A Self-Adaptive Spike Detection Algorithm with Application in Performance and Capacity Management

Shivam Sahai, Maitreya Natu, Manoj Jain

Tata Research Development and Design Center
Pune, MH, India, 411013
{shivam.sahai, maitreya.natu, kumar.manoj}@tcs.com

Abstract

This paper presents a simple, yet novel, approach to address the problem of spike detection. The definition of a spike varies with the domain under study. This results in domain-specific solutions that are often rendered unusable in other domains. Moreover, the spike detection algorithms proposed in the past are effective only when suitably tuned. This often limits their usability in real-life scenarios. In this paper, we present a generic approach to address the problem. We propose various spike definitions and present a computationally light-weight algorithm that can cater to all these definitions. The algorithm can automatically adapt itself to changing data characteristics. We demonstrate the soundness of our approach by experimental evaluation. We also compare the proposed algorithm with the existing implementations. We then present an application of spike detection in the domain of performance and capacity management in enterprise systems by presenting real-life case-studies.

1. Introduction

Detection of spikes (peaks and troughs) has been a subject of interest across diverse fields such as mass spectrometry [13], signal processing [11], image processing [3], bio-informatics [2] to name a few. The subject has found such widespread applicability owing to its ability to highlight *deviations*, in an otherwise *routine* scenario.

A close examination of such *deviations* can lead to interesting insights relevant to the applicable domain. Some such examples are shown in Figure 1. Figure 1a shows the price of oil exhibiting *divergence* from routine trend at times of recession in the economy. Figure 1b shows a musical signal that *suddenly intensifies* for a short time at places of high pitch. Figure 1c shows a signal that *slowly intensifies* as the breath is inhaled during respiration process. Figure 1d shows an EEG signal that starts to *fluctuate fiercely* at times of increased brain activity.

The problem of spike detection is challenging due to various reasons. One of the biggest challenges is the *ambiguous* definition of a spike. It is hard to formalize the problem in one single definition. One key reason for the absence of a common definition is the tight coupling of the definition to the applicable domain as shown in Figure 1. In Figure 1a, a spike is a *sudden* rise *or* fall at certain point in time (start *or* end of recession period). It can also be a *gradual pattern* spread across the entire recession period. In Figure 1b a spike is a *sudden* rise followed by a *sudden* fall. The shape of the spike is *symmetric* on both sides and the spike occurs *periodically*. Moreover, the *true* spikes are hidden in the underlying noise. In Figure 1c, a spike is a *gradual* rise followed by a *gradual* fall. The shape of the spike is *symmetric* and the occurrences are *periodic*. However, the fall sustains for relatively longer duration than the rise. The signal here is free of noise. In Figure 1d a spike is a *sudden rise* followed by a *sudden fall*. However, the shape of the spike is *asymmetric* and the spike occurs at *non-periodic* intervals. From these examples it can be seen that it is very difficult to derive a single definition. The existence of such a wide category of applicable domains, each with its own problem definition, has resulted in various spike detection algorithms. Nevertheless, there exist a set of common challenges that restrict the accuracy of the existing spike detection algorithms. Some of these challenges are as follows:

1. **Noise** in the underlying data: Figure 2a shows high level of noise in the data. It is imperative to estimate the right level of noise to avoid detection of many *spurious* spikes (high rate of false positives).
2. Criteria for setting **threshold(s)**: Figure 2c shows a signal with true spikes depicted by numbers 1 to 5. A *solid* threshold (bold line) to detect spikes prevents many *true* spikes (#4 and #5) from getting detected. An *adjustable* threshold (dotted line) does a better job (captures #5 but still leaves #4). A robust

15th International Conference on Management of Data
COMAD 2009, Mysore, India, December 9–12, 2009

criterion for dynamic threshold determination is crucial to achieve high rate of accuracy.

3. Presence of both **strong** and **weak** spikes: Figure 2b shows a data-series where both strong and weak spikes co-exist. In some scenarios, it is necessary to detect both strong as well as weak spikes.
4. **Minimal tuneable parameters**: Existence of wide range of highly unpredictable data characteristics demands different settings of algorithm parameters. It is highly cumbersome and impractical to manually tune the algorithm depending on the situation. Therefore, reliance on minimal tuneable parameters is a foremost necessity.

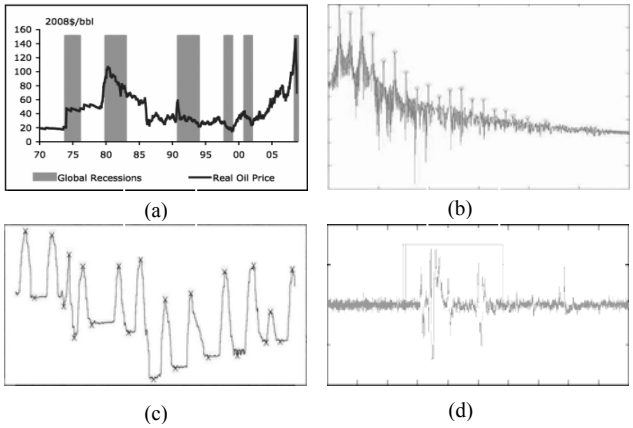


Figure 1 (a) Spikes in price level of oil (b) Spikes in a musical note (c) Spikes in a signal depicting human respiration behaviour (d) Spikes in an EEG signal

In this paper, we present a spike detection algorithm that caters to these challenges. We formalize various spike definitions and show the generic nature of our algorithm across these definitions. We present a technique to estimate the level of noise in the underlying data. We then propose an algorithm that dynamically derives the threshold value based on input data characteristics. The algorithm is capable of detecting both strong as well as weak spikes. Moreover, we rely on minimal tuneable parameters while producing consistently sound results across wide range of data characteristics.

We also present an application of the proposed spike detection algorithm in the area of performance and capacity management in enterprise systems. Many operations in this domain require analysis of various system, workload, and performance metrics. Spike detection proves to be a very useful tool in such analysis. A few relevant instances are listed below:

- **Identification of points of interest**: One of the first steps in the 'as-is' analysis of the system involves identification of *points of interest*. For instance, these points could be specific days in a week when system workload shoots up. Spike detection in such scenarios can identify these points that become the focus for further analysis. We applied spike detection on data gathered from a real-life batch processing system and successfully identified such points of high workload and low performance. We present such a scenario in Section 6.2.
- **Capacity planning**: Spike detection can also be a useful tool in assessing the capacity of a system. For instance, consider a scenario of an online shopping service. Here, an increase in the number of users can cause abnormal spikes in metrics such as available memory, CPU utilization and response time. These spikes can be a useful indicator of the saturation and instability of the system. We present such a scenario in Section 6.3.

