes geometrically increasing, and chosen to give MOSUM greatest power for the size of changes specified by the grid used for Page-20p. We also measure performance of the FOCuS^0 approximation on the 10 points grid used for Page-10p. We call this approximation FOCuS^0-10p.

For each of these methods, we first estimate the run length as a function of the threshold based on data, with no change, of length two million observations. We average the results across 100 different replicates and summarise them in Figure 3. For each method we choose the threshold that gives an average run length of 1×10^6 observations, and evaluate the detection delay with this threshold over a range of different change magnitudes. To generate profiles with changes we superimpose on the previous 100 null profiles a piece-wise constant signal with a change at 1×10^5. For simplicity we only show changes with a positive increase in magnitude, however the study extends to negative changes too.

In Figure 4 we report log-ratios of the average detection delay for pairs of methods against the different change magnitudes. The most striking comparison is between FOCuS^0 and Page-10p. The relative performance of the methods depends on how the size of change matches with grid used for Page-10p. If these match exactly, then Page-10p has a slightly smaller average detection delay because it has a smaller threshold (see Figure 3). However as we move to changes that are different from the grid points, the Page-10p method loses power relative to FOCuS^0 and the latter can be substantially faster at detecting a change. Once we increase the grid size to 20, we have similar qualitative patterns but now the quantitative differences in performance are small, as is the difference between FOCuS^0 and FOCuS^0-10p. Finally, we see that MOSUM gives noticeably the worst performance of all methods.

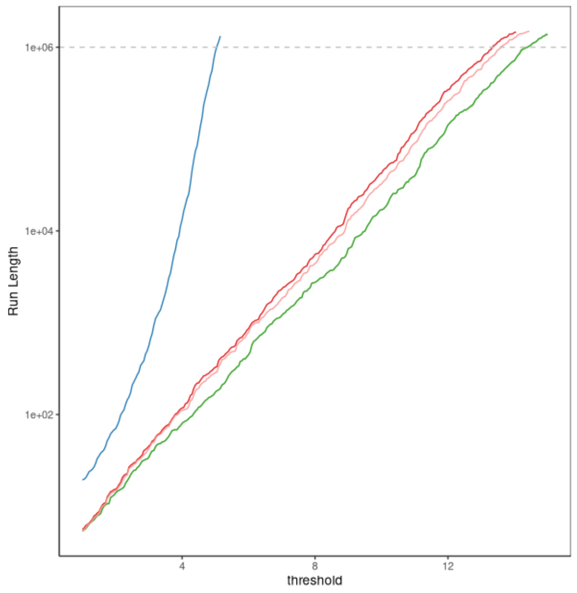


Figure 3: Average Run Length against threshold for FOCuS^0 and FOCuS^0-10p (both identical, and shown in green), Page-20p (pink), Page-10p (red) and finally MOSUM (blue). Results are averaged across 100 simulations. We use a log scale on the y axis.

3. Extensions of FOCuS

3.1 FOCuS when the pre-change mean is unknown

Assume we are observing a sequence of observations $x_1, \ldots, x_n$ distributed as a $N(\mu_0, \sigma)$ under the null and as $N(\mu_1, \sigma)$ under the alternative, with σ known and $\mu_1 \neq \mu_0$. As suggested by Yu et al. (2020), a natural test for a change in the likelihood ratio statistic

$$\mathcal{Q}_n = \max_{\substack{\tau \in \{1,\ldots,n\} \\ \mu_0, \mu_1 \in \mathbb{R}}} \left\{ \sum_{t=1}^{\tau} (x_t - \mu_0)^2 - \sum_{t=\tau+1}^{n} (x_t - \mu_1)^2 \right\}. \tag{10}$$

Yu et al. (2020) present finite-sample results that demonstrate the statistical optimality of such a test. They also present algorithms for evaluating this test statistic. Their fastest algorithm that avoids any approximation is $O(n)$ in computational complexity per iteration while being $O(n)$ in storage, which make their methodology infeasible to a true online setting. For the rest of this paper will refer to such algorithm as Yu-CUSUM.

Let us explain how we can solve (10) using the functional recursion of FOCuS^0. A key observation is that the difference in cost between a change at τ and a change at n for any two means μ_0 and μ_1 can be written as:

$$\sum_{t=1}^{\tau} (x_t - \mu_0)^2 + \sum_{t=\tau+1}^{n} (x_t - \mu_1)^2 - \sum_{t=1}^{n} (x_t - \mu_0)^2 = -(\mu_1 - \mu_0) \left(2 \sum_{\tau+1}^{n} x_t - \mu_0 - \mu_1 \right) \tag{11}$$

The right hand size can be rewritten as $-\delta(2\sum_{\tau+1}^{n} x_t - m)$, with $\delta = \mu_1 - \mu_0$ and $m = \mu_0 + \mu_1$. As before we can consider separately the case $\delta > 0$ (up-change) and $\delta < 0$ (down-change).

