Knowledge-Based Systems 105 (2016) 248–269

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

ZCR-aided neurocomputing: A study with applications

Rodrigo Capobianco Guido*

Instituto de Biociências, Letras e Ciências Exatas, Unesp - Univ Estadual Paulista (São Paulo State University), Rua Cristóvão Colombo 2265, Jd Nazareth, 15054-000, São José do Rio Preto - SP, Brazil

A R T I C L E I N F O

Article history:
Received 18 March 2016
Revised 1 May 2016
Accepted 7 May 2016
Available online 10 May 2016

Keywords:
Zero-crossing rates (ZCRs)
Pattern recognition and knowledge-based systems (PRKbS)
Feature extraction (FE)
Speech segmentation
Image border extraction
Biomedical signal analysis

A B S T R A C T

This paper covers a particular area of interest in pattern recognition and knowledge-based systems (PRKbS), being intended for both young researchers and academic professionals who are looking for a polished and refined material. Its aim, playing the role of a tutorial that introduces three feature extraction (FE) approaches based on zero-crossing rates (ZCRs), is to offer cutting-edge algorithms in which clarity and creativity are predominant. The theory, smoothly shown and accompanied by numerical examples, innovatively characterises ZCRs as being neurocomputing agents. Source-codes in C/C++ programming language and interesting applications on speech segmentation, image border extraction and biomedical signal analysis complement the text.

1. Introduction

1.1. Objective and tutorial structure

In a previous work, I published a tutorial on signal energy and its applications [1], introducing alternative and innovative digital signal processing (DSP) algorithms designed for feature extraction (FE) [2–4] in pattern recognition and knowledge-based systems (PRKbS) [5,6]. At that time, I intended to cover the lack of novelty in related approaches based on consistency among *creativity*, *simplicity* and *accuracy*. So it is presently, opportunity in which three methods for FE from unidimensional (1D) and bidimensional (2D) data are defined, explained and exemplified, pursuing and taking advantage of my own three previous formulations [1]. The differences between that and this work are related to the concepts and their corresponding physical meanings adopted to substantiate them: antecedently, signal energy was used to provide information on workload, on the other hand, zero-crossing rates (ZCRs) are currently handled to retrieve spectral behaviour [7] of signals. Complementarily, ZCRs are interpreted as being neurocomputing agents, which characterises an innovation that this work offers to the scientific community. Another remarkable contribution consists of the use of ZCRs for 2D signal processing and pattern recognition, a concept practically inexistent up to date.

As in the previous, this essay suggests possible future trends for the PRKbS community. In doing so, it is organised as follows. The concept of ZCRs and some recent related work pertaining to these constitute the next subsections of these introductory notes. Then, Section 2 presents the proposed algorithms for FE, their corresponding implementations in C/C++ programming language [8] and my particular point-of-view which characterises ZCRs as being neurocomputing agents. Moving forward, Section 3 shows numerical examples and Section 4 describes the tests and results obtained during the analyses of both 1D and 2D data. Lastly, Section 5 reports the conclusions that are followed by the references.

Throughout this document, detailed descriptions, graphics, tables and algorithms are abundant, however, for a much better understanding, I strongly encourage you, the reader of this tutorial, to learn my previous text [1] before proceeding any further.

1.2. A review on ZCRs and their applications

Although its roots were traced back before [9] and throughout [10,11] the beginning of DSP, the suitability of ZCRs has been intensively pointed out by the speech processing community, the one in which their applications are more frequent [12]. Thus, ZCRs, as being the simplest existing tools used to extract basic spectral information from time-domain signals without their explicit conversion to the frequency-domain [13], play an important role in DSP and PRKbS.

Despite the word *rate* in its name, ZCR is defined, in its elementary form, as being the number of times a signal waveform

* Corresponding author.
E-mail address: guido@ieee.org
URL: http://www.sjrp.unesp.br/˜guido/

http://dx.doi.org/10.1016/j.knosys.2016.05.011
0950-7051/© 2016 Elsevier B.V. All rights reserved.

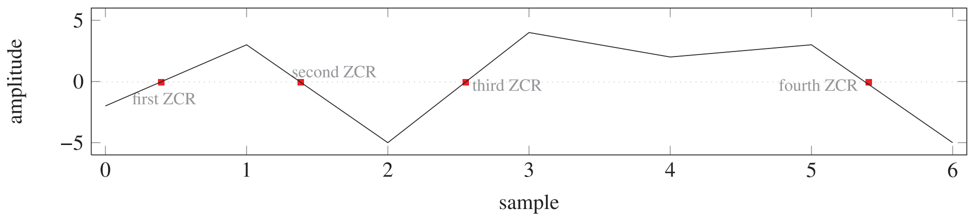


Fig. 1. The example signal $s[\cdot] = \{-2, 3, -5, 4, 2, 3, -5\}$ and its four zero-crossings represented as red square dots. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

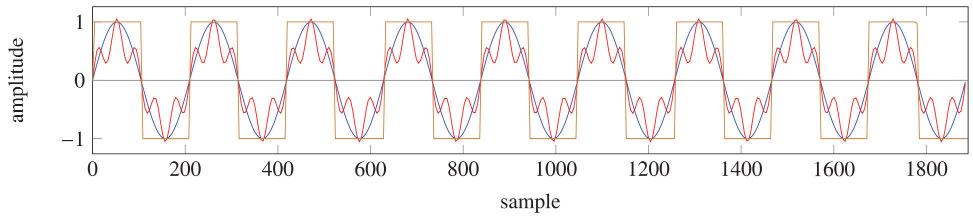


Fig. 2. In blue, the pure sine wave; in red, the composed sine wave; in brown, the square wave. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

crosses the amplitude zero. An alternative and formal manner to express this concept, letting $s[\cdot] = \{s_0, s_1, s_2, ..., s_{M-1}\}$ be a discrete-time signal of length $M > 1$, is

$$ZCR(s[\cdot]) = \frac{1}{2}\sum_{j=0}^{M-2} |sign(s_j) - sign(s_{j+1})|, \tag{1}$$

being $ZCR(s[\cdot]) \geq 0$ for any $s[\cdot]$ and $sign(x) = \begin{cases} 1 & \text{if } x \geqslant 0; \\ -1 & \text{otherwise} \end{cases}$. In the next section, distinct normalisation procedures will be applied to ZCRs in order for the word *rate* to make the intended sense.

As an example, let $s[\cdot]$, of size $M = 7$, be the discrete-time signal for which the samples are $\{-2, 3, -5, 4, 2, 3, -5\}$. Then, $ZCR(s[\cdot]) = \frac{1}{2}\sum_{j=0}^{M-2} |sign(s_j) - sign(s_{j+1})| = \frac{1}{2}\sum_{j=0}^{5} |sign(s_j) - sign(s_{j+1})| = \frac{1}{2}(|-1-1| + |1-(-1)| + |-1-1| + |1-1| + |1-1| + |1-(-1)|) = \frac{1}{2}(|-2| + |2| + |-2| + |0| + |0| + |2|) = \frac{1}{2}(2+2+2+0+0+2) = 4$, i.e., the waveform of $s[\cdot]$ crosses its amplitude axis four times at the value 0, as can be easily seen in Fig. 1.

The elementary example I have just described is really quite simple, however, I ask for your attention in order to figure out the correct physical meaning of ZCRs, avoiding underestimations. For that, a basic input drawn from Fourier's theory and his mathematical series [14] is required: the statement which confirms that any signal waveform distinct of the sinusoidal can be decomposed as an infinite linear combination of sinusoids with multiple frequencies, called *harmonics*. Thus, a signal waveform that matches *exactly* a sinusoidal function, with a certain period, phase and amplitude, is classified as being *pure*. Conversely, any other type of signal waveform consists of a main sinusoid called *fundamental* or *first harmonic*, owning the lowest frequency among the set, added together with the other sinusoids of higher frequencies, i.e., the second harmonic, the third harmonic, the fourth harmonic, and so on, in a descending order of magnitude.