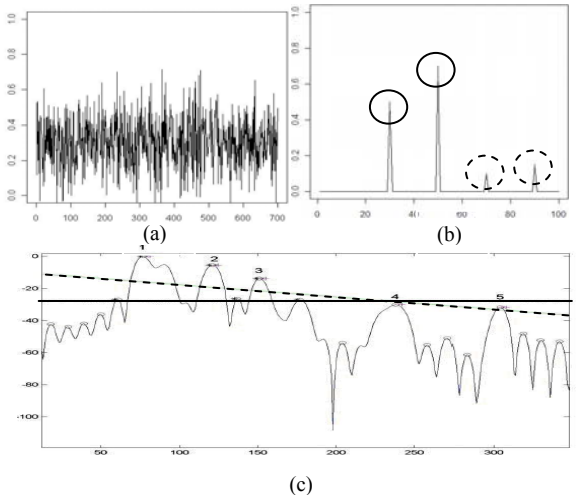


Figure 2 (a) A series with high level of noise with no *true* spikes. (b) A signal with *true* spikes marked by numbers 1 to 5 along with solid and adjustable threshold values. (c) A series where both *strong* (solid circles) and *weak* (dotted circles) spikes co-exist.

The main contributions of this paper are as follows. (1) We propose a spike detection algorithm that is computationally light weight and caters to various spike definitions. (2) We propose a novel approach that automatically tunes the algorithm parameters based on the input data characteristics. Thus, the algorithm shows high accuracy over a wide range of input data. (3) We present an application of the proposed spike detection algorithm in the area of performance and capacity management in enterprise systems. We present experimental evaluation of the proposed algorithm on various real-life case-studies.

The rest of this paper is organized as follows; Section 2 talks about the related work. The proposed approach is discussed in Section 3, followed by the pseudo code in Section 4. Section 5 presents the experimental evaluation of the proposed algorithm. In this section, we validate the effectiveness of our approach and compare the proposed algorithm with spike detection algorithms proposed in the past. In Section 6 we show the application of spike detection algorithm in the domain of performance and capacity management through various real-life case-

studies. Section 7 presents the conclusion. Open issues and future work are discussed in Section 8.

2. Related work

Spike detection has been a subject of interest across diverse domains. Several algorithms have been proposed in context of each applicable domain [1, 2, 3, 5, 7, 8, 9, 11, 13]. Although, all the algorithms cater to the same conceptual problem, the algorithm's design is tightly coupled to the domain. This is evident from the existence of a variety of spike definitions as discussed in Section 1. This coupling restricts their re-use in other domains.

A few mathematical formalizations have been proposed [12], however, they result in large number of false positives unless specifically tailored for the intended domain.

In general, a common approach to address the problem includes *Smoothing* followed by *Baseline correction* and finally *Peak Identification* [13]. *Smoothing* involves capturing the important patterns in the data, while leaving out noise or other fine-scale structures. *Baseline correction* then flattens the baseline of the smoothed data. It attempts to average the baseline to zero. This improves the accuracy in the *Peak Identification* phase which sets a threshold on the baseline corrected data to produce the final result. Each spike detection algorithm incorporates either *all* or a *subset* of these functions in its lifecycle.

Moreover, many spike detection algorithms rely on the usage of a single or multiple threshold values. This notion plays a decisive role in evaluating the effectiveness of any algorithm. In some way or the other, this notion prevails at each phase of spike detection. The amount of desired *Smoothing* might depend on the amount of noise in the underlying data, and so does the amount of *Baseline correction*, which might depend on the nature of overall data. Also, the final criterion that decides whether a point is a spike or not, is always some comparison against a threshold value. It is therefore imperative for each algorithm to incorporate a dynamic criterion for threshold determination while producing consistent results across wide range of data characteristics. In the past, some work has been done in dynamically computing these thresholds [6]. Most of these algorithms work under some assumptions such as absence of outliers in the data [6], reliance on a minimal threshold [4], etc. In addition, some algorithms require additional input parameters, such as window size [10], algorithm-specific parameters (such as minimum momentum as in [4]), for which a constant value might not work in every situation.

Adding to the complexity of an already volatile situation, are unpredictable data characteristics, justifying an indispensable need for an algorithm to be self-tuneable.

3. Proposed approach

Our approach for detection of spikes (peaks and troughs) in a time-series is a sequence of six steps. At each step some data-points are filtered based on certain criteria. The selected data-points are passed onto successive steps to be *rinsed* further. Here we will explain our approach to detect *peaks*. Detection of *troughs* follows a conceptually similar approach, which will be a part of Section 4.

We will use the term *data-point* to refer to the *value* at some time-instant. Also, we use the term *data-series* to refer to a collection of data-points. Below, we explain each of the six steps used in our approach.

3.1 Removal of non-candidate data-points

Intuitively, a spike would *resemble* a data-point as shown in Figure 3a; a data-point (solid) with both *left and right* neighbours (hollow) having a *significant* lower value. In this step, we identify such data points that are candidates for further processing.

For the sake of simplicity, we adhere to certain limitations in this step:

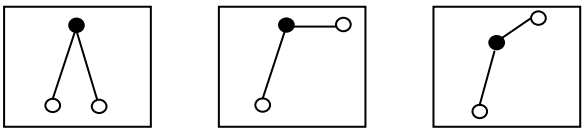
- We consider *only immediate* left and right neighbours.
- The distance from the neighbouring points (*significance*) is not a concern here.

However, this basic *definition* can be extended to include a few more scenarios, as explained using Figure 3.

- A spike can be a data-point where the left neighbour possess a lower value, while the right neighbour can have an *equal* value as shown in Figure 3b.
- A spike can be a data-point where the left neighbour possess a lower value, while the right neighbour can have a *higher* value as shown in Figure 3c.

Such extensions are important since they add more flexibility in matching varying set of expectations. As a whole, we derive four possible *spike definitions* from Figure 3, as listed in Table 1.

Our algorithm, by default, adheres to definition D2. Therefore, as our first step, we consider only those points in the *original* time-series, which adhere to definition D2. By doing this, we are removing all those data-points from the original time-series which can *never* be spikes. This reduces the probability of detecting a false positive, thereby contributing to the accuracy rate. Note that, the algorithm is flexible enough to adapt to the other *spike definitions* (listed in Table 1). This aspect will be highlighted in Section 4.



(a) (b) (c)

Figure 3: Examples of spike shapes

Definition / Figure 3	(a)	(b)	(c)
D1	Included	-	-
D2	Included	Included	-
D3	Included	-	Included
D4	Included	Included	Included

Table 1: Spike definitions

3.2 Bifurcation

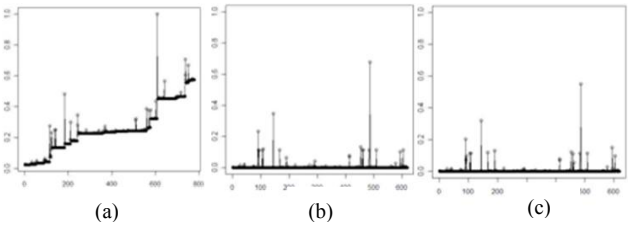
The selected set of data-points is now *bifurcated* into two data-series. The first data-series is composed of the '**difference'** between the data-points with their *immediate left* neighbours. We will refer to this series as *left-data-series*. Similarly, the other data-series is composed of the '**difference'** between the data-points with their *immediate right* neighbours. We will refer to this series as *right-data-series*.