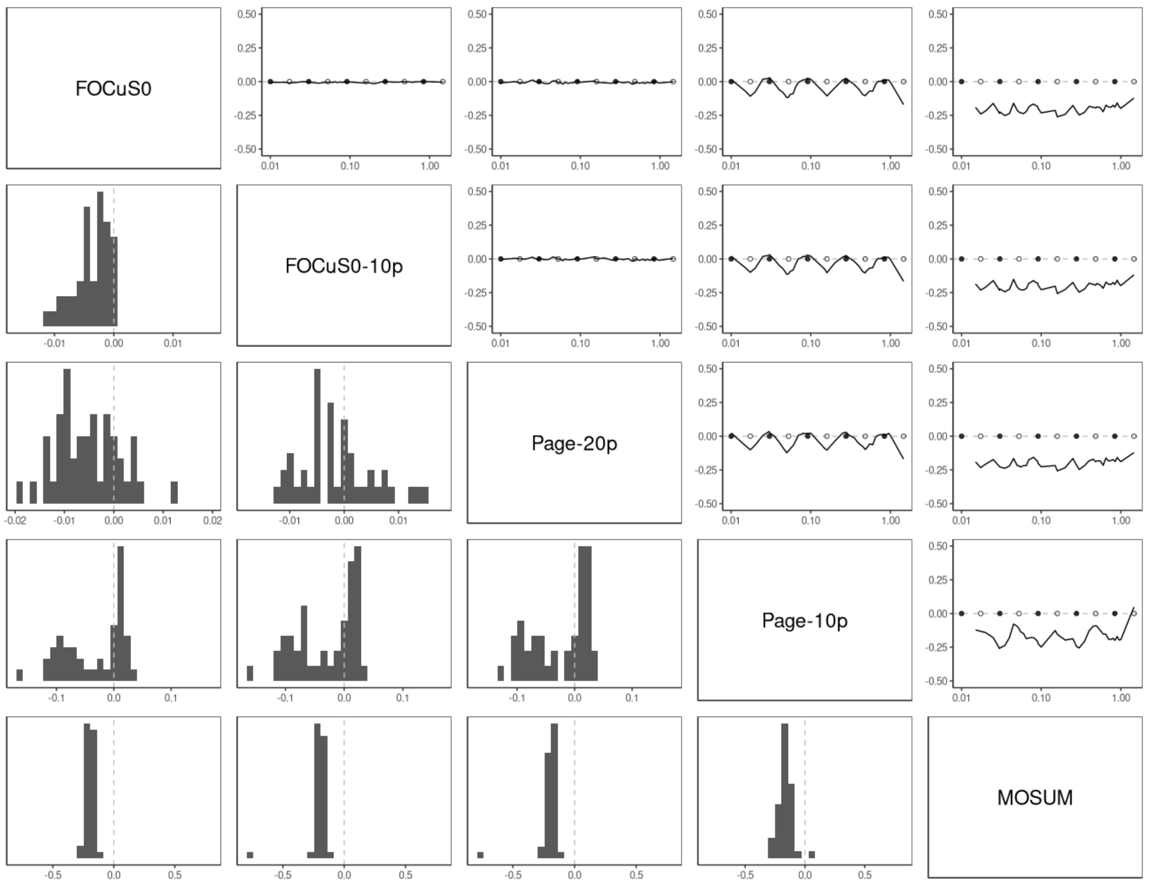


Figure 4: Plot matrix for log-ratios of the average detection delays of each method considered in the study. The diagonal indicates the name of the i-th tested method. The (i,j)-th plot shows the log-ratios of the average detection delays of methods i and j, with the highest index method in the numerator and the lower in the denominator, so values below zero mean the lower index method has a lower detection delay. For example, the (1, 2) and (2, 1) plots give the log-ratios between FOCuS^0 against FOCuS^0-10p, and a value below 0 would mean FOCuS^0 has the smaller average detection delay. The plots of the upper diagonal show the log-ratio as a function of the change magnitude; the dots indicate the 20 points grid, with the filled ones being the one in common with the 10 points grid. The plots on the lower diagonal show histograms of all the log-ratios, with the vertical dotted line being at 0.

We will discuss the case $\delta > 0$ only, as the other case follows immediately by symmetry. For $\delta > 0$ the sign of (11) only depends on m, and we recover, as in (7) for the known pre-change mean case, that a change at n is better than a change at τ if m is in the interval :

$$\left[2\sum_{t=\tau+1}^{n}\frac{x_t}{(n-\tau)}\ ,\ +\infty\right). \tag{12}$$

Hence, we can update the intervals and quadratics corresponding to candidate changes exactly as in Step 1 of FOCuS^0 using Algorithm 2.

In practice there are two small differences

1. For the interval update (step 1), in FOCuS^0 for up-changes we could restrict our attention to $\mu_1 \in [\mu_0, +\infty)$ (resp. $(-\infty, \mu_0]$ for down-changes), whereas in FOCuS, as we do not know the value of the first segment mean, we need to consider all cases for m, i.e. $m \in (-\infty, +\infty)$.

2. For the maximisation (step 2) in FOCuS^0 we only need to optimize the value of the last segment, whereas in FOCuS we also need to optimize the pre-change mean.

By the same argument as that of Theorem 2, we have the solving of the recursion has an average cost per iteration that is constant. We derive in Theorem 4 in Appendix B the same bound on the expected number of candidates as for FOCuS^0, showing that the expected per-iteration time and memory complexity of maximising the solution of FOCuS at time T is $O(\log(T))$.

3.2 FOCuS in the presence of outliers

Further extensions of FOCuS are to use different loss functions to the square error loss obtained from a Gaussian log-likelihood. Motivated by the application in Section 4 we will consider a robust loss function, the biweight loss, which enables us to detect changepoints in the presence of outliers (see Fearnhead and Rigaill, 2019). We define this loss as

$$L(x_t, \mu_1) = -\max\left\{\left(\frac{\mu_1}{2} - x_t\right)^2, K\right\}, \tag{13}$$

where K is a user specified threshold. This leads to the following functional recursion:

$$Q_n(\mu) = \max\left\{\sum_{t=1}^{n} L(x_t, 0),\ Q_{n-1}(\mu) + L(x_n, \mu)\right\}. \tag{14}$$

Using ideas described in Section 3.2 of Fearnhead and Rigaill (2019) it is straightforward to implement this recursion for all μ efficiently. For this model we are unable to recover a bound on the expected number of candidate changepoints. However we observed empirically that the cost for iteration T is in $O(\log(T))$ (see Figure 5).