The connection between ZCRs and Fourier's series is now explained on the basis of the following example, illustrated in Fig. 2. In blue, red and brown, respectively, a pure sine wave, a composition of two sine waves and a square wave that is essentially the sum of infinite sinusoids, are shown, all with the same length. Interestingly, the three curves have exactly the same number of ZCRs, however, according to Fourier's theory, their frequency contents are considerably different. Based on the example, the learnt lesson is: the first harmonics of a *non pure* signal are dominant over the others, whilst mandatory to define its general waveform shape. Consequently, it is often the minor oscillations produced by the higher harmonics that do not generate zero-crossings. Therefore, the ZCR of a given signal is much more likely to provide information on its fundamental frequency than a detailed description of its complete frequency content.

Another relevant concept is the direct relationship between the fundamental frequency of a signal and its ZCR. Since sinusoids are periodic in 2π, each period contains two zero-crossings, as shown in Fig. 3. Thus, if a 1D signal $s[\cdot]$ of length M crosses G times the amplitude zero, it contains $\frac{G}{2}$ sinusoidal periods at that frequency. Considering that, at the time the signal was converted from its analog to its digital version [14], the sampling rate was R samples per second, then $\frac{1}{R}$ is the period of time between consecutive samples, entailing that $M \cdot \frac{1}{R} = \frac{M}{R}$ is the time extension of the analog signal in seconds. Concluding, in $\frac{M}{R}$ seconds there are $\frac{G}{2}$ sinusoidal periods, implying that, proportionally, there are $\frac{G \cdot R}{2 \cdot M}$ periods per second, i.e., the frequency, F, caught by the ZCRs is

$$F(ZCR(f[\cdot])) = \frac{G \cdot R}{2 \cdot M} \quad Hz. \tag{2}$$

Obviously, the previous formulation is only valid if the sinusoids are not shifted on the amplitude axis, i.e., no constant value is added to them. Equivalently, the signal under analysis is required to have its arithmetic mean equal zero, implying that an initial adjustment may be necessary prior to counting the ZCRs, otherwise they would not be physically meaningful. The simplest process to normalise a signal $s[\cdot]$ in order to turn its mean to zero is to shift each one of its samples, subtracting its original mean, i.e.,

$$s_k \leftarrow s_k - \frac{\left(\sum_{j=0}^{M-1} s_j\right)}{M}, \qquad (0 \leqslant k \leqslant M-1) \quad . \tag{3}$$

In order to illustrate the concepts I have just exposed, the readers are requested to consider the signal $s[\cdot] = \{\frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}\}$, of length $M = 9$, that was sampled at 36 samples per second and is illustrated in Fig. 4. Its arithmetic mean is $\frac{\frac{12}{10}+3+\frac{12}{10}+3+\frac{12}{10}+3+\frac{12}{10}+3+\frac{12}{10}}{9} = 2 \neq 0$, i.e., the normalisation defined in Eq. (3) must be applied before the ZCRs are counted. Thus, $s[\cdot]$ becomes $\{\frac{12}{10}-2, 3-2, \frac{12}{10}-2, 3-2, \frac{12}{10}-2, 3-2, \frac{12}{10}-2, 3-2, \frac{12}{10}-2\} = \{-\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}\}$, which has its mean equal zero and is also shown in Fig. 4. ZCRs are now ready to be counted, according to Eq. (1), resulting in $G = 8$ zero-

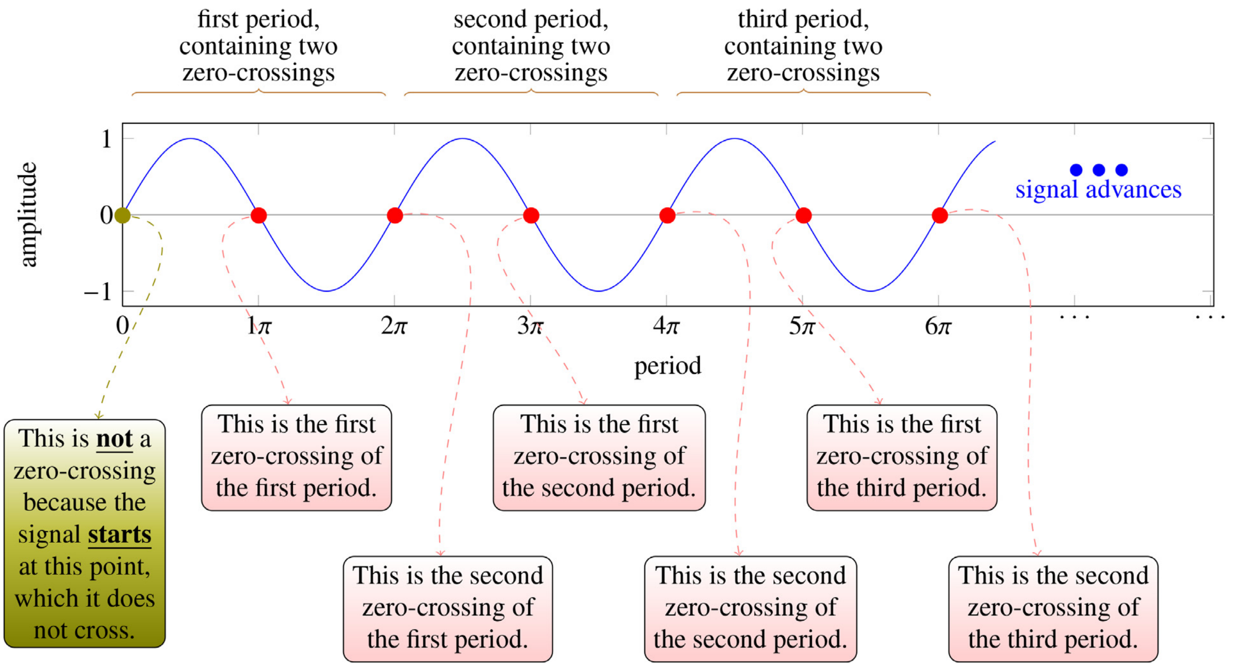

first period, containing two zero-crossings
second period, containing two zero-crossings
third period, containing two zero-crossings
amplitude
1
0
−1
0
1π
2π
3π
4π
5π
6π
period
signal advances
This is **not** a zero-crossing because the signal **starts** at this point, which it does not cross.
This is the first zero-crossing of the first period.
This is the first zero-crossing of the second period.
This is the first zero-crossing of the third period.
This is the second zero-crossing of the first period.
This is the second zero-crossing of the second period.
This is the second zero-crossing of the third period.

Fig. 3. A sine wave and its zero-crossings.

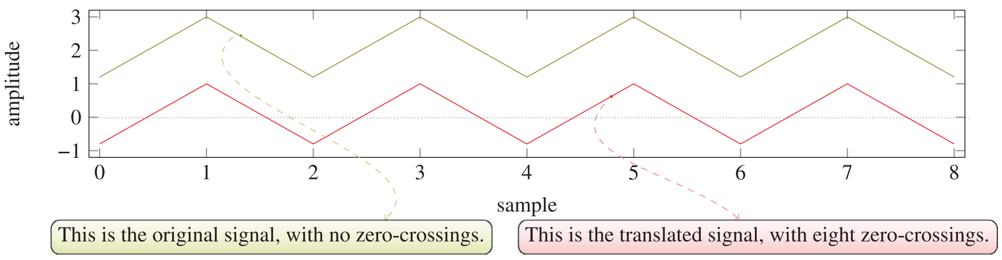


Fig. 4. The example original signal $s[\cdot] = \{\frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}, 3, \frac{12}{10}\}$, in olive, and its translated version, $\{-\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}, 1, -\frac{8}{10}\}$, with zero mean, in red. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

crossings. Therefore, the signal fundamental frequency is $\frac{G \cdot R}{2 \cdot M} = \frac{8 \cdot 36}{2 \cdot 9} = 16$ Hz.

In comparison with the most common features, ZCRs have the following advantages. First, they are extremely simple to be computed, with a linear order of time and space complexities [15]. Second, as mentioned above, they are the only features which reveal spectral information on input data without an explicit conversion from time to frequency-domain. Third, as consolidated in the literature, relevant problems on DSP and PRKbS, such as those involving speech processing [12], can benefit from them. Nonetheless, the disadvantages presented by ZCRs must be considered. Initially, spectral information on the signal under analysis is not complete, as obtained, for instance, with the Discrete Fourier Transform [13] or the Discrete Wavelet Transform [69]. In addition, a joint time-frequency mapping is not possible with them, i.e., frequency localisation can only be performed based on a manual-controlled partition of the signal. Lastly, features extracted on their bases may be considerably disturbed if originated from noisy inputs.