Each data-series is now analyzed *independently*. This simple step gives bountiful of advantages:

- We have achieved *baseline correction* without the use of any explicit baseline correction method. Figure 4a shows the original time-series. Baseline correction is noticeably visible in Figure 4b and Figure 4c.
- The original time-series characteristics (such as *precise* amplitude of spikes, amount of noise) are still *preserved* in the data-series pair. Figure 4b and Figure 4c collectively reflect the original characteristics of the time-series (Figure 4a).
- It provides greater flexibility in deciding the *outcome of a data-point* (spike or not). Consider the encircled data-point in Figure 5a. The data-point is a *candidate* spike as per definition D2. However, the distance of the data-point with its left neighbour is *negligible* when compared to its distance with the right neighbour, as encircled in Figure 5b and Figure 5c respectively. Such *uneven* scenarios become apparent due to independent analysis of the data-series. A flexible criterion can then be applied to decide the *outcome* of such data-points (spike or not). This argument will be supported by a discussion in Section 3.5.

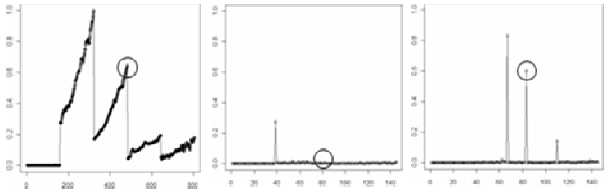
Note that this step can be altered to consider *k* neighbouring points, instead of just the immediate neighbour as per our definition of a spike. Criterion for formation of data-series in such case will be explained in Section 4.

Both the data-series are now independently processed in the next step. We will use the term *data-series* to refer to each series obtained as a result of bifurcation.



(a) (b) (c)

Figure 4: (a) Original time series, (b) Left data-series, (c) Right data-series

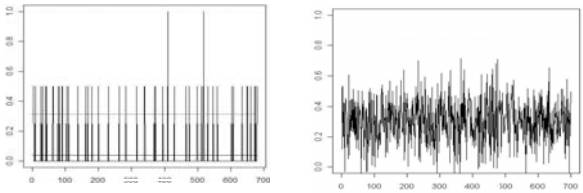


(a) (b) (c)

Figure 5: (a) Original time series, (b) Left data-series, (c) Right data-series

3.3 Noise Estimation

We argue that standard deviation alone is not a sufficient criterion for effective spike detection. Rather, the amount of underlying noise in the data series is an equally important measure. Our approach to detect spikes is based on a robust technique that estimates the right level of noise in a data-series. Figure 6 shows two scenarios with same standard deviation but different noise levels. In Figure 6a, the data-series is almost *steady* with seldom spikes (low noise). However, in Figure 6b, high level of noise is visually noticeable in the data-series.



(a) (b)

Figure 6: (a) Low noise data series, (b) High noise data series

We explain our approach for noise estimation using Figure 7. In Figure 7, we present classification of data-series into four categories on the basis of noise and standard deviation. Figure 7a and Figure 7b are *both* low noise data-series. However, owing to greater deviation from the mean, the standard deviation of Figure 7b is relatively higher. Similarly, data-series in Figure 7c and Figure 7d possess high level of noise. However, greater deviation from the mean in the data-series in Figure 7d increases its standard deviation. In all these figures, a dotted line shows the *mean* value of the data-series.

These data-series show some *peculiar* behaviour as highlighted below:

- In a *low noise* data-series (Figure 7a & Figure 7b), there is *highly uneven distribution* of data-

points about the mean. Figure 8a shows the *density estimate* of such *uneven* distribution. The data-points are mainly concentrated below the mean (dotted line).

- In a *high noise* data-series (Figure 7c & Figure 7d), the distribution of the data-points about the mean is *relatively even*. Figure 8b shows the *density estimate* of an *even* distribution. The data-points are evenly distributed about the mean (dotted line).

We calculate the *density* of data-points (in terms of population) around the mean for the data-series shown in Figure 6. Statistically, for the *low noise* data-series, this population is approximately 93 percent on one side of the mean. This is unlikely the case with the *high noise* data-series where almost 50 percent of data-points evenly fall on *each* side of the mean. On the basis of the above observation, we deduce that the density distribution of data-points about the *mean* is a promising indicator of noise in a data-series (a result of bifurcation process). We argue that low noise in a data series leads to an uneven density distribution and vice-versa. We validate this argument by experimental evaluation in Section 5.1 where we show the effect of increase in noise on the flattening of the density curve. In this section, we use these concepts to compute the noise estimate and the threshold.

3.4 Determination of the threshold value

Our algorithm *auto-computes* the threshold value based on the level of noise in a data-series; the higher the noise level, the higher is the threshold required to avoid detection of *spurious* spikes. We use the threshold to further *trim* the set of candidate spikes by rejecting all the data-points that lie below the threshold value.

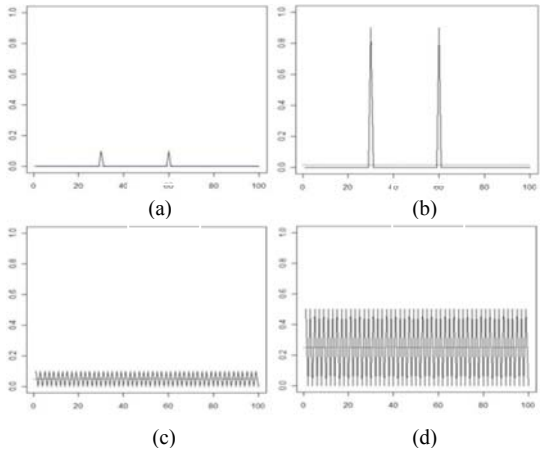


Figure 7: Categorization of data-series on the basis of noise and standard deviation. (a) Low noise and low standard deviation (b) Low noise and high standard deviation (c) High noise and low standard deviation (d) High noise and high standard deviation.

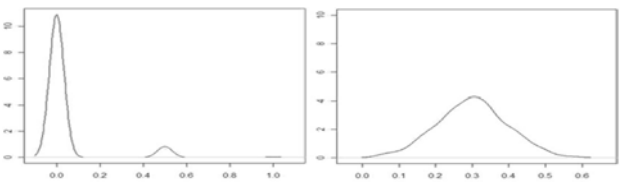


Figure 8: Density estimates of low (a) and high (b) noise data-series of Figure 6

Formally, we define the threshold value (λ) for a data-series with mean (μ) and standard deviation (σ) as:

$$\lambda = \mu + (\beta * \sigma) \qquad (1)$$

where β is the *noise estimate* for the data-series.