3.3 Simulation Study

In Figure 5 we give a comparison of the runtime between FOCuS^0, FOCuS, the robust implementation introduced in (14) (R-FOCuS) and Algorithm 3 from Yu et al. (2020),

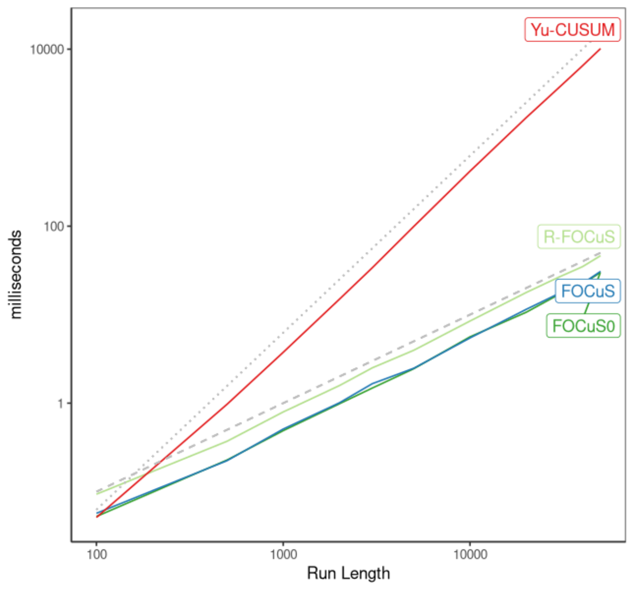


Figure 5: Runtime in milliseconds of FOCuS^0, FOCuS, R-FOCuS and Yu-CUSUM in function of the length of the sequence (log-scale on both axes). Grey lines refer to an expected $O(n)$ increase (dashed) and $O(n^2)$ increase (dotted).

denoted as Yu-CUSUM. Runtimes were recorded for multiple finite sequences of lengths ranging from 100 to 5×10^4. To produce a fair comparison both implementations were written in C++, all simulations were performed on a common personal computer. We find little difference when comparing FOCuS^0 with FOCuS, both showing an empirical linear increase in timings with the latter being slightly slower. When comparing FOCuS to Yu-CUSUM, we find a comparable runtime only up to $n = 100$, after which FOCuS is faster, due to the quadratic computational complexity of Yu-CUSUM. Lastly, we notice how R-FOCUS, while still retaining a linear computational complexity, has a larger overhead compared to the simpler implementations.

The second comparison we make is between FOCuS with the pre-change mean unknown and FOCuS^0 with the pre-change-mean known learned over a training sequence. This is because, as mentioned in the introduction, one could estimate the mean of a Gaussian process when the pre-change mean is unknown, and use such a value to run the algorithms introduced in Section 2. We study in particular the performances of FOCuS^0 as we vary the size of training data from 1000 observations up to 1×10^5. As a benchmark we also compare to FOCuS^0 with known pre-change mean – which should show the best possible performance. We compare both average run-length as a function of the threshold, and detection delay as a function of the magnitude of a change. For each experiment, we report summaries over 100 replicates, and the results on detection delay are for thresholds chosen so each algorithm has an average run-length of 1×10^6. In all cases we simulate data with 1×10^5 data points prior to the change. Results are summarised in Figure 6.

We note how FOCuS requires a smaller threshold to achieve the same average run length of various FOCuS^0 implementations, however differences become negligible when comparing

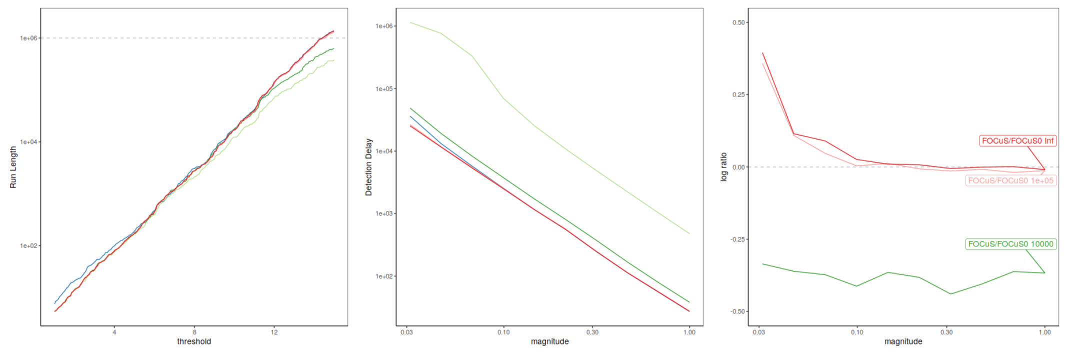


Figure 6: Comparison between FOCuS pre-change unknown and pre-change known: Average run length against threshold (left); average detection delay against magnitude of change (middle); and log-ratio of average detection delay of pairs of methods against magnitude of change (right). For the first two plots: the methods are FOCuS (blue); FOCuS^0 with pre-change mean known (red); and FOCuS^0 with different training data sizes: 1000 (light green), 1×10^4 (dark green) and 1×10^5 (pink).

the methods over larger training sizes. Concerning detection delay, the advantage of FOCuS is that it can improve its estimate of the pre-change mean using the data prior to any change, thus we see substantial benefits of FOCuS relative to FOCuS^0 when the amount of training data is small. However, when the amount of training data is of the same order as the amount of data prior to the change, FOCuS^0 has more power than FOCuS for detecting very small changes, whereas FOCuS has more slightly more power for detecting larger changes. The fact that FOCuS is worse than FOCuS^0 with a large training dataset for small magnitudes is intuitively expected. On the one hand, the cost optimized by FOCuS is bounded making small changes undetectable (the cost will always be smaller than the pre-defined threshold); this is the cause for the detection threshold of Yu et al. (2020). On the other hand, the cost of FOCuS^0 is unbounded, therefore assuming we have a reasonably close estimate of the pre-change mean small changes can be identified.

4. Application of FOCuS to the AWS Cloudwatch CPU utilization

We now evaluate FOCuS by comparing with a bespoke anomaly detection algorithm on the Amazon CPU utilization datasets from the the Numenta Anomaly Benchmark (Ahmad et al., 2017). The aim with these datasets is to detect anomalous behaviours in the CPU utilization of various Amazon Cloudwatch instances. For each dataset anomalous behaviours have been manually flagged by experts, and those stand as the ground truth. The data sets are shown in Figure 7, and demonstrate a range of behaviour. As point anomalies are common we will use the R-FOCuS algorithm.