Summarising, ZCRs are neither better nor worse than other features, from a general point-of-view. Instead, they present advantages and disadvantages that have to be taken into account for each particular PRKbS problem. In any event, they should be considered every time that modest or incomplete spectral information is found to be useful. In order to complement the review on ZCRs, the remaining of this section is dedicated to describe their recent applications, those found in the literature.

In article [16], authors used ZCR in addition to a least-mean squares filter for speech enhancement. Their successful strategy consists of using ZCRs at the final stage of their algorithm in order to identify patterns, providing the desired improvements. Signal-to-noise ratio (SNR) increased about 22 dB in the signals analysed, confirming the important role of ZCR-based PRKbS. Speech enhancement based on ZCRs is also the focus of the paper [17], which points out that zero-crossing information is more accurate than the cross-correlation for the proposed task.

In association with short-time energy, the authors of article [18] applied ZCRs to distinguish speech from music. In conjunction with an ordinary k-means algorithm, ZCRs allowed the identification of quasi-periodic patterns that are key for their task, attaining 96.10% accuracy with 110 music clips and 140 speech files. Reciprocally, the authors of article [19] showed a ZCR-based estimator that works better than the sample autocorrelation method to analyse stationary Gaussian processes. Their work was successfully developed on the basis of a mathematical analysis of random noise. In [20], researchers proposed a method to estimate the frequency of a harmonic trend based on ZCRs. In addition to a low computation time, their results demonstrate the possible use of such features in practical applications.

An interesting aspect of ZCRs was shown by the authors of article [21], who analysed transient signals to demonstrate these can be accurately found based on zero-crossings. Applications related to the estimate of epochs in speech signals were performed, confirming the authors' assumptions. In conjunction with other fea-

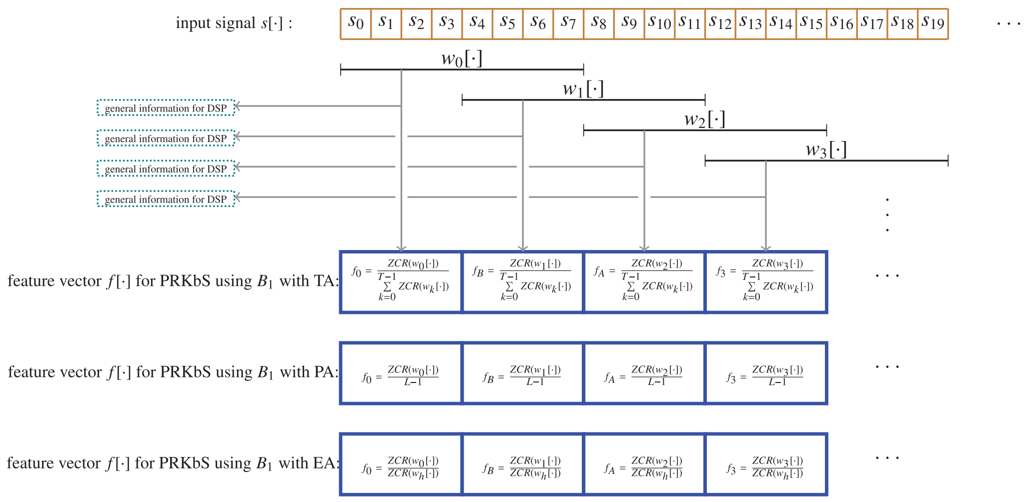

input signal $s[\cdot]$:
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} ...
$w_0[\cdot]$
$w_1[\cdot]$
$w_2[\cdot]$
$w_3[\cdot]$
general information for DSP
general information for DSP
general information for DSP
general information for DSP
feature vector $f[\cdot]$ for PRKbS using B_1 with TA:
$f_0 = \frac{ZCR(w_0[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$f_B = \frac{ZCR(w_1[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$f_A = \frac{ZCR(w_2[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$f_3 = \frac{ZCR(w_3[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
feature vector $f[\cdot]$ for PRKbS using B_1 with PA:
$f_0 = \frac{ZCR(w_0[\cdot])}{L-1}$
$f_B = \frac{ZCR(w_1[\cdot])}{L-1}$
$f_A = \frac{ZCR(w_2[\cdot])}{L-1}$
$f_3 = \frac{ZCR(w_3[\cdot])}{L-1}$
feature vector $f[\cdot]$ for PRKbS using B_1 with EA:
$f_0 = \frac{ZCR(w_0[\cdot])}{ZCR(w_h[\cdot])}$
$f_B = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$
$f_A = \frac{ZCR(w_2[\cdot])}{ZCR(w_h[\cdot])}$
$f_3 = \frac{ZCR(w_3[\cdot])}{ZCR(w_h[\cdot])}$

Fig. 5. 1D example for B_1 aiming DSP and PRKbS with its variants TA, PA and EA: sliding window with length $L = 8$ traversing $s[\cdot]$ with overlap $V = 50\%$. The symbols w_i represent the kth positioning of the window, for $k = 0, 1, 2, ..., T-1$, and w_h is the window that contains the highest number of ZCRs.

tures, such as energy, the authors of paper [22] present different methods to distinguish voiced from unvoiced segments in speech signals. Empirically, the size of the speech segments were determined for better accuracy during their successful analyses. Similar experiments were also performed by the authors of paper [23], confirming the findings. In [24], a practical and noise-robust speech recognition system based on ZCRs was developed. Authors showed improvements on baseline approaches at a rate of about 18.8%. Humanoid robots also benefit from ZCRs, for speech recognition and segregation purposes, according to the experiments described in [25].

In order to successfully predict epileptic seizures in scalp electroencephalogram signals, the authors of paper [26] modeled a Gaussian Mixture of ZCRs intervals of occurrence, obtaining relevant results. Interestingly, the authors of the paper [27] used a modified ZCR to determine fractal dimensions of biomedical signals. Similarly, in paper [28], authors evaluate a modified ZCR approach for the detection of heart arrhythmias. A prominent application of ZCRs can be found in paper [29], in which authors developed a brain-computer interface on their basis. A health monitoring scheme based on ZCRs characterises the work described in [30], for which interesting aspects of such features are pointed out.

Not surprisingly, a wide search on *Web of Science* and other scientific databases, aiming to find possible research articles describing applications of ZCRs on image processing and computer vision, i.e., 2D signals, returned a modest number of results: two conference papers, being one recent [31] and the other published twenty years ago [32], and one journal paper published almost thirty years ago [33]. Possibly, this is due to the fact that digital images usually have their pixels represented as being positive integer numbers, inhibiting the use of zero-crossings. In this study, in addition to the novel ZCR-based algorithms designed for 1D signals, 2D ones are also considered just after a proper pre-processing strategy discussed herein.

2. The proposed methods

Three different methods, i.e., B_1, B_2 and B_3, respectively inspired on A_1, A_2 and A_3 introduced in [1], are proposed in this section. Their corresponding details follow.

2.1. Method B$_1$

B_1, illustrated in Fig. 5, is the simplest method I present in this study, in which an 1D discrete-time signal $s[\cdot]$ of length M is considered as being the input. The procedure consists of a sliding rectangular window, w, of length L traversing the signal so that, for each placement, the ZCR over that position is determined. Each subsequent positioning overlaps in V% the previous one, being the surplus samples at the end of the signal, which are not long enough to be overlapped by a L-sample window, disposed. The restrictions $(2 \leqslant L \leqslant M)$ and $(0 \leqslant V < 100)$ are mandatory.

In a DSP context, the ZCRs computed over the fragments of $s[\cdot]$ may be directly used to determine the fundamental frequencies it contains, based on Eq. 2. On the other hand, in case PRKbS associated with handcrafted FE is the objective, as explained in [1]-pp.2, $s[\cdot]$ requires its conversion to a *feature vector*, $f[\cdot]$, of length $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V) \cdot L} \rfloor$, being $\lfloor \cdot \rfloor$ the floor operator. In this case, each f_k, $(0 \leqslant k \leqslant T-1)$, corresponds to the ZCR computed over the kth position of the window w. Of fundamental importance is the fact that, for handcrafted FE, $f[\cdot]$ requires normalisation prior to its use as an input for a classifier, as documented in [1]-pp.2.