As already discussed in Section 3.3, standard deviation alone is not a sufficient criterion for spike detection. Therefore, we compute the noise estimate β based on the technique discussed in Section 3.3, and use β to compute the threshold value λ. We next present a technique to compute β.

In accordance to the noise estimation technique discussed in Section 3.3, we argue that the *density* distribution about the *mean* is a promising indicator of underlying noise. We have supported this argument by calculating the *population* of data-points (percentage) about the mean value for both low and high noise data-series. We will use this notion to derive the *noise estimate* β. Formally, we define the noise estimate β as:

$$\beta = f(\rho) \qquad (2)$$

where ρ is the percentage population on either side of the mean, with the *higher* density estimate.

We observed that a *linear* equation is not the right representative of the relationship between β and ρ. The reason for this inference arose from experimental analysis. We observed that the threshold bar β should be raised *more aggressively* levels below $\rho \sim 80$ to avoid detection of spurious spikes. The β value in the range $80 < \rho < 100$ was found to work well with *lower* increments. Hence, we empirically derived an approximate *polynomial* curve of second degree to represent the relationship between β and ρ. Figure 9 shows the relationship between β and ρ. The *goodness of fit* can be judged by a near one R^2 value of 0.9928. Equation 1 thus governs the *dynamic* value of threshold across varying level of noise in a data-series.

We extended the above concept to introduce the notion of *noise sensitivity*. We define noise sensitivity as the ratio of β to ρ. The higher the ratio, higher is the noise sensitivity. Providing noise sensitivity as a parameter adds more flexibility in customizing the *algorithm's sensitivity to noise*, as per user expectations. In accordance, we deduce three *levels of noise sensitivity*, as shown in Figure10.

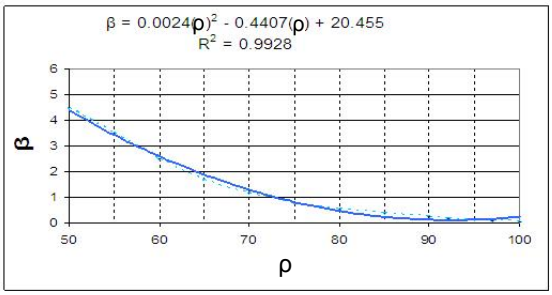


Figure 9: Relationship between β and ρ

The *coefficients* and R^2 values of equation of type $y = ax^2 + bx + c$, for all three levels are listed in Table 2.

Thus, we have derived a novel approach to *dynamically* determine the threshold value depending on the noise estimate of the data-series. In addition, the concept of noise estimate when encapsulated within the notion *of noise sensitivity*, gives more flexibility in customizing the algorithm as per user expectations. We present an experimental evaluation of this concept in Section 5.1.

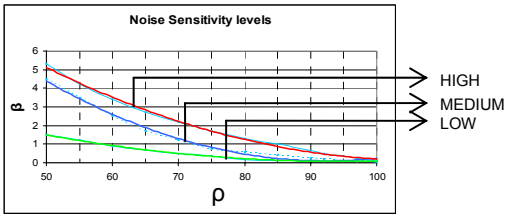


Figure 10: Noise sensitivity levels

Noise Sensitivity	a	b	c	R^2
Low	0.0007	- 0.1369	6.5232	0.998
Medium	0.0024	- 0.4407	20.455	0.993
High	0.0015	- 0.3293	17.732	0.997

Table 2: Coefficients and R^2 values of polynomial equations

3.5 Spike selection criterion

Independent analysis of each data-series in the previous step results in two sets of *candidate* spikes. By default, our algorithm takes the *union* of candidate spikes in both the sets. Thus a data-point is a spike, if individual analysis of *either* of the data-series considers it to be a spike. This is supported by a scenario in Figure 5.

Note that this step is flexible enough to consider the *intersection* of *candidate* spikes in the two sets. This step is useful in deciding the desired *shape* of the spike.

3.6 Comparison with spike intensity

The resultant set of *candidate* spikes undergoes the final step of filtering. This is a simple, yet efficient step that addresses one of the major challenges already discussed in Section 1. The intensities (amplitudes) of the *candidate* spikes are now compared against a *user-desired* intensity value, if any. We calculate the *intensity* of a data-point relative to the maximum value in the original time-series. This gives the flexibility to either accept or reject *weak* spikes. The co-existence of *weak* and *strong* spikes has been a subject of discussion in Section 1. The resulting set of data-points forms the *final* set of spikes

4. Proposed algorithm

In this section, we present the pseudo-code of the proposed spike detection algorithm. We will explain its working against the default spike definition, D2 in Table 1. For the sake of simplicity, the pseudo-code that we present here makes the following assumptions; it will detect *only* the peaks and it considers only the immediate neighbours for analysis.

The generic nature of the algorithm across various aspects such as adapting to other spike definitions, criterion for considering *k* immediate neighbours, and detection of troughs will be a subject of discussion at the end of this section.

Consider a univariate uniformly sampled time-series $T = \langle x_1, x_2, x_3 \ldots x_N \rangle$ of length N. The time-instants are assumed to be 1, 2…N. The value at the i^{th} time-instant is represented by x[i]. We will use symbol N for noise sensitivity and S for spike intensity.

```
procedure spike.detection (T, N, S) do
  variables P, lds, rds
  for each data-point in T, do
    if the data-point matches the spike definition D2 do
      # add the data-point to the left-data-series
      lds = the difference between the data-point and its immediate left
            neighbour
      # add the data-point to the right-data-series
      rds = the difference between the data-point and its immediate
            right neighbour
    end
  end

  if (standard deviation of both lds and rds is zero) do
    return 'No Spike'
  end

  Compute density estimates (ρ) for both lds and rds do
    ρ = density on either side of the mean, whichever is higher
  end

  Compute noise estimate (β) for both lds and rds using Table 2 do
    β = a*ρ² + b*ρ+ c [Equation depends on N]
  end

  Compute the threshold (λ) for both lds and rds do
    λ = μ + (β * σ)
  end

  Select only those data-points from lds and rds that are above the
  threshold value (λ) do
      P = union (data-points above the threshold λ)
  end

  Select only those points from P that exceed the spike intensity value
  (S) do
      P = {P} – {data-points in P with intensity less than S}
  end
end
```

The above pseudo code follows certain assumptions that have already been discussed in this section. We will now

discuss the generic nature of the algorithm across various aspects.