When evaluating algorithms we will follow the methodology in Ahmad et al. (2017). A detection is deemed to be correct if it lies within $\pm 0.05 \cdot n$ of the true anomaly, where n is the length of the time series; and multiple detection within the window are allowed. A method can use the first 15% of each dataset, a portion of data known to not include any

Detector	Precision	Recall
R-FOCuS	0.58	0.82
Numenta HTM	0.50	0.76

Table 1: Precision and Recall for R-FOCuSand Numenta HTM.

anomalies, to set tuning parameters. We use this data to tune both K in the biweight loss and the detection threshold as described in Appendix D.

As some data sets have multiple anomalies to be detected, we have to adapt R-FOCuS so that it does not stop once a change to some anomalous behaviour has occurred. To adapt R-FOCuS we simply initiate the procedure again at the estimated changepoint location after a detection is triggered. In order to reduce the number of false positives and to extend the average run length of the algorithm, at each detection we inflate the penalty by a factor of $\log(\tau_s)/\log(\tau_s - \tau_{s-1})$, with $\tau_0, ..., \tau_k$ being a vector of estimated changepoint locations.

We compare R-FOCuS with numenta HTM, the best performing algorithm to date on these data. Numenta HTM (Ahmad et al., 2017) is an anomaly detection algorithm that employs an unsupervised neural network model to work with temporal data (Cui et al., 2016) to perform anomaly detection.

Results are summarised in Figure 7 and Table 1. We find that R-FOCuS has better performances in term of Precision, the proportion of true anomalies detected, and Recall, the proportion of detections that are true anomalies, compared with Numenta HTM. On a case to case basis, in most of the sequences both algorithms flagged correctly the anomalies. HTM overall achieves slightly shorter detection delays (with the exception of **f**), however it produces more false positives (13 false detections against 7 of R-FOCuS). In terms of missed detections, both algorithms perform similarly, with R-FOCuS missing an anomaly flagged by HTM in **a**, and HTM missing an anomaly flagged by R-FOCuS in **d**.

Overall, this shows the flexibility of R-FOCuS for online change detection, especially considering that R-FOCuS is a simpler approach which is operating under model misspecification, and with significantly shorter computational run-times than Numenta HTM (on 3000 observations R-FOCuS takes roughly 2 milliseconds against the 4 minutes for HTM).

5. Discussion

As shown in Section 3, one of the advantages of the functional pruning recursion lies in the numerous extensions possible when manipulating the cost directly. For instance, conditions on the underlying mean object of inference could be implemented to produce inference constrained to specific change patterns (Hocking et al., 2020b; Runge et al., 2020), or to account for fluctuating signals and autocorrelation in the noise (Romano et al., 2020), or allow for known behaviour of the the mean between changes (Jewell et al., 2020).

The main limitation of the recursions are that they rely on functional pruning ideas that currently only work for functions of a univariate parameter (though see Runge, 2020, for ideas on extending pruning to higher dimensions). However there is still potential for applying versions of FOCuS in situations where multiple parameters may change, for example by separately testing for changes in each parameter and merging this information.