There are, basically, three possible ways to normalise $f[\cdot]$: in relation to the total amount (TA) of zero-crossings, in relation to the maximum possible amount (PA) of zero-crossings and in relation to the maximum existing amount (EA) of zero-crossings. Each of the normalisations characterises a particular physical meaning for the ZCRs contained in $f[\cdot]$, being adequate for a specific task in PRKbS. Comments on each of them follow, nonetheless, all the normalisations force $f[\cdot]$ to express a *rate*, bringing the proper sense to the letter "R" used in the abbreviation "ZCR".

The division of each individual ZCR in $f[\cdot]$ by the sum of all ZCRs it contains, i.e.,

$$f_r \leftarrow \frac{f_r}{\left(\sum_{k=0}^{T-1} f_k\right)}, \qquad (0 \leqslant r \leqslant T-1),$$

characterises TA. Once this procedure is adopted, $\sum_{k=0}^{T-1} f_k = 1$. Physically, TA forces $f[\cdot]$ to express the fraction of ZCRs in each segment of $s[\cdot]$, being ideal to describe the way the fundamental frequencies of an input signal, $s[\cdot]$, vary in relation to its overall behaviour.

To force $f[\cdot]$ express individual spectral properties related to each fragment of $s[\cdot]$, in isolation, PA is required. The corresponding normalisation consists of dividing each ZCR in $f[\cdot]$ by $L-1$, i.e., the highest possible number of ZCRs inside a window of length L:

$$f_r \leftarrow \frac{f_r}{L-1}, \qquad (0 \leqslant r \leqslant T-1).$$

With this procedure, $f_k \leq 1$, for $(0 \leqslant k \leqslant T-1)$. The closer a certain f_k is to 0 or to 1, respectively, the lowest or highest the fundamental frequency at the corresponding window is, disregarding the remaining fragments of $s[\cdot]$.

Lastly, EA is chosen whenever evaluation by comparison is needed, particularly forcing the highest ZCR in $f[\cdot]$ to be 1 and adjusting the remaining ones, proportionally, within the range (0–1). The corresponding procedure consists of dividing each individual ZCR in $f[\cdot]$ by the highest unnormalised ZCR contained in it, the one computed over the window placement named w_h, i.e.,

$$f_r \leftarrow \frac{f_r}{ZCR(w_h)}, \qquad (0 \leqslant r \leqslant T-1).$$

All the previous formulations and concepts can be easily extended to a 2D signal, $m[\cdot][\cdot]$, with N rows and M columns, which represent, respectively, the height and width of the corresponding image. As in the unidimensional case, the computation of a bidimensional ZCR requires all the values in $m[\cdot][\cdot]$ to be previously shifted so that its arithmetic mean becomes equal zero, i.e.,

$$m_{p,q} \leftarrow m_{p,q} - \frac{\left(\sum_{i=0}^{N-1}\sum_{j=0}^{M-1} m_{i,j}\right)}{M \cdot N}, \quad (0 \leqslant p \leqslant M-1), \quad (0 \leqslant q \leqslant N-1). \tag{4}$$

Once $m[\cdot][\cdot]$ presents zero mean, its ZCR is simply the sum of individual ZCRs in each row and column, i.e.,

$$\text{ZCR}(m[\cdot][\cdot]) = \frac{1}{2}\sum_{i=0}^{N-1}\sum_{j=0}^{M-2}|sign(m_{i,j}) - sign(m_{i,j+1})| + \frac{1}{2}\sum_{j=0}^{M-1}\sum_{i=0}^{N-2}|sign(m_{i,j}) - sign(m_{i+1,j})|. \tag{5}$$

Similarly to 1D signals, ZCRs computed in 2D are useful for both DSP and FE in PRKbSS, as illustrated in Fig. 6. In the latter, the case of interest, the feature vector, $f[\cdot]$, contains not only T, but $T \cdot P$ elements, being $P = \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V) \cdot L} \rfloor$, as in the bidimensional case of method A_1, explained in [1]-pp.3. During the analysis, $m[\cdot][\cdot]$ is traversed along the horizontal orientation based on T placements of the square window w of side L, being $L < M$ and $L < N$. Then, the process is repeated for each one of the P shifts along the vertical orientation.

TA, PA and EA are also the possible normalisations for the 2D version of B_1 applied for FE in PRKbSS. Particularly, TA requires each component of $f[\cdot]$ to be divided by the sum of all ZCRs it contains, i.e., the sum of all the values in that vector prior to any normalisation. On the other hand, if PA is adopted, each element in $f[\cdot]$ is divided by the maximum possible number of ZCRs inside w, i.e.,

$$\underbrace{(L-1)}_{\substack{\text{maximum ZCR}\\ \text{in one row}}} \cdot \underbrace{(L)}_{\substack{\text{number of}\\ \text{rows}}} + \underbrace{(L-1)}_{\substack{\text{maximum ZCR}\\ \text{in one column}}} \cdot \underbrace{(L)}_{\substack{\text{number of}\\ \text{columns}}} = 2 \cdot L \cdot (L-1).$$

Lastly, the choice for EA implies that each component of $f[\cdot]$ is divided by the highest ZCR contained in it, the one computed over the window placement named w_h.

Exactly as in A_1 [1], for both 1D and 2D signals, respectively, B_1 is only capable of generating a T, or a $T \cdot P$, sample-long vector $f[\cdot]$ if the value of L is subjected to the value of M, or M and N. Thus, the value of L intrinsically depends on the length of the input 1D signal $s[\cdot]$, or the dimensions of the input 2D matrix $m[\cdot][\cdot]$, bringing a disadvantage: irregular, temporal or spatial analysis. Oppositely, the advantage is that a few sequential elements of $f[\cdot]$, obtained by predefining L, T and P, allow the detection of some particular event in the 1D or 2D signal under analysis.

The algorithms 1, 2 and 3, respectively, contain the source code in C/C++ programming language that implement method B_1 with the normalisations TA, PA and EA, all of them for 1D input signals. At variance with this, algorithms 4, 5 and 6 correspond, respectively, to the 2D versions of B_1 with the same normalisations.

Algorithm 1 : fragment of C++ code for method B_1 in 1D, adopting the normalisation TA.

```
//...
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than M */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100 * M − L * V)/((100 − V)*L));
int ZCR = 0; // ZCR is the total number of zero-crossings over all the window placements, required for normalisation
double *f = new double[T]; // dynamic vector declaration
for(int k = 0; k < T; k++)
        {
        f[k] = 0;
            for(int    i = k * ((int)(((100 − V)/100.0) * L)); i < k * ((int)(((100 − V)/100.0) * L)) + L − 1; i++)
                f[k]+ =(s[i] * s[i + 1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        ZCR+ = f[k];
        }
for(int k = 0; k < T; k++) // normalisation
        f[k]/ = (double)(ZCR); /* the casting, i.e., the explicit conversion of ZCR from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
// at this point, the feature vector, f[·], is ready
//...
```