- **Adapting to a different spike definition**: By default, the algorithm follows spike definition D2 of Table 1. We will now explain the approach to *customize* the algorithm for other spike definitions (listed in Table 1). This aspect of generality is a concern in the first step of our approach (Section 3.1). In accordance to Section 3.1, we consider only those data-points in the time-series that follow the spike definition rules. These rules are now formalized in Table 3 for all the four definitions. We will present the interpretation of these rules for one of the spike definition, D3. The same interpretation is applicable to all the other definitions. We will interpret the definition rule (D3) for the selection of candidate peaks (in Section 3.1). The interpretation of the rule is as follows; Any data-point x[i] is considered to be a candidate peak only if, x[i] is higher than its immediate left neighbour (x[i-1]) **and** x[i] is not equal to the immediate right neighbour (x[i+1]). Similarly, rules applicable to trough detection are listed in the third column of Table 3.

Definition	x[i] (Peaks)		x[i] (Troughs)	
	x[i-1]	x[i+1]	x[i-1]	x[i+1]
D1	>	>	<	<
D2	>=	>=	<=	<=
D3	>	!=	<	!=
D4	>=	none	<=	none

Table 3: Spike definition rules for peaks and troughs

Table 3 thus governs the rules that should be followed in order to customize the algorithm for a different spike definition.

- **Considering neighbouring *k* data-points:** In the *bifurcation* phase of our approach (Section 3.2), the algorithm (by default) considers only the immediate left and right neighbours for construction of the data-series (*left-data-series* and *right-data-series*). This aspect however can be extended to consider *k* neighbouring data-points, instead of just the immediate neighbour. We will first explain the approach to construct *left-data-series*. The criteria for selecting the appropriate data-point (among *k* neighbouring data-points on either side of x[i]) differs for both peak and trough detection. For peak detection, we consider the data-point possessing the *minimum* value among the *k* neighbours. Conversely, for trough detection, we consider the data-point possessing the *maximum* value among the *k* neighbours.

  Similar approach applies to the construction of the *right-data-series*. The criterion of selection of the appropriate data-point (among k neighbouring data-points) in this case is listed in Table 4, for both peak and trough detection.

- **Deciding the shape of a spike:** By default, in the spike selection criteria (Section 3.6), the algorithm takes the *union* of both the sets (of candidate spikes) that has resulted from independent analysis of both the data-series. The algorithm can also be customized on this aspect to consider the *intersection* of both the sets. This aspect governs the desired shape of a spike.

Peaks x[i]		Troughs x[i]	
min (x[i-k]:i)	min (i: x[i+k])	max (x[i-k]:i)	max(i: [i+k])

Table 4: Bifurcation rules to construct *left-data-series* and *right-data-series* in case of *k* neighbours.

5. Experimental evaluation

In this section we present the experimental evaluation of the proposed algorithm. We first evaluate the correctness of the proposed approach to detect spikes and the auto-tuning behaviour of the algorithm across changing data characteristics. We validate the proposed auto-tuning approach by running the algorithm with and without auto-tuning support. We show the effectiveness of auto-tuning across a wide range of time-series varying in their level of noise. Section 5.1 covers this aspect of evaluation.

We then compare the proposed algorithm with two widely used spike detection algorithms: (1) entropy-based algorithm [10] (2) a mathematical formalization [10]. We identify the strengths and weaknesses of these algorithms and show that the proposed algorithm outperforms the existing algorithms in various aspects. Section 5.2 presents this evaluation.

5.1 Effectiveness of auto-tuning

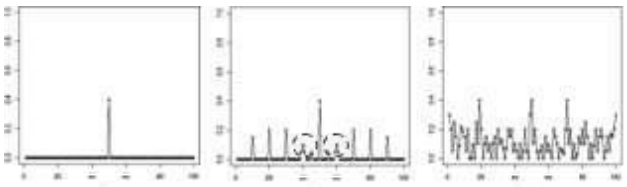
Test-bed description

For evaluating the effectiveness of auto-tuning, we generate artificial noise in a *synthetic* time-series. We started with an almost *steady* time-series of 100 data-points. All the data-points, except one (spike), are at the zero level (Figure 11a). We slowly increment the amount of artificial noise over a span of fifteen time-series (increments). The number of *expected (true)* spikes *increases* for the first seven time-series. However, with increase in noise at further levels, most of them begin to get *buried* in the noise. Thus, the number of expected spikes increases up to the middle and then gradually reduces towards the fifteenth time-series.

In order to achieve comprehensive evaluation of our technique, we plant both *strong* and *weak* spikes. Figure 11b shows the presence of both *weak* (encircled) and *strong* spikes in an intermediate time-series.

In order to give a visual feel of our test-bed, Figure 11a and Figure 11c show the *initial* and *final* state of the synthetic time-series. Time-series in Figure 11a is free of noise, with a prominent spike. An intermediate time-series shown in Figure 11b contains both strong and weak spikes. Figure 11c shows the final state of the time-series. A high level of noise is visually apparent.

Also, to justify our criterion of noise estimation, we present plots of the *density estimates* of four intermediate time-series in order of increasing noise, see Figure 12. The first density plot clearly shows a high density estimate below the mean (a dotted line, almost at zero level), with just a small bump at 0.4 level (corresponding to spike at 0.4 in Figure 11a). With increment in the noise level, this curve starts to *flatten* as apparent in the fourth density plot, for time-series in Figure 11c.



(a) (b) (c)

Figure11: Sample time-series from the test-bed (a) Initial state (b) Intermediate state with weak spikes (c) Final state with noise

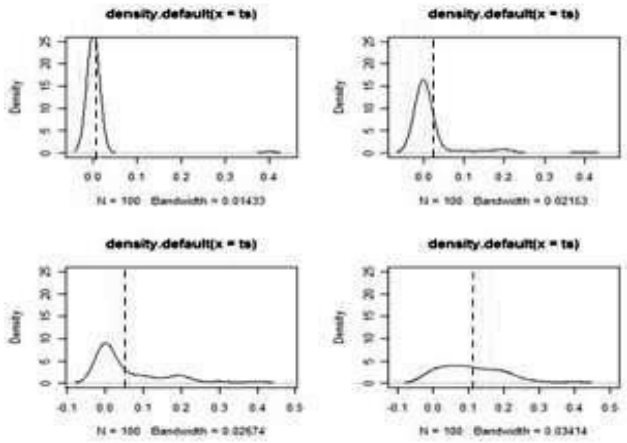


Figure12: Density plots of sample time-series in increasing order of noise.

Parameter tuning

We validate our technique by running the algorithm with and without auto-tuning support. In the *active* mode, the algorithm automatically computes β. The algorithm is tuned to *noise sensitivity* of medium level, which is the default. The *spike intensity* is set to the default value of five percent.

In the *inactive* mode, the noise estimate β is *fixed* at value one.

Evaluation criteria

We evaluate the algorithm on the basis of three simple parameters; detected spikes, false positives and false negatives. We refer to the set of actual spikes as the *expected* set. The *unexpected* set comprises of all data-points other than the *expected* set.