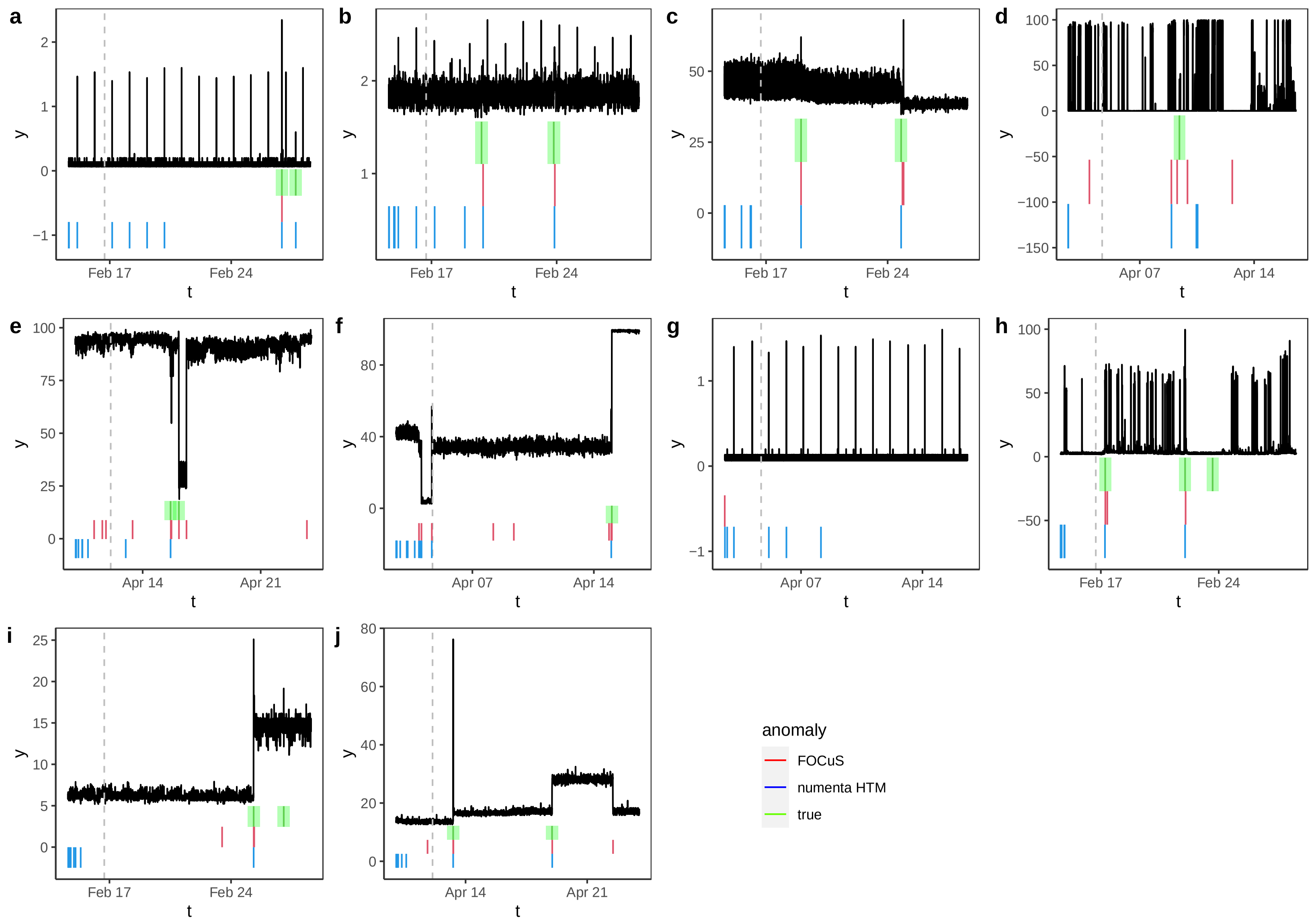


Figure 7: The 8 (**a - j**) different time series of AWS Cloudwatch CPU utilization. Green, red and blue segments correspond, respectively, to the real and estimated anomaly locations of R-FOCuSand Numenta HTM. The green rectangle around each anomaly is the anomaly window (the area in which an anomaly must be detected to count as a true positive). Multiple detections within the green lines are allowed, and are not considered as false positives. The dashed line corresponds to the probation period, used for training of tuning parameters: detections before the dashed line are not accounted for in the final result.

See Chen et al. (2020) for an example of this type of idea for detecting changes in mean in high-dimensional time-series.

In line with Theorem 4, one further possible area of development would be on studying the explicit distribution of the number of quadratics under the alternative, in case the statistics does not reach the threshold. In such case, we expect the number of changes to increase at a faster rate then when under the null: an additional test could be placed on the expected number of quadratics as a fail-safe mechanism to declare a change.

Following the proof of Theorem 4 it is fairly easy to show that the set of candidate changepoints stored by the offline pDPA algorithm Rigaill (2015) run for one change is included in the set of changepoints stored by FOCuS. This provides a bound on the expected complexity of the pDPA for one change. Future work may consider extending the proof of Theorem 4 to get the expected complexity of pDPA for more than one change or of other functional pruning algorithms such as FPOP Maidstone et al. (2017) or GFPOP Hocking et al. (2020a).

References

Joshua Simon Abramson. *Some Minorants and Majorants of Random Walks and Lévy Processes*. PhD thesis, UC Berkeley, 2012.

Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

Joham Alvarez-Montoya, Alejandro Carvajal-Castrillón, and Julián Sierra-Pérez. In-flight and wireless damage detection in a UAV composite wing using fiber optic sensors and strain field pattern recognition. *Mechanical Systems and Signal Processing*, 136:106526, 2020.

Erik Sparre Andersen. On the fluctuations of sums of random variables ii. *Mathematica Scandinavica*, 2:195–223, 1955.

Michele Basseville, Albert Benveniste, Maurice Goursat, and Laurent Meve. In-flight vibration monitoring of aeronautical structures. *IEEE Control Systems Magazine*, 27(5): 27–42, 2007.

Yudong Chen, Tengyao Wang, and Richard J Samworth. High-dimensional, multiscale online changepoint detection. *arXiv preprint arXiv:2003.03668*, 2020.

Zhanshou Chen and Zheng Tian. Modified procedures for change point monitoring in linear models. *Mathematics and Computers in Simulation*, 81(1):62–75, 2010.

Chia-Shang J Chu, Kurt Hornik, and Chung-Ming Kaun. MOSUM tests for parameter constancy. *Biometrika*, 82(3):603–617, 1995.

Gari D Clifford, Ikaro Silva, Benjamin Moody, Qiao Li, Danesh Kella, Abdullah Shahin, Tristan Kooistra, Diane Perry, and Roger G Mark. The PhysioNet/computing in cardiology challenge 2015: reducing false arrhythmia alarms in the ICU. In *2015 Computing in Cardiology Conference (CinC)*, pages 273–276. IEEE, 2015.

Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *Neural Computation*, 28(11):2474–2504, 2016.

Peter Eiauer and Peter Hackl. The use of MOSUMS for quality control. *Technometrics*, 20 (4):431–436, 1978.

Paul Fearnhead and Guillem Rigaill. Changepoint detection in the presence of outliers. *Journal of the American Statistical Association*, 114(525):169–183, 2019. ISSN 0162-1459.

PA Fridman. A method of detecting radio transients. *Monthly Notices of the Royal Astronomical Society*, 409(2):808–820, 2010.

Fabio Fuschino, RICCARDO Campana, CLAUDIO Labanti, Yuri Evangelista, MARCO Feroci, L Burderi, Fabrizio Fiore, Filippo Ambrosino, G Baldazzi, P Bellutti, et al. HERMES: An ultra-wide band X and gamma-ray transient monitor on board a nano-satellite constellation. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 936:199–203, 2019.

Josua Gösmann, Tobias Kley, and Holger Dette. A new approach for open-end sequential change point monitoring. *arXiv preprint arXiv:1906.03225*, 2019.

Toby Hocking, Guillem Rigaill, Paul Fearnhead, and Guillaume Bourque. Constrained dynamic programming and supervised penalty learning algorithms for peak detection in genomic data. *Journal of Machine Learning Research*, 21:1–40, 2020a.

Toby Dylan Hocking, Guillem Rigaill, Paul Fearnhead, and Guillaume Bourque. Constrained dynamic programming and supervised penalty learning algorithms for peak detection in genomic data. *Journal of Machine Learning Research*, 21:1–40, 2020b.

Daniel R Jeske, Nathaniel T Stevens, Alexander G Tartakovsky, and James D Wilson. Statistical methods for network surveillance. *Applied Stochastic Models in Business and Industry*, 34(4):425–445, 2018.