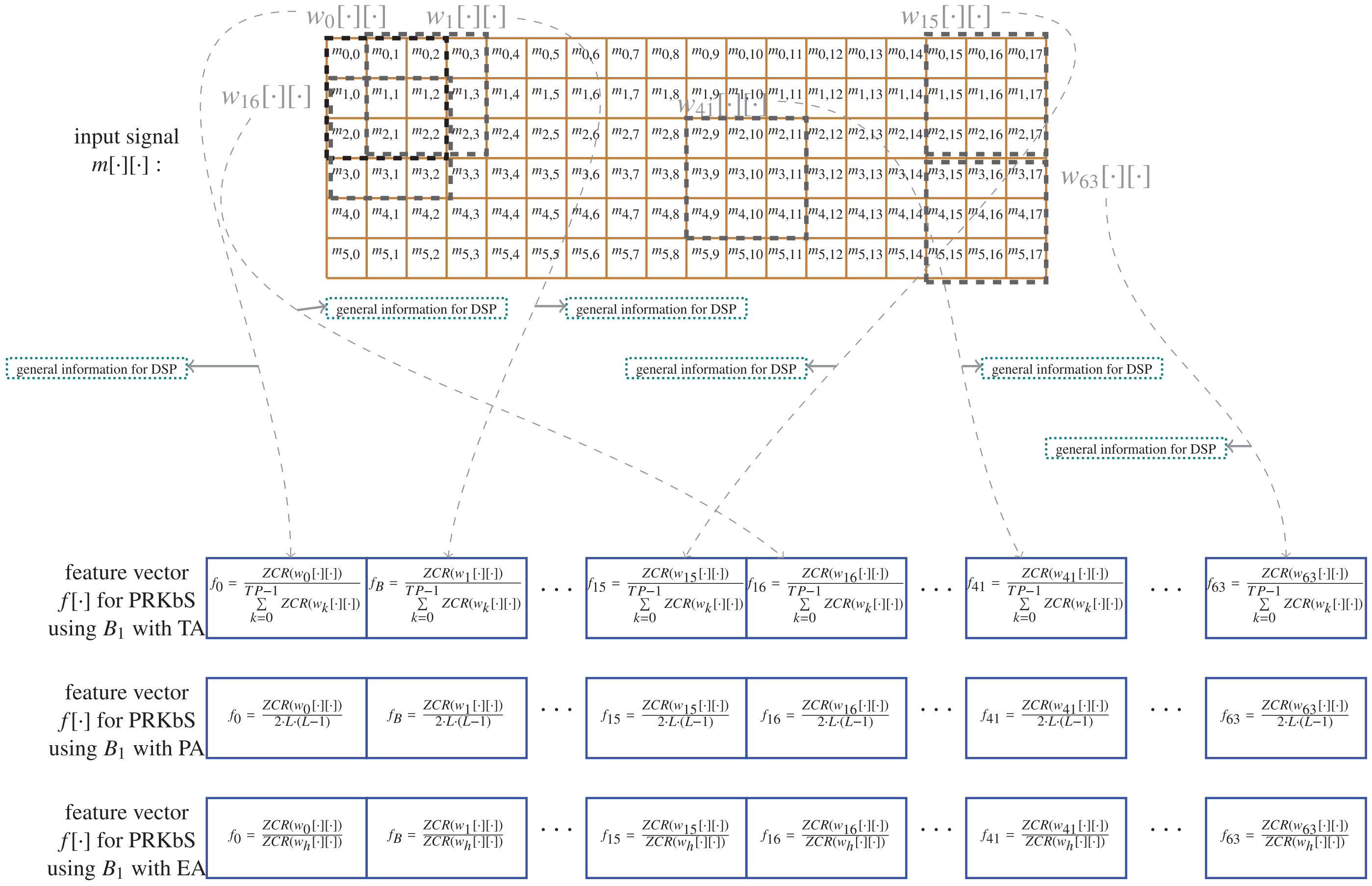
Algorithm 2 : fragment of C++ code for method B_1 in 1D, adopting the normalisation PA.

```
//...
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than M */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100 * M − L * V)/((100 − V)*L));
double *f = new double[T]; // dynamic vector declaration
for(int k = 0; k < T; k++)
        {
        f[k] = 0;
            for(int    i = k * ((int)(((100 − V)/100.0) * L)); i < k * ((int)(((100 − V)/100.0) * L)) + L − 1; i++)
                f[k]+ =(s[i] * s[i + 1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        f[k]/ = (double)(L − 1); /* the casting, i.e., the explicit conversion of L from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
        }
// at this point, the feature vector, f[·], is ready
//...
```

2.2. Method B$_2$

B_2, as A_2 in [1], is also based on a sliding window w traversing $s[\cdot]$, or $m[\cdot][\cdot]$. Two differences, however, exist: there are no overlaps and the window length for 1D, or the rectangle sizes for 2D, vary. Thus, $s[\cdot]$ or $m[\cdot][\cdot]$ are inspected in different levels of resolution.

After applying Eq. (3) or Eq. (4), respectively to remove the mean of $s[\cdot]$ or of $m[\cdot][\cdot]$, the feature vector, $f[\cdot]$, is defined as being the concatenation of Q sub-vectors of different dimensions, i.e., $f[\cdot] = \{\xi_1[\cdot]\} \cup \{\xi_2[\cdot]\} \cup \{\xi_3[\cdot]\} \cup \ldots \cup \{\xi_Q[\cdot]\}$. For 1D, each sub-vector is created by placing w over T non-overlapping sequential positions



input signal $m[\cdot][\cdot]$:
$w_0[\cdot][\cdot]$
$w_1[\cdot][\cdot]$
$w_{15}[\cdot][\cdot]$
$w_{16}[\cdot][\cdot]$
$w_{41}[\cdot][\cdot]$
$w_{63}[\cdot][\cdot]$
general information for DSP
feature vector $f[\cdot]$ for PRKbS using B_1 with TA
feature vector $f[\cdot]$ for PRKbS using B_1 with PA
feature vector $f[\cdot]$ for PRKbS using B_1 with EA
$f_0 = \frac{ZCR(w_0[\cdot][\cdot])}{\sum_{k=0}^{TP-1} ZCR(w_k[\cdot][\cdot])}$
$f_0 = \frac{ZCR(w_0[\cdot][\cdot])}{2\cdot L\cdot(L-1)}$
$f_0 = \frac{ZCR(w_0[\cdot][\cdot])}{ZCR(w_h[\cdot][\cdot])}$

Fig. 6. 2D example for B_1 aiming DSP and PRKbS with its variants TA, PA and EA: sliding square with length $L = 3$ traversing $m[\cdot][\cdot]$ with overlap $V = 66.67\%$. Again, $w_k[\cdot]$ indicate the kth position of the window, for $k = 0, 1, 2, ..., (T \cdot P) - 1$. Dashed squares in the arbitrary positions 0, 1, 15, 16, 41 and 63, are shown.

of $s[\cdot]$ and then calculating the normalised ZCRs using TA, PA, or EA, as previously explained during the description of B_1, i.e.,:

- subvector $\xi_1[\cdot]$ is obtained by letting $L = \lfloor \frac{M}{2} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 2$, to traverse $s[\cdot]$ and get the normalised ZCRs;
- idem to subvector $\xi_2[\cdot]$, obtained by letting $L = \lfloor \frac{M}{3} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 3$;
- idem to subvector $\xi_3[\cdot]$, obtained by letting $L = \lfloor \frac{M}{5} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 5$;
- . . .
- idem to subvector $\xi_Q[\cdot]$, obtained by letting $L = \lfloor \frac{M}{X} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = X$.

Q is defined on the basis of the desired refinement and, similarly, the values 2, 3, 5, 7, 9, 11, 13, 17, ..., X are choices for T, that is essentially restricted to prime numbers in order to avoid one subvector to be a linear combination of another, implying in no gain for classification.

For the 2D case, each sub-vector is created by framing $m[\cdot][\cdot]$ with $T \cdot P$ non-overlapping rectangles, being $T = P$ prime numbers, to compute the corresponding normalised ZCRs, i.e.,

- subvector $\xi_1[\cdot]$ is created by letting $L = \lfloor \frac{M}{2} \rfloor$ and $V = 0\%$ to obtain $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 2$ and then by letting $L = \lfloor \frac{N}{2} \rfloor$ and $V = 0\%$ to obtain $P = \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 2$. Subsequently, $m[\cdot][\cdot]$ is traversed by $T \cdot P = 2 \cdot 2 = 4$ non-overlapping rectangles ;
- idem to subvector $\xi_2[\cdot]$, obtained by letting $L = \lfloor \frac{M}{3} \rfloor$ and $V = 0\%$ and then $L = \lfloor \frac{N}{3} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 3$ and $P = \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 3$, respectively, implying that $T \cdot P = 3 \cdot 3 = 9$ non-overlapping rectangles traverse $m[\cdot][\cdot]$;
- idem to subvector $\xi_3[\cdot]$, obtained by letting $L = \lfloor \frac{M}{5} \rfloor$ and $V = 0\%$ and then $L = \lfloor \frac{N}{5} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 5$ and $P = \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V)\cdot L} \rfloor = 5$, respectively, implying that $T \cdot P = 5 \cdot 5 = 25$ non-overlapping rectangles traverse $m[\cdot][\cdot]$;
- . . .
- idem to subvector $\xi_Q[\cdot]$, obtained by letting $L = \lfloor \frac{M}{X} \rfloor$ and $V = 0\%$ and then $L = \lfloor \frac{N}{X} \rfloor$ and $V = 0\%$, which that $T = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V)\cdot L} \rfloor = X$ and $P = \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V)\cdot L} \rfloor = X$, respectively, implying that $T \cdot P = X \cdot X = X^2$ non-overlapping rectangles traverse $m[\cdot][\cdot]$;