As apparent, *detected spikes* is the number of data-points detected as spikes by the algorithm. *False positives* are the number of data-points detected as spikes, from the *unexpected* set. *False negatives* are the number of data-points not detected from the *expected* set.

Results

Figure 13a shows the *detected spikes* during 'active' and 'inactive' mode, against all fifteen time-series in order of increasing noise. The dotted curve shows the behaviour in *inactive* mode. The solid curve shows the behaviour in the *active* mode.

As apparent from the figure, with the increase in noise level, the *inactive* mode detects more data-points as spikes. However, the active mode senses the increase in the noise level, and the effect of auto-tuning is apparent by the downward motion of the curve (solid) beyond the seventh time-series.

Although this parameter does not anyhow point to the accuracy of the algorithm in either of the modes, it does however show the effect of auto-tuning. The observation that the *active mode* curve remains above the *inactive mode* curve till the seventh time-series produces an interesting insight. Both modes detect the strong spikes in the initial stages. The higher curve of *active* mode shows that, unlike the *inactive* mode, the auto-tuning was effective in detecting even the 'weak spikes' present in the initial stages.

Figure 13b shows the *false positives* produced in both modes. In the *inactive* mode, with increase in noise level the algorithm started to detect *spurious* spikes (dotted curve). However, in the *active* mode, the algorithm kept the *false positives* at a significantly lower level (solid curve). It shows the effectiveness of our approach across varying time-series characteristics.

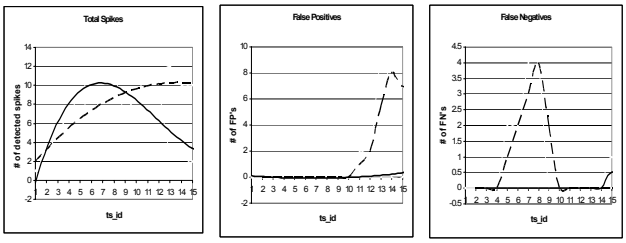
Figure 13c shows the *false negatives* produced in both modes. In the *inactive* mode, the algorithm produces low false negatives for the initial stages having low noise. With further increase in noise, the *inactive* mode fails to capture weak spikes resulting in increase in false negatives. The decrease in false negatives after a point is attributed to the increase in false positives (Figure 13b). With high noise, the algorithm detects both *true* spikes as well as *spurious* spikes resulting in low false negatives but high false positives. In the *active* mode however, the algorithm maintains low false negatives over the entire range (solid curve, Figure 13 c).

The above experiments validate the effectiveness of auto-tuning. The results clearly indicate that auto-tuning correctly captures the time-series properties and provide effective spike detection over a wide range of time series.

5.2 **Comparison with other algorithms**

In this section, we compare the proposed algorithm with two widely used spike detection algorithms; (1) entropy based algorithm [10] (2) a mathematical formalization (mf based) [10]. Both the algorithms are based on the sliding-window concept. The entropy based algorithm first computes the entropy in the window size of *2k* data-points (*k* on each side of the i^{th} data-point). It then

includes the i^{th} data-point and re-computes the entropy now in the window size of *2k+1*. It computes the probability density of the window as an intermediate step while computing the entropy. The difference in the entropy with and without the i^{th} data-point reflects the significance of the i^{th} data-point in the region. On the other hand, the mathematical formalization first computes (i) the average of the distances of i^{th} *data*-point from its *k* left neighbours and (ii) the average of the distances of i^{th} data-point from its *k* right neighbours. Then it takes the average of (i) and (ii). This value reflects the significance of the data-point in the region. The output of both the algorithms then undergoes a common post-processing step to filter out spurious spikes. Moreover, both the algorithms use an additional parameter *h* to adjust the threshold that decides the significance of a spike.



(a) (b) (c)

Figure13: Evaluation results against (a) Total number of detected spikes (b) False Positives (c) False Negatives

Parameter settings: For evaluation of these algorithms, we use the default settings of their arguments. The purpose of this exercise is not to evaluate their behaviour across different set of arguments. Therefore, they are evaluated with the default setting which is expected to produce the best results. We choose window size (*k*) as 15 and *h* as 1.5. The proposed algorithm is also evaluated against its default setting of arguments; noise sensitivity at medium level and spike intensity of five percent.

Test-bed description: The test-bed used for this section comprises of thirteen time-series. In order to achieve a comprehensive review of all the three algorithms across wide range of data characteristics, the time-series differ in various aspects; such as amount of noise, standard deviation, length, presence of weak as well as strong spikes, etc. to name a few. We will evaluate all the three algorithms in their ability to detect only the *peaks*. The data-points that are expected to be spikes are chosen on the basis of their *visual appeal*. This is necessary due to the absence of an optimal solution. In cases where weak spikes were present, they are included in the expected set of spikes.

Comparison criteria: The algorithms are compared on the basis of three metrics; false positives, false negatives and execution time. False positives and false negatives in a time-series are calculated in terms of percentage as follows:

$$\text{False positives} = \frac{(|\{\text{Detected spikes}\} \cap \{\text{Unexpected set}\}|) * 100}{(|\{\text{Unexpected set}\}|)}$$

$$\text{False negatives} = \frac{(|\{\text{Expected set}\}-\{\text{Detected spikes}\}|) * 100}{(|\{\text{Expected set}\}|)}$$

An additional parameter, execution time (seconds) is the time taken by the algorithm to complete its entire process of spike detection. The foremost expectation from these algorithms was to keep minimal false negatives. Detection of almost all the true spikes is of paramount importance. Low false positives are undoubtedly another necessity, but it is not expected to be achieved at the cost of an increase in the false negatives.

Results:

- In terms of false negatives (Figure 14), the proposed algorithm turns out to be a clear winner. It was able to detect all the true spikes. Both *entropy-based* and the *mf-based* were unable to detect many true spikes in a substantial number of time-series (8 out of 13). The set of time-series where these algorithms failed had a common characteristic; they were all low noise time-series with half of them having presence of weak spikes (3, 8, 12 and 13). The proposed algorithm detected all the true spikes in one of the time-series of just 29 data-points, where the window-based algorithms failed.
- Figure 15 shows the false positives produced by the three algorithms. It can be seen that the *entropy-based* algorithm generates smallest number of false positives. However, note that these low false positives come at the cost of high false negatives, as shown in Figure 14. The *mf-based* algorithm is prone to high noise and hence generates high false positives in many cases (7 out of 13). The high false positives account for low false negatives in Figure 14, allowing the algorithm to detect true spikes even in high noise time-series (1, 5). The proposed algorithm produces significantly lower number of false positives than the *mf-based* algorithm. It produces higher false positives than the *entropy-based* algorithm in four cases (1, 4, 6, and 12). In most of these cases the number of false positives is not very high. The data-points detected as false positives had a peculiar behaviour; the significance of fall on the sides of the data-point was *highly uneven*. The proposed algorithm caters to such *shapes* by default, and therefore detected existence of such spikes. Note that, due to its generic nature, the algorithm can be customized to prevent detection of such spikes.
- In terms of execution time (Figure 16), both the proposed algorithm and the *mf-based* algorithm did equally well. However, the *entropy-based* algorithm was found to be computationally expensive. The

reason for this lies in the complexity of its operation. Although the implementation definitely leaves the scope of improvement in terms of optimization, but such a profound difference in the performance is hard to be equalized.