Sean W Jewell, Toby Dylan Hocking, Paul Fearnhead, and Daniela M Witten. Fast nonconvex deconvolution of calcium imaging data. *Biostatistics*, 21(4):709–726, 2020. doi: 10.1093/biostatistics/kxy083.

Claudia Kirch and Joseph Tadjuidje Kamgaing. On the use of estimating functions in monitoring time series for change points. *Journal of Statistical Planning and Inference*, 161:25–49, 2015.

Claudia Kirch, Silke Weber, et al. Modified sequential change point procedures based on estimating functions. *Electronic Journal of Statistics*, 12(1):1579–1613, 2018.

Robert Maidstone, Toby Hocking, Guillem Rigaill, and Paul Fearnhead. On optimal multiple changepoint algorithms for large data. *Statistics and Computing*, 27(2):519–533, Mar 2017.

Avraham A Melkman. On-line construction of the convex hull of a simple polyline. *Information Processing Letters*, 25(1):11–12, 1987.

ES Page. A test for a change in a parameter occurring at an unknown point. *Biometrika*, 42(3/4):523–527, 1955.

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

Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Proactively detecting distributed denial of service attacks using source IP address monitoring. In *International conference on research in networking*, pages 771–782. Springer, 2004.

Theodor D Popescu and Dorel Aiordăchioaie. New procedure for change detection operating on Rényi entropy with application in seismic signals processing. *Circuits, Systems, and Signal Processing*, 36(9):3778–3798, 2017.

AD Pouliezos and George S Stavrakakis. *Real time fault monitoring of industrial processes*, volume 12. Springer Science & Business Media, 2013.

Marion R Reynolds. Approximations to the average run length in cumulative sum control charts. *Technometrics*, 17(1):65–71, 1975.

Guillem Rigaill. A pruned dynamic programming algorithm to recover the best segmentations with 1 to kmax change-points. *Journal de la Societe Francaise de Statistique*, 156 (4):180–205, 2015.

Gaetano Romano, Guillem Rigaill, Vincent Runge, and Paul Fearnhead. Detecting abrupt changes in the presence of local fluctuations and autocorrelated noise. *Journal of the American Statistical Association*, 2020.

Vincent Runge. Is a finite intersection of balls covered by a finite union of balls in Euclidean spaces? *Journal of Optimization Theory and Applications*, 187(2):431–447, 2020.

Vincent Runge, Toby Dylan Hocking, Gaetano Romano, Fatemeh Afghah, Paul Fearnhead, and Guillem Rigaill. gfpop: an R package for univariate graph-constrained change-point detection. *arXiv preprint arXiv:2002.03646*, 2020.

Alexander G Tartakovsky, Aleksey S Polunchenko, and Grigory Sokolov. Efficient computer network anomaly detection by changepoint detection methods. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):4–11, 2012.

Liyan Xie, Yao Xie, and George V Moustakides. Asynchronous multi-sensor change-point detection for seismic tremors. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 787–791. IEEE, 2019.

Yi Yu, Oscar Hernan Madrid Padilla, Daren Wang, and Alessandro Rinaldo. A note on online change point detection. *arXiv preprint arXiv:2006.03283*, 2020.

SUPPLEMENTARY MATERIAL

Appendix A. Proof of Proposition 1

It is straightforward to show, for example by induction, that the solution $Q_n(\mu)$ to recursion (6) can be written in the form

$$Q_n(\mu) = \max_{\tau=1,\dots,n} \left\{ \sum_{i=\tau}^{n} \mu \left(x_i - \frac{\mu}{2} \right) \right\}.$$

Hence

$$\begin{aligned} \max_{\mu} Q_n(\mu) &= \max_{\tau=1,\dots,n} \left\{ \max_{\mu} \sum_{i=\tau}^{n} \mu \left(x_i - \frac{\mu}{2} \right) \right\} \\ &= \max_{\tau=1,\dots,n} \left\{ \frac{1}{2} \sum_{i=\tau}^{n} \left(\frac{\sum_{j=\tau}^{n} x_j}{n-\tau+1} \right)^2 \right\} \\ &= \max_{w=1,\dots,n} \left\{ \frac{1}{2} w \left(\frac{\sum_{j=n-w+1}^{n} x_j}{w} \right)^2 \right\} \end{aligned}$$

The second line uses the fact that the maximum over μ is when μ is the sample mean of $x_{\tau:n}$. The terms in the final expression are just $(1/2)M_w(n)^2$ as required. The result in terms of $P(n)$ follows directly from $P(n) = \max_w M_w(n)$. $\square$

Appendix B. On the expected number of changes stored by FOCuS

B.1 Variants of the FOCuS implementations

We study the number of candidate changepoints $\tau \in \{1, \cdots, n\}$ stored by FOCuS at each iteration. We report the possible FOCuS optimizations introduced in the main body of the paper:

- FOCuS^0 which solves the problem for a known pre-change mean μ_0 (typically 0) and unknown post-change mean μ_1.

$$\mathcal{Q}_n^0 = \max_{\substack{\tau \in \{1,\dots,n\} \\ \mu_0 = 0, \mu_1 \in \mathbb{R}}} \left\{ \sum_{t=1}^{\tau} (x_t - \mu_0)^2 - \sum_{t=\tau+1}^{n} (x_t - \mu_1)^2 \right\}. \quad (15)$$

  This problem is solved through Algorithm 2, an algorithm similar to the Melkman's algorithm (Melkman, 1987).

- FOCuS which solves the problem for both unknown pre-change and post-change means:

$$\mathcal{Q}_n = \max_{\substack{\tau \in \{1,\dots,n\} \\ \mu_1, \mu_0 \in \mathbb{R}}} \left\{ \sum_{t=1}^{\tau} (x_t - \mu_0)^2 - \sum_{t=\tau+1}^{n} (x_t - \mu_1)^2 \right\}. \quad (16)$$

B.2 Assumptions and definitions

In the rest of this section, we let $x_1, \dots x_n$ be our ordered sequence of observations. To denote a subset of such sequence, we will write, from i to j: $x_{i:j} = x_i, \dots, x_j$ for $i < j$. We denote a true changepoint with τ^*.