In the 2D version of B_2, the normalisations TA and EA are implemented exactly as they were in B_1. One particular note regarding the normalisation PA is, however, important. Differently to B_1 in 2D, in which L is the same for both horizontal and vertical orientations, B_2 divides the input image into rectangles, i.e., the horizontal and vertical sides are $\lfloor \frac{M}{X} \rfloor$ and $\lfloor \frac{N}{X} \rfloor$, respectively. Thus, each element of $f[\cdot]$ is not divided by $2 \cdot L \cdot (L-1)$, but by

$$\underbrace{\left(\lfloor \frac{M}{X} \rfloor - 1\right)}_{\substack{\text{maximum ZCR} \\ \text{in one row}}} \cdot \underbrace{\lfloor \frac{N}{X} \rfloor}_{\substack{\text{number of} \\ \text{rows}}} + \underbrace{\left(\lfloor \frac{N}{X} \rfloor - 1\right)}_{\substack{\text{maximum ZCR} \\ \text{in one column}}} \cdot \underbrace{\lfloor \frac{M}{X} \rfloor}_{\substack{\text{number of} \\ \text{columns}}}$$

$$= 2 \cdot \lfloor \frac{M}{X} \rfloor \cdot \lfloor \frac{N}{X} \rfloor - \lfloor \frac{M}{X} \rfloor - \lfloor \frac{N}{X} \rfloor,$$

Algorithm 3 : fragment of C++ code for method B_1 in 1D, adopting the normalisation EA.

```
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean += s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k] -= mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than M */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100*M - L*V)/((100 - V)*L));
int highest_ZCR = 0;
double *f = new double[T]; // dynamic vector declaration
for(int k = 0; k < T; k++)
        {
        f[k] = 0;
            for(int i = k*((int)(((100 - V)/100.0)*L)); i < k*((int)(((100 - V)/100.0)*L)) + L - 1; i++)
                f[k] += (s[i]*s[i+1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        if (f[k] > highest_ZCR)
                highest_ZCR = f[k];
        }
for(int k = 0; k < T; k++)
        f[k] /= (double)(highest_ZCR);/* the casting, i.e., the explicit conversion of highest_ZCR from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
// at this point, the feature vector, f[·], is ready
```

Algorithm 4 : fragment of C++ code for method B_1 in 2D, adopting the normalisation TA.

```
// ensure that m[·][·], with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean += m[p][q]/(double)(M*N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q] -= mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than the higher between M and N */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100*M - L*V)/((100 - V)*L));
int P = (int)((100*N - L*V)/((100 - V)*L));
int ZCR = 0; // ZCR is the total number of zero-crossings over all the window placements, required for normalisation
double *f = new double[T*P]; // dynamic vector declaration
for(int k = 0; k < T*P; k++)
        {
        f[k] = 0;
            for(int i = k*((int)(((100 - V)/100.0)*L)); i < k*((int)(((100 - V)/100.0)*L)) + L; i++)
                for(int j = k*((int)(((100 - V)/100.0))*L); j < k*((int)(((100 - V)/100.0))*L) + L - 1; j++)
                        f[k] += (m[i][j]*m[i][j+1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
            for(int i = k*((int)(((100 - V)/100.0)*L)); i < k*((int)(((100 - V)/100.0)*L)) + L - 1; i++)
                for(int j = k*((int)(((100 - V)/100.0))*L); j < k*((int)(((100 - V)/100.0))*L) + L; j++)
                        f[k] += (m[i][j]*m[i+1][j] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        ZCR += f[k];
        }
for(int k = 0; k < T*P; k++)
        f[k] /= (double)(ZCR); /* the casting, i.e., the explicit conversion of ZCR from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
// at this point, the feature vector, f[·], is ready
```

Algorithm 5 : fragment of C++ code for method B_1 in 2D, adopting the normalisation PA.

```
// ensure that m[·][·], with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean += m[p][q]/(double)(M*N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q] -= mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than the higher between M and N */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100*M - L*V)/((100 - V)*L));
int P = (int)((100*N - L*V)/((100 - V)*L));
double *f = new double[T*P]; // dynamic vector declaration
for(int k = 0; k < T*P; k++)
        {
        f[k] = 0;
         for(int i = k*((int)(((100 - V)/100.0)*L)); i < k*((int)(((100 - V)/100.0)*L)) + L; i++)
                for(int j = k*((int)(((100 - V)/100.0))*L); j < k*((int)(((100 - V)/100.0))*L) + L - 1; j++)
                        f[k] += (m[i][j]*m[i][j+1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
         for(int i = k*((int)(((100 - V)/100.0)*L)); i < k*((int)(((100 - V)/100.0)*L)) + L - 1; i++)
                for(int j = k*((int)(((100 - V)/100.0))*L); j < k*((int)(((100 - V)/100.0))*L) + L; j++)
                        f[k] += (m[i][j]*m[i+1][j] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        }
for(int k = 0; k < T*P; k++)
        f[k] /= (double)(2*L*(L - 1)); the casting, i.e., the explicit conversion of L from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
// at this point, the feature vector, f[·], is ready
```

that corresponds to the maximum possible number of zero-crossings in each rectangular sub-image.

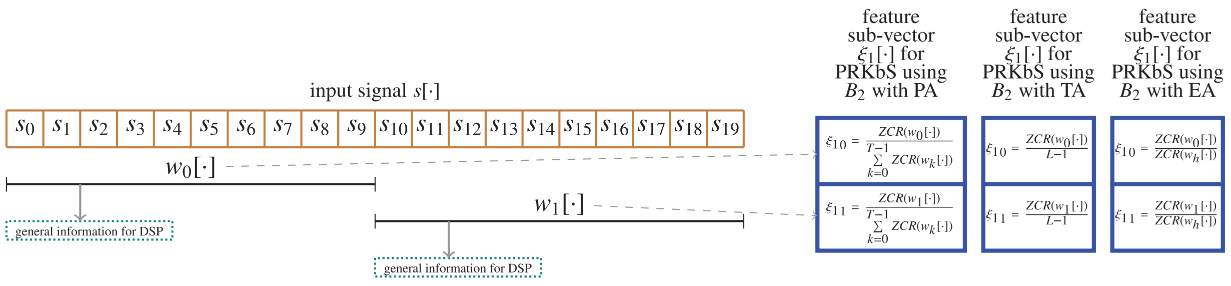
Figs. 7 and 8 show the sliding window for 1D and the sliding rectangle for 2D, respectively, for TA, PA and EA. In addition, the algorithms 8, 7 and 9 contain the corresponding 1D implementations. The 2D ones are in the algorithms 10–12.

2.3. Method B_3

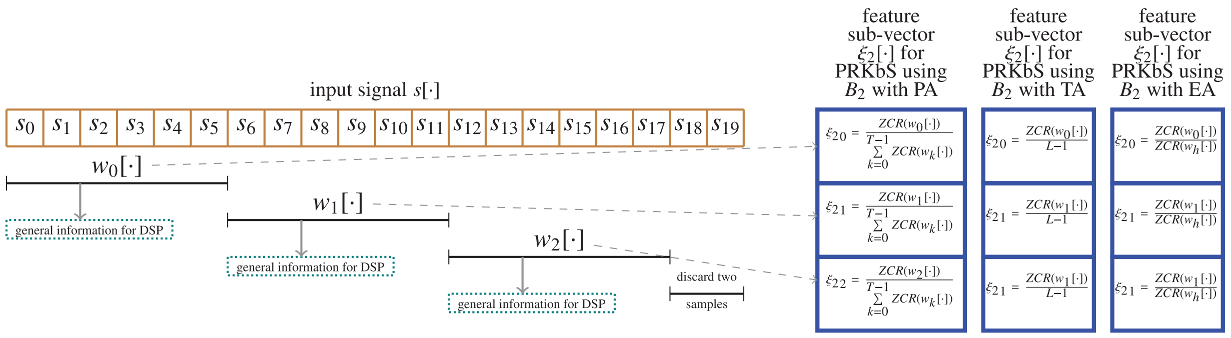
As described above, B_1 and B_2 focus on measuring the levels of normalised ZCRs over windows or rectangles of certain dimensions. B_3, on the other hand, is quite similar to A_3 [1] and consists of determining the proportional lengths, or areas, of the signal under analysis that are required to reach predefined percentages of the total ZCR. Normalisations do not apply in this case. The direct consequence of this approach is the characterisation of B_3 as being ideal to inspect the constancy in frequency of the physical entity responsible for generating $s[\cdot]$, or $m[\cdot][\cdot]$.