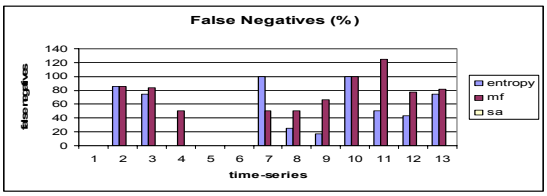


Figure14: False Negatives

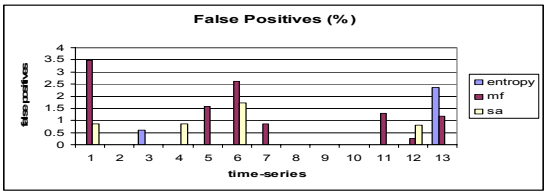


Figure15: False Positives

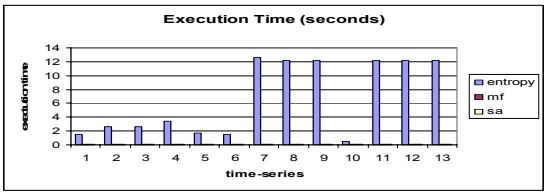


Figure16: Execution Time

Inference: It is evident from the results that the performance of the proposed algorithm was most stable and robust against variety of data characteristics. It achieved the objective of producing less false negatives, while maintaining a less false positives. It performed outstanding in a few cases, especially in cases where weak spikes were present. Owing to the generic nature of the algorithm, a few cases of false positives can be handled easily. It proved to be light-weight owing to its simplicity, a parameter crucial in determining the algorithms practicality in real life scenarios.

6. Application in performance and capacity management domain

In this section, we present various application scenarios to demonstrate the importance of spike detection in the domain of performance and capacity management.

6.1 Background

Today's enterprise systems are expanding in size as well as complexity. Moreover, they continue to host more and more performance critical applications. Such an expansion has made the automated management of these systems a foremost necessity. The current state-of-the-art solutions that attend to these needs rely on deployment of monitors on servers, routers, load balancers, switches, etc. to collect various performance statistics. These statistics are then analyzed for insights into the **as-is** state of the system. It also assists in forecasting the system's performance in the future. Spike detection turns out to be a very useful tool for such analysis. In this section, we present application of spike detection in two such analysis operations: (1) identification of points of interest and (2) capacity planning. We use real-life case studies to demonstrate the effectiveness of the proposed algorithm in capturing interesting insights.

Below, we describe the case-studies that we have used for this purpose:

Trade-plant data: We used the monitored data of the mainframe jobs of a leading investment bank. Various performance metrics are available at a job-level such as the number of requests, CPU used, elapsed time in execution, number of failed executions, etc. The monitors also collect workload metrics (number of requests, arrival time of requests) and performance metrics (elapsed time, throughput). The data was collected for a period of 150 days. We applied spike detection on this data set, in Section 6.2, to identify the critical hours in a day when the components observe high CPU activity.

Server-farm data: This data is obtained by monitoring the server farms of a data centre. The data consists of various system metrics such as total processor time used, available memory, page faults per second, bytes sent and received per second, etc. The data was collected for a month and approximately 1500 data points were collected for each metric every day. In Section 6.2 we apply spike detection to this data to identify interesting regions where various metrics simultaneously show high activity. This analysis provides insights into the steady state of the system.

Petstore data: We also performed a simulation experiment using Load-Runner and Petstore to model a 3-tier web-service architecture. We gradually increased the number of users to observe its effect on various performance metrics such as the elapsed time, CPU utilization, percent free memory, etc. In Section 6.3 we use this data. We demonstrate the applicability of spike detection in providing insights into the conditions of saturation and instability.

6.2 Identification of points of interest

Given the large scale of today's enterprise systems it is absolutely essential to identify the points of interest to narrow down the scope of further detailed analysis. The interestingness can be based on critical time window analysis, critical component analysis, critical path analysis, etc. In this section, we present two such real-life scenarios in which application of spike detection highlights such interesting regions.

We first consider the Trade-plant data for this purpose. We grouped the CPU usage data of 150 days over 5 days * 24 hours space. This helped us in identifying the CPU usage characteristics that are specific to the day of the week and hour of the day. Application of spike detection on this data-set clearly identified the hours with significantly high CPU usage (Figure 17). It can be seen that on all the five days of the week, the system observed significantly high CPU usage on the last few hours of the day. Such spikes in the workload pattern provided insights into better scheduling of the jobs. The jobs that are not bound for execution in this critical region can be scheduled at non-critical time durations for better resource utilization and increased system performance.

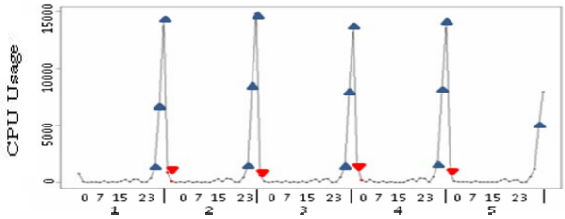


Figure 17: Spikes identified on 2100, 2200, and 2300 hours on all five days of the week.

Next, we present the application of spike detection on the Server-farm data. Here, we analyzed the data collected for a single server on a specific day. We applied spike detection on the single-day time-series of 20 performance metrics. Figure 18 and Figure 19 show the spikes detected on five such metrics namely: total % free disk space, free disk space on drive C, free disk space on drive E, bytes sent from the network interface, bytes received at the network interface. It can be seen that all five metrics observe huge spikes between time-instants corresponding to 500th and 550th data-point. Simultaneous occurrences of spikes in these performance metrics provide valuable insights; (a) they highlight times of high activity in a day; (b) they indicate a change in the steady state of the system.

6.3 Capacity management

Spike detection can also be used to gather insights into various capacity management issues. Though spike detection cannot address complex capacity management problems, it can however provide interesting insights that can be used to carry out further analysis. In this context, we performed an experiment to detect signs of saturation and instability using spike detection. We used the Petstore data in this regard. In this experiment, we gradually increased the number of users and observed its effect on various system parameters.