We assume that:

$$x_i = \mu_i + \varepsilon_i, \tag{17}$$

where ε_i are i.i.d with a continuous distribution and μ_i is a piecewise constant signal in 1 or 2 pieces.

We define the cost of a segmentation $x_{i:j}$ with a change at τ as:

$$q_{i:j,\tau}(\mu_0, \mu_1) = \sum_{t=i}^{\tau} (x_t - \mu_0)^2 + \sum_{t=\tau+1}^{j} (x_t - \mu_1)^2.$$

with pre-change and post-change means μ_0 and μ_1. As a convention, for $j = \tau$, we take: $q_{i:j,j}(\mu_0, \mu_1) = \sum_{t=i}^{j} (x_t - \mu_0)^2$.

Sets of candidate changepoints We call $\mathcal{I}^0_{i:j}$ the set of candidate changepoints stored by FOCuS^0 for $\mu_1 > 0$, and $\mathcal{I}_{i:j}$ the set stored by FOCuS for $\mu_0 < \mu_1$. By definition those will be:

$$\begin{aligned}
\mathcal{I}^0_{i:j} &= \left\{\tau \mid \exists\ 0 < \mu_1,\ \forall \tau' \neq \tau, \quad q_{i:j,\tau}(0, \mu_1) < \quad q_{i:j,\tau'}(0, \mu_1)\right\}, \\
\mathcal{I}_{i:j} &= \left\{\tau \mid \exists\ \mu_0 < \mu_1, \forall \tau' \neq \tau, \quad q_{i:j,\tau}(\mu_0, \mu_1) < \quad q_{i:j,\tau'}(\mu_0, \mu_1)\right\},
\end{aligned}$$

We will first show that $\mathcal{I}^0_{i:j}$ is in $\mathcal{I}_{i:j}$. Our goal will then be to control the size of the set $\mathcal{I}_{i:j}$ of candidate changepoints stored by FOCuS.

B.3 Main results

On the assumption of a realization from (17) we can get the following bound on the number of changepoints stored by FOCuS^0 and FOCuS.

Theorem 4 *For all $n \geq 1$*

$$E(\#\mathcal{I}^0_{1:n}) \leq E(\#\mathcal{I}_{1:n}).$$

For $D = 0$ we have:

$$E(\#\mathcal{I}_{1:n}) = \sum_{1}^{n} 1/t \leq (1 + \log(n))$$

and for $D = 1$ we have

$$E(\#\mathcal{I}_{1:n}) \leq 2(1 + \log(n/2)).$$

Note that by symmetry, the same result holds for the number of quadratics stored by FOCuS^0 for $0 > \mu_1$ and FOCuS for $\mu_0 > \mu_1$.

Overview of the proof The proof to this theorem relies on a combination of three lemmas, summarized here:

1. Lemma 5: $\mathcal{I}_{i:j}$ includes both $\mathcal{I}_{i:j}^{Opt}$ and $\mathcal{I}_{i:j}^{0}$;
2. Lemma 6: for $i \leq j < k$ we have $\mathcal{I}_{i:k} \subseteq \mathcal{I}_{i:j} \cup \mathcal{I}_{j+1:k}$;
3. Lemma 7: $\mathcal{I}_{i:j}$ are the extreme points of the largest convex minorant of the sequence $S_{i:j}$;

and Lemma 8, that controls the number of extreme point of a random-walk (derived from Andersen, 1955; Abramson, 2012). The four lemmas are covered in details and proven in Appendix B.4.

Proof: Using Lemma 5 we obtain the first two inequalities.
For $D = 0$ we apply Lemma 8.
For $D = 1$, using Lemma 6 we get that

$$\mathcal{I}_{1:n} \subseteq \mathcal{I}_{1:\tau^*} \cup \mathcal{I}_{\tau^*+1:n}.$$

We then apply Lemma 8 on $\mathcal{I}_{1:\tau^*}$ and $\mathcal{I}_{\tau^*+1:n}$. The worst case is obtained for $\tau^* = n/2$. $\square$
An empirical evaluation of this bound can be found in Appendix C.1.

B.4 Inclusions and convex hull Lemmas

We begin with the two inclusion lemmas.

Lemma 5

$$\mathcal{I}_{1:n}^{0} \subseteq \mathcal{I}_{1:n} \quad \text{and} \quad \mathcal{I}_{1:n}^{Opt} \subseteq \mathcal{I}_{1:n}$$

Proof: We get the first inclusion by definition of $\mathcal{I}_{1:n}^{0}$ and $\mathcal{I}_{1:n}$.
Consider a change τ in $\mathcal{I}_{1:n}^{Opt}$. Then there exists μ_1 such that for all $\tau' \neq \tau$

$$\min_{\mu_0|\mu_0<\mu_1} q_{1:n,\tau}(\mu_0, \mu_1) \leq \min_{\mu_0|\mu_0<\mu_1} q_{1:n,\tau'}(\mu_0, \mu_1).$$

Defining $\hat{\mu} = \arg\min_{\mu_0|\mu_0<\mu_1} q_{1:n,\tau}(\mu_0, \mu_1)$, we get:

$$q_{i:j,\tau}(\hat{\mu}, \mu_1) = \min_{\mu_0|\mu_0<\mu_1} q_{i:j,\tau}(\mu_0, \mu_1) < \min_{\mu_0|\mu_0<\mu_1} q_{i:j,\tau'}(\mu_0, \mu_1) \leq q_{i:j,\tau'}(\hat{\mu}, \mu_1).$$

Therefore τ is also in $\mathcal{I}_{1:n}$. $\square$

Lemma 6 *For* $i \leq j \leq k$

$$\mathcal{I}_{i:k} \quad \subseteq \quad \mathcal{I}_{i:j} \cup \mathcal{I}_{j+1:k} \tag{18}$$