Specifically, C is defined as being the critical base-level of ZCRs, ($0 < C < 100$), and then, for 1D, the feature vector $f[\cdot]$ of size T is determined as follows:

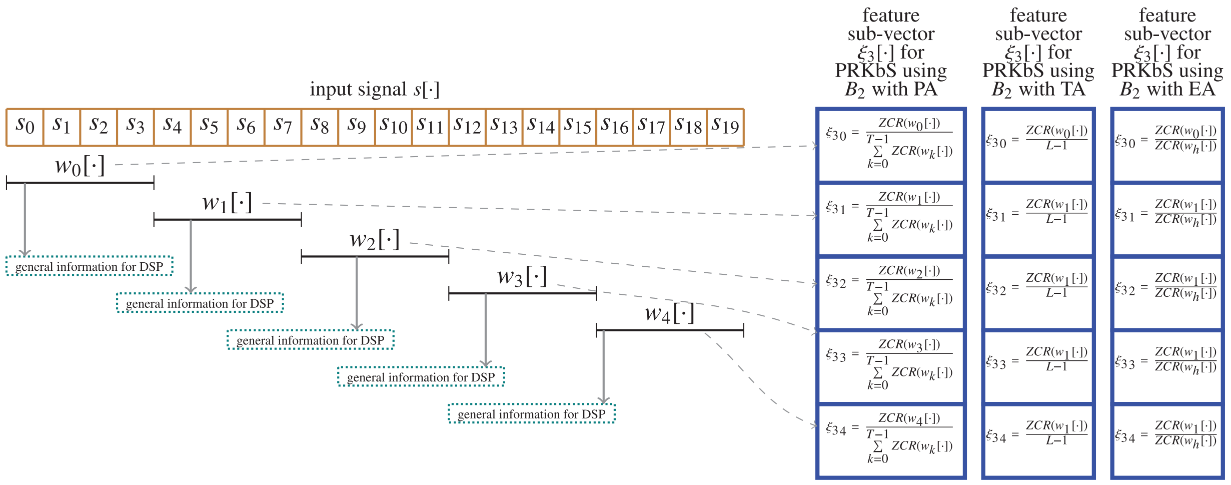
- f_0 is the proportion of the length of $s[\cdot]$, i.e., M, starting from its beginning, which is covered by the window placement $w_0[\cdot]$, required to reach C% of the total ZCR;
- f_B is the proportion of the length of $s[\cdot]$, i.e., M, starting from its beginning, which is covered by the window placement $w_1[\cdot]$, required to reach $2 \cdot C$% of the total ZCR;


input signal $s[\cdot]$
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19}
$w_0[\cdot]$
$w_1[\cdot]$
general information for DSP
general information for DSP
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with PA
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with TA
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with EA
$\xi_{10} = \frac{ZCR(w_0[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{11} = \frac{ZCR(w_1[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{10} = \frac{ZCR(w_0[\cdot])}{L-1}$
$\xi_{11} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{10} = \frac{ZCR(w_0[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{11} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$

(a) windowing the input signal to form the first sub-vector


input signal $s[\cdot]$
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19}
$w_0[\cdot]$
$w_1[\cdot]$
$w_2[\cdot]$
general information for DSP
general information for DSP
general information for DSP
discard two samples
feature sub-vector $\xi_2[\cdot]$ for PRKbS using B_2 with PA
feature sub-vector $\xi_2[\cdot]$ for PRKbS using B_2 with TA
feature sub-vector $\xi_2[\cdot]$ for PRKbS using B_2 with EA
$\xi_{20} = \frac{ZCR(w_0[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{21} = \frac{ZCR(w_1[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{22} = \frac{ZCR(w_2[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{20} = \frac{ZCR(w_0[\cdot])}{L-1}$
$\xi_{21} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{21} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{20} = \frac{ZCR(w_0[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{21} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{21} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$

(b) windowing the input signal to form the second sub-vector


input signal $s[\cdot]$
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19}
$w_0[\cdot]$
$w_1[\cdot]$
$w_2[\cdot]$
$w_3[\cdot]$
$w_4[\cdot]$
general information for DSP
general information for DSP
general information for DSP
general information for DSP
general information for DSP
feature sub-vector $\xi_3[\cdot]$ for PRKbS using B_2 with PA
feature sub-vector $\xi_3[\cdot]$ for PRKbS using B_2 with TA
feature sub-vector $\xi_3[\cdot]$ for PRKbS using B_2 with EA
$\xi_{30} = \frac{ZCR(w_0[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{31} = \frac{ZCR(w_1[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{32} = \frac{ZCR(w_2[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{33} = \frac{ZCR(w_3[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{34} = \frac{ZCR(w_4[\cdot])}{\sum_{k=0}^{T-1} ZCR(w_k[\cdot])}$
$\xi_{30} = \frac{ZCR(w_0[\cdot])}{L-1}$
$\xi_{31} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{32} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{33} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{34} = \frac{ZCR(w_1[\cdot])}{L-1}$
$\xi_{30} = \frac{ZCR(w_0[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{31} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{32} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{33} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$
$\xi_{34} = \frac{ZCR(w_1[\cdot])}{ZCR(w_h[\cdot])}$

(c) windowing the input signal to form the third sub-vector

Fig. 7. 1D example for B_2 assuming $Q = 3$: (a) sliding window, with length $L = \lfloor \frac{M}{2} \rfloor = \lfloor \frac{20}{2} \rfloor = 10$ traversing $s[\cdot]$ in order to compose $\xi_1[\cdot]$; (b) sliding window with length $L = \lfloor \frac{M}{3} \rfloor = \lfloor \frac{20}{3} \rfloor = 6$ traversing $s[\cdot]$ in order to compose $\xi_2[\cdot]$; (c) sliding window with length $L = \lfloor \frac{M}{5} \rfloor = \lfloor \frac{20}{5} \rfloor = 4$ traversing $s[\cdot]$ in order to compose $\xi_3[\cdot]$. The window positions do not overlap and the symbols w_i indicate the i^{th} window position, for $i = 0, 1, 2, ..., T-1$.

- f_A is the proportion of the length of $s[\cdot]$, i.e., M, starting from its beginning, which is covered by the window placement $w_2[\cdot]$, required to reach $3 \cdot C$% of the total ZCR;
- . . .
- f_{T-1} is the proportion of the length of $s[\cdot]$, i.e., M, starting from its beginning, which is covered by the window placement $w_{T-1}[\cdot]$, required to reach $(T \cdot C)$% of the total ZCR, so that $(T \cdot C) < 100$%;

For B_3, the value of T is defined as being:

