Simple and effective algorithm for constant state detection in time series.

Andrei Keino

October 1, 2021

Abstract

The article introduces very simple and quite effective algorithm for constant state detection in time series. The algorithm, based on sliding window of variable length, searches a sections of time series with some given minimal length, that have all the values in some given range. It is shown that the computational complexity of aforementioned algorithm is $\mathcal{O}(N \log N)$, where N is the length of time series.

Introduction.

Some technical applications of signal processing requires the steady state detection in time series. This problem has been investigated in many articles. For example, in [wikb] described a computationally efficient method for identification of steady state in time series data. But there are applications for some more straightforward method, described in this article.

Author introduces a very basic and simple algorithm for constant state detection, based on the sliding window with variable length, which have reasonable good computational complexity. The aim of this article is to propose a simple and effective algorithm for identifying the nearly - constant sections in time series.

The algorithm description.

Let we have a time series $y[i]$, $y \in R$, $i = \{1, \ldots, N\}$, where N is the time series size. Let us specify the window length L_W and its height H_W. The proposed algorithm works as follows:

1. Initialize the algorithm structures: create the window associative array (AA) based on self-balancing binary search tree [wika], append to it the key - values pairs, where keys are all the different values of $y[i]$, $i \in \{1, \ldots, L_W + 1\}$, values of AA is the count of repetition of the same values of $y[i]$. Create the arrays for storing indexes of the start position (SP) and the finish positions (FP) of the interval with the steady state of the time series. Store the count of points in AA in a some variable.

2. While algorithm is not finished, repeat:

3. Get the maxima (K_{max}) and minima (K_{min}) key values from AA.

4. If $K_{max} - K_{min} > H_W$:

   (a) If count of points in AA is more than L_W, then move left moving window boundary one point to the left:
   - Remove the value of y[left window boundary] from the AA, if its count is one, in other case decrements its counter.
   - Increment the left window boundary index.
   - Decrement the count of points in AA.

   (b) If count of points in AA is equal to L_W,then move right and left window boundaries one point to the left:
   - Remove the value of y[left window boundary] from the AA, if its count is one, in other case decrements its counter.
   - Increment the left window boundary index.
   - Increment the right window boundary index.
   - Add the value of y[right window boundary + 1] to the AA, if no this value is presented in AA, in other case increment its counter.

5. If $K_{max} - K_{min} \leq H_W$:

   (a) Append left and right moving window boundaries to the SP and FP, if index of the left boundary of moving window is not the same, as the last index in SP. In other case, assign the value of last index in FP to the left moving window boundary index.

   (b) Move the right boundary of the moving window one point to the right:
   - Increment the right window boundary index.
   - Add the value of y[right window boundary + 1] to the AA, if no this value is presented in AA, in other case increment its counter.

The algorithm complexity.

Theorem 1.

Complexity of aforementioned algorithm is $\mathcal{O}(\mathbf{N} \log \mathbf{N})$.

Proof:

Complexity of lookup, deletion, insertion, of pairs in AA is $\mathcal{O}(logM)$, where M is the count of items in AA [wika]. As AA is ordered, the complexity of finding key maxima and minima is $\mathcal{O}(1)$. The complexity of appending the boundaries is $\mathcal{O}(1)$ also. Let $N^{(del)}$ and $C^{(del)}$ are the number of deletions from AA and the count of the operations for one deletion from AA respectfully, $N^{(ins)}$ and $C^{(ins)}$ are the same for operation of insertion in AA, $M_i^{(AA)}$ is the i - th count of keys in AA, $N^{(op)}$ is the overall number of operations.
Then, neglecting the operations with complexity $\mathcal{O}(1)$, we have:

$$\begin{aligned} N^{(op)} &\leq \sum_{i=1}^{N^{(del)}} C^{(del)} \log(M_i^{(AA)}) + \sum_{i=1}^{N^{(ins)}} C^{(ins)} \log(M_i^{(AA)}) \\ &\leq \left(C^{(del)} + C^{(ins)}\right) N \log(N) \end{aligned}$$

The last line of the upper equation means that the complexity of presented algorithm is $\mathcal{O}(N \log N)$.

Some results.

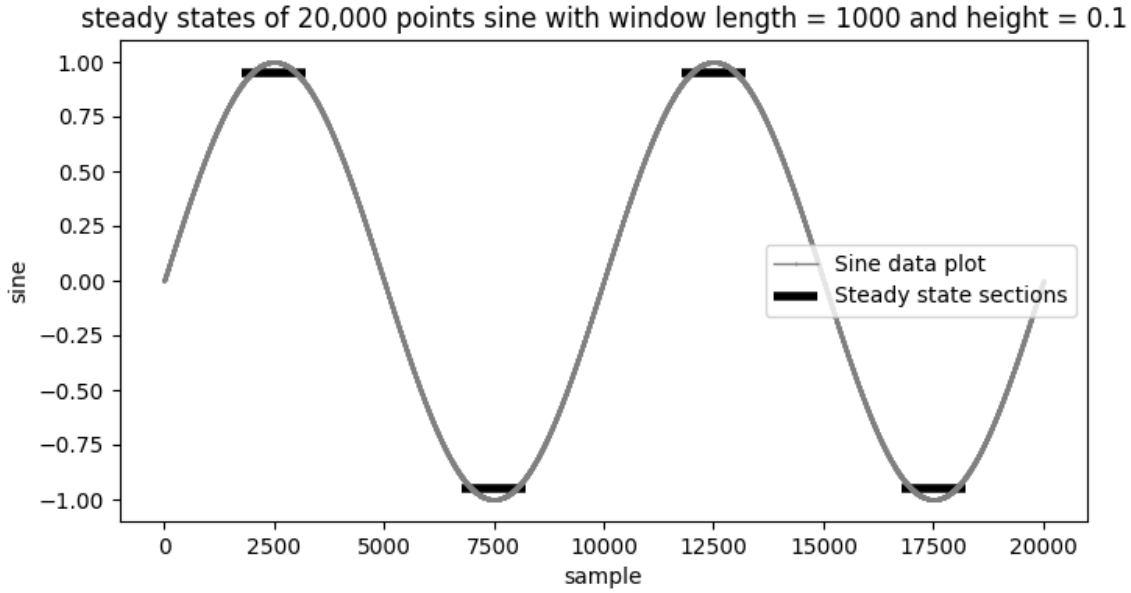

Figure 1: Results of steady state detection for sine time series. Number of sine periods is 2, number of points in time series is 20,000, minimal window length is 1,000, maximal window height is 0.1.

Discussion.

As it's easy to suppose, the algorithm has its drawbacks. For some kinds of time series, the sections found can overlap. This issue can be resolved by post- processing the sections found.

References

[wika] Associative arrays. https://en.wikipedia.org/wiki/Associative_array.

[wikb] A computationally efficient method for identification of steady state in time series data from ship monitoring. https://www.sciencedirect.com/science/article/pii/S2468013320300103.