Proof: Consider any τ in $(i + 1 : j) \cap \mathcal{I}_{i:j}$, then by definition and using equation (19) we get that

$$\exists\ \mu_0 < \mu_1, \forall\ \tau' \neq \tau, \quad q_{i:j,\tau}(\mu_0, \mu_1) < q_{i:j,\tau'}(\mu_0, \mu_1),$$

therefore τ is also in $\mathcal{I}_{i:k}$. We proceed similarly for any τ in $(j + 1 : k) \cap \mathcal{I}_{j+1:k}$. $\square$

A useful identity For any τ, τ', μ_0 and μ_1 we have that

$$q_{i:j,\tau}(\mu_0, \mu_1) - q_{i:j,\tau'}(\mu_0, \mu_1) = (\mu_0 - \mu_1) \left(2 \sum_{\tau+1}^{\tau'} x_t - \mu_0 - \mu_1 \right), \tag{19}$$

which does not depend on i and j. This identity simplifies the proof of the following lemma which relates the set $\mathcal{I}_{i:j}$ to the lower convex hull of $S_{i:j}$.

Lemma 7 *The set of τ in $\mathcal{I}_{i:j}$ are the extreme points of the largest convex minorant of the sequence $S_{i:j}$.*

Proof: Using equation (19), we get that if τ' is in $\mathcal{I}_{i:j}$ it must be that for any τ and τ''

$$\bar{x}_{\tau+1:\tau'} < \bar{x}_{\tau'+1:\tau},$$

with $\bar{x}_{\tau+1:\tau'} = \frac{\sum_{t=\tau+1}^{\tau'} x_t}{\tau' - \tau}$. This is equivalent to for all τ and τ'':

$$\frac{S_{\tau'} - S_\tau}{\tau' - \tau} < \frac{S_{\tau''} - S_{\tau'}}{\tau'' - \tau'},$$

and therefore is part of the largest convex minorant of the sequence $S_{i:j}$.

□

The next lemma is based on Andersen (1955).

Lemma 8 *Assuming the x_t follow an i.i.d continuous distribution on $i : j$ with $\mu_t = \mu_i$ for all t in $i : j$ then $E(\#\mathcal{I}_{i:j}) = \sum_1^{j-i-1} 1/(t+1)$*

Proof: We use Lemma 7 and then apply Andersen (1955) (definitions at pages 1 and 2, then results of pages 23 and 24). □

Appendix C. Additional Empirical Results

C.1 Empirical bound evaluation

To illustrate the bound of Theorem 4 we simulate signals of various length n (from $n = 2^{10}$ to $n = 2^{22}$) without change (5 replicates) and with one change (100 replicates). We then record the number of candidates stored by FOCuS. Where a change was present, its location was sampled uniformly between 1 and $n - 1$. Similarly, the change magnitude was sampled uniformly at random in $[0, 4]$. Results are summarised in figure 8. We denote how the observed number of candidates is always less than $4(\log(n) + 1)$, with the average being around $2(\log(n) + 1)$.

Appendix D. Estimation of initial parameters

We propose a simple sequential estimator for the initial parameters needed to run R-FOCuS (and the other implementations) in an semi-supervised manner. The basic idea to tune the

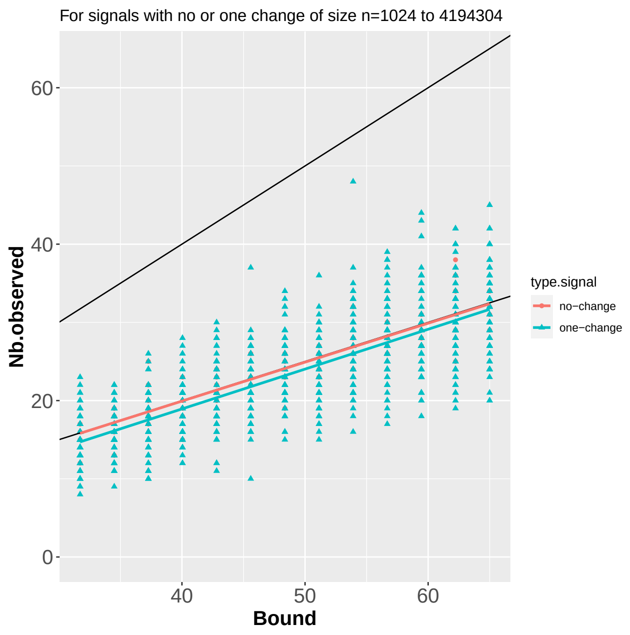


Figure 8: Number of observed candidate stored by FOCuS for signals with no change or with one change. The two black lines represent the function $y = x$ and $y = 0.5x$. Red and blue line are the fitted regression lines for respectively the no-change and single-change scenarios.

λ threshold value is to run the R-FOCuS procedure without a threshold on at most w values, for $w \ll n$, then compute:

$$\lambda^{(w)} = \max\{\max_{\mu} \; Q_i(\mu) \; \forall \; i = 1, 2, \ldots w\}$$

Using such value as a penalty will very naively ensure that the R-FOCuS procedure will run for at least w observations. The next step would be the one of inflating this value $\lambda^{(w)}$ in order to provide a longer average run length to the algorithm: which is we set $\lambda = \kappa\lambda^{(w)}$. On the application within Section 4 simply found that the value of $\kappa = \frac{\log(w+m)}{\log(w)}(1 - \hat{\sigma})$ with $w = 300, \; m = 700$ and $\hat{\sigma}$ being an estimate of the standard deviation of $x_{1:w}$ produced reasonable results.

For tuning the K parameter of the bi-weight loss, one could simply store the initial w values and, if any observation within 1.5 interquantile ranges of those values are present, *i.e.* if we are in presence of outliers, then simply tune the K to be the highest occurring value within this limit. Then, one can simply run FOCuS as described above to tune the λ parameter with the newly found K. In total, this estimation adds a linear in w computational overhead, which consist in temporarily storing the initial w values and performing the estimation of the K.