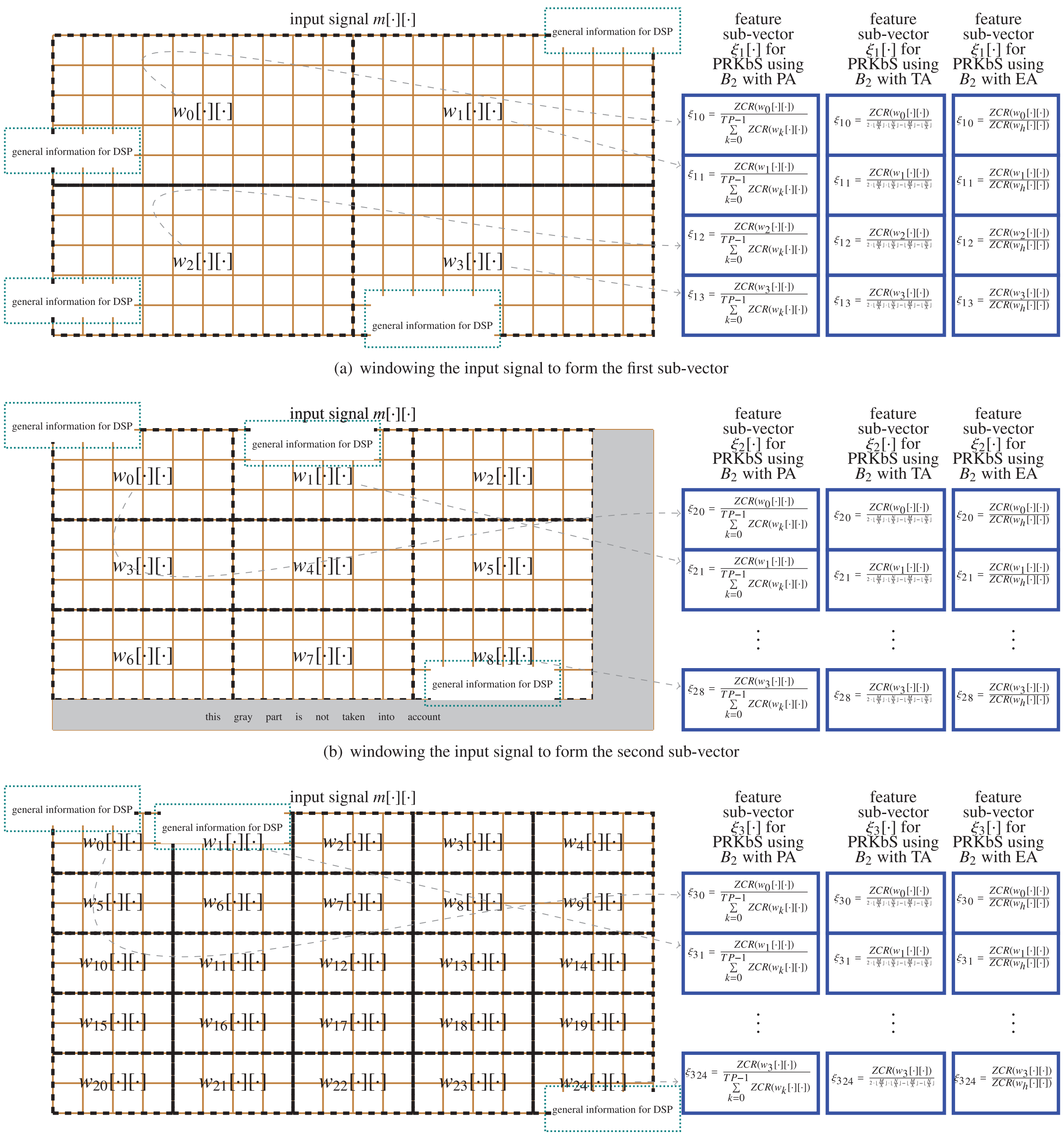
$$T = \begin{cases} \frac{100}{C} - 1 & \text{if } C \text{ is multiple of 100;} \\ \lfloor \frac{100}{C} \rfloor & \text{otherwise.} \end{cases}$$

The 2D version of B_3 implies that $f[\cdot]$, with the same size T, is determined as follows:

- f_0 is the proportion of $m[\cdot][\cdot]$ area, i.e., $M \cdot N$, starting from $m_{0,\,0}$ and covered by the $\lfloor \alpha_0 \rfloor$ x $\lfloor \beta_0 \rfloor$ rectangle $w_0[\cdot][\cdot]$, required to reach C% of the total ZCR;
- f_B is the proportion of $m[\cdot][\cdot]$ area, i.e., $M \cdot N$, starting from $m_{0,\,0}$ and covered by the $\lfloor \alpha_1 \rfloor$ x $\lfloor \beta_1 \rfloor$ rectangle $w_1[\cdot][\cdot]$, required to reach $2 \cdot C$% of the total ZCR;
- f_A is the proportion of $m[\cdot][\cdot]$ area, i.e., $M \cdot N$, starting from $m_{0,\,0}$ and covered by the $\lfloor \alpha_2 \rfloor$ x $\lfloor \beta_2 \rfloor$ rectangle $w_2[\cdot][\cdot]$, required to reach $3 \cdot C$% of the total ZCR;
- . . .
- f_{TP-1} is the proportion of $m[\cdot][\cdot]$ area, i.e., $M \cdot N$, starting from $m_{0,\,0}$ and covered by the $\lfloor \alpha_{T-1} \rfloor$ x $\lfloor \beta_{T-1} \rfloor$ rectangle $w_{T-1}[\cdot][\cdot]$, required to reach $T \cdot C$% of the total ZCR, so that $(T \cdot C) < 100$%;

The values of α_i and β_i, $(0 \leqslant i \leqslant T-1)$, are determined according to the following rule, the exact same used for A_3 in 2D [1][1]:

[1] I will take this opportunity to correct an error in my previous published tutorial [1]-pp.270 regarding the description of A_3 in 2D: the way α_i and β_i vary is in accordance with a relationship between N and M, as shown above, instead of α_i and


input signal $m[\cdot][\cdot]$
general information for DSP
$w_0[\cdot][\cdot]$ $w_1[\cdot][\cdot]$ $w_2[\cdot][\cdot]$ $w_3[\cdot][\cdot]$
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with PA
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with TA
feature sub-vector $\xi_1[\cdot]$ for PRKbS using B_2 with EA
$\xi_{10} = \frac{ZCR(w_0[\cdot][\cdot])}{\sum_{k=0}^{TP-1} ZCR(w_k[\cdot][\cdot])}$
$\xi_{10} = \frac{ZCR(w_0[\cdot][\cdot])}{ZCR(w_h[\cdot][\cdot])}$
this gray part is not taken into account
feature sub-vector $\xi_2[\cdot]$ for PRKbS using B_2 with PA
feature sub-vector $\xi_3[\cdot]$ for PRKbS using B_2 with PA

(a) windowing the input signal to form the first sub-vector

(b) windowing the input signal to form the second sub-vector

(c) windowing the input signal to form the third sub-vector

Fig. 8. 2D example for B_2 assuming $Q = 3$ subvectors: [above] sliding square with length $\{\lfloor \frac{M}{2} \rfloor x \lfloor \frac{N}{2} \rfloor\} = \{\lfloor \frac{10}{2} \rfloor x \lfloor \frac{20}{2} \rfloor\} = 5x10$ traversing $m[\cdot][\cdot]$ in order to compose $\xi_1[\cdot]$; [middle] sliding square with length $\{\lfloor \frac{M}{3} \rfloor x \lfloor \frac{N}{3} \rfloor\} = \{\lfloor \frac{10}{3} \rfloor x \lfloor \frac{20}{3} \rfloor\} = 3x6$ traversing $m[\cdot][\cdot]$ in order to compose $\xi_2[\cdot]$; [below] sliding square with length $\{\lfloor \frac{M}{5} \rfloor x \lfloor \frac{N}{5} \rfloor\} = \{\lfloor \frac{10}{5} \rfloor x \lfloor \frac{20}{5} \rfloor\} = 2x4$ traversing $m[\cdot][\cdot]$ in order to compose $\xi_3[\cdot]$. Again, w_i indicates the i^{th} window position, for $i = 0, 1, 2, ..., (T \cdot P) - 1$, with no overlap. Dashed squares represent the sliding window in all possible positions.

Beginning: $(\alpha_i \leftarrow 0)$ and $(\beta_i \leftarrow 0)$, unconditionally.

$$\text{repeat}\begin{cases}(\alpha_i \leftarrow \alpha_i + 1) \ \text{and} \ (\beta_i \leftarrow \beta_i + 1) \ \text{if} \ (N = M) \\ (\alpha_i \leftarrow \alpha_i + 1) \ \text{and} \ (\beta_i \leftarrow \beta_i + \frac{M}{N}) \ \text{if} \ (N > M) \\ (\alpha_i \leftarrow \alpha_i + \frac{N}{M}) \ \text{and} \ (\beta_i \leftarrow \beta_i + 1) \ \text{otherwise.}\end{cases}$$

until the desired level of energy, i.e., C, $2 \cdot C$, $3 \cdot C$, ..., $T \cdot C$ is reached.