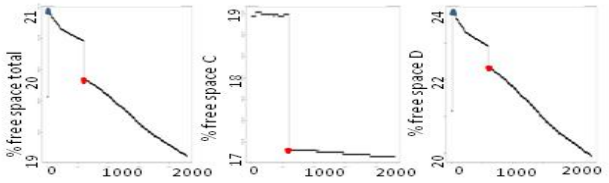


Figure 18: Large spikes observed around 500th point in times series of all three metrics

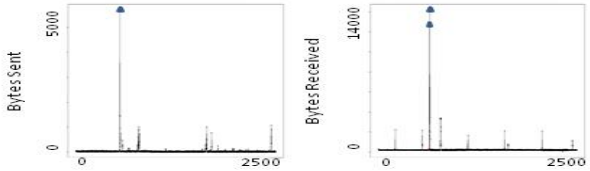
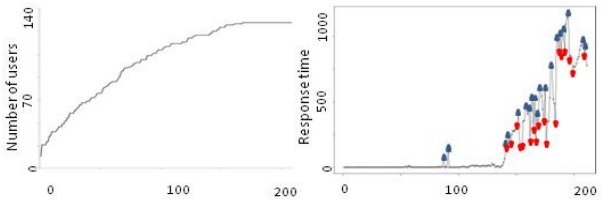


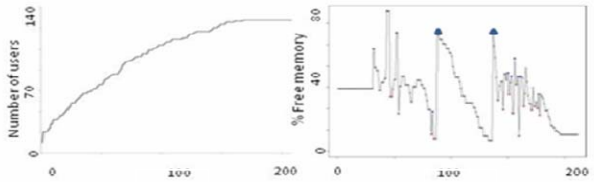
Figure 19: Large spikes observed around 550th point in time series of both metrics

In Figure 20 we show how spike detection can provide symptoms showing the effect of saturation. Figure 20b shows high spikes in the *response time* beyond a certain level of increase in users (Figure 20a). Furthermore the spikes tend to increase with further increase in the number of users. Such symptoms are indicators of saturation of the system. We verified this inference by analyzing the CPU utilization metric which was found to have reached the saturation level.



(a) (b)

Figure 20: Effect of increase in number of users on response time.



(a) (b)

Figure 21: Effect of increase in number of users on % free memory.

Another interesting insight was obtained by running spike detection on the time series of percentage free memory. With the increase in number of users (Figure 21a), the percentage free memory periodically shows a gradual decrease followed by a sudden large spike (Figure 21b). These spikes are indicative of the garbage collection activity. They provide insights into the JVM heap-size

settings. We observed that with an increase in the heap-size, these spikes are detected at larger intervals, indicating the system's capacity to support more users.

7. Conclusion

In this paper, we have proposed a spike detection algorithm that caters to its various inherent challenges. It requires no pre-processing such as explicit baseline correction or smoothing. The need to refrain from such pre-processing methods is evident. Different baseline removal and smooth algorithms can produce different results, are sensitive to parameter settings and overall have a negative affect on the performance of further analysis. The proposed algorithm is computationally light-weight and still maintains high standards of accuracy. Moreover, it is highly flexible in various aspects. It can cater to various spike definitions, offers greater control to decide the *shape* of a spike (due to bifurcation), and offers various levels of noise sensitivity. The algorithm is capable of detecting both strong as well as weak spikes. We have proposed a technique to estimate the right level of noise in the underlying data. The advantage of this technique is harnessed to make the algorithm self-tunable. This has resulted in reliance on minimal tunable parameters, which has been a foremost need against the ever changing data characteristics.

We have also presented an application of the proposed algorithm in the domain of performance and capacity management in enterprise systems. We have shown its importance across various operations relevant to the domain. We have applied the proposed algorithm on various real-life examples to demonstrate the usefulness of spike detection in the studied domain.

8. Open issues & future work

One concern is the applicability of the proposed algorithm on data sets with multiple modes. Consider a bi-modal distribution with one mode (steady) having a significantly lower standard deviation than the other (noisy). The proposed algorithm will set an averaged threshold based on the entire time-series. This can result in unintended spikes getting detected in the noisy node (second part). The advantage is that since the time-series has been steady since the beginning, the detection of spikes in the second half (noisy) does indicate beginning of abnormal activity. However, this issue is rather debatable and is highly dependent on the user expectations. One walk-around this problem is to make the algorithm window-based by choosing a larger size of the window. Another approach could be to split the time series across modes and perform independent analysis the data of these modes. Such scenarios are possible course of action in the future.

References

1. Azzini, R. Dell'Anna, F. Ciocchetta, F. Demichelis, A. Sboner, E. Blanzieri, and A. Malossini, Simple Methods for Peak Detection in Time Series Microarray Data, In Proc. of CAMDA'04 (Critical Assessment of Microarray Data), 2004.
2. P. Du, W. A. Kibbe and S. M. Lin, Improved peak detection in mass spectrum by incorporating continuous wavelet transform based pattern matching, Bioinformatics 2006, vol. 22, pages 2059-2065.
3. R. B. Fisher and D. K. Naidu, A Comparison of Algorithms for Subpixel Peak Detection, Image Technology - Springer-Verlag, Heidelberg, 1996.
4. K. Harmer, G. Howells, W. Sheng, M. Fairhurst and F. Deravi, A Peak-Trough Detection Algorithm Based on Momentum, Congress on Image and Signal Processing, vol. 4, 2008, pages 454-458.
5. P. V. Hese, H. Hallez, B. Vanrumste, Y. D'Asseler and P. Boon, Evaluation of Temporal and Spatial EEG Spike Detection Algorithms, Fifth FTW PhD Symposium, Faculty of Engineering, Ghent University, Dec. 2004.
6. L. Jacobson, Auto-threshold peak detection in physiological signals, In Proc. of the 23rd Annual International Conference of the IEEE, 2001.
7. Kleinberg, Bursty and Hierarchical Structure in Streams, In Proc of 8th ACM SIGKDD, 2002, pages 91–101.
8. M. Ma, A. V. Genderen, and P. Beukelman, Developing and Implementing Peak Detection for Real-Time Image Registration, In Proc. of the 16th Annual Workshop on Circuits, Systems & Signal Processing , ProRISC, 2005.
9. G. M. Nijm, A. V. Sahakian, S. Swiryn, and A. C. Larson, Comparison of Signal Peak Detection Algorithms for Self-Gated Cardiac Cine MRI, Computers in Cardiology, 2007.
10. G.K. Palshikar, Simple Algorithms for Peak Detection in Time-Series, in Proc. of 1st Int. Conf. Advanced Data Analysis, Business Analytics and Intelligence (ICADABAI2009), Ahmedabad, June 2009.
11. H. Satar-boroujeni and B. Shafai, Detection of Peaks in Spectral Representation of Music Signals, Communications in Computing, 2004.
12. B. S. Todd and D. C. Andrews, The Identification of Peaks in Physiological Signals, Computers and Biomedical Research, vol. 32, 1999, pages 322-335.
13. C. Yang, Z. He, and W. Yu, Comparison of public peak detection algorithms for MALDI mass spectrometry data analysis, BMC Bioinformatics, vol. 10(4), 2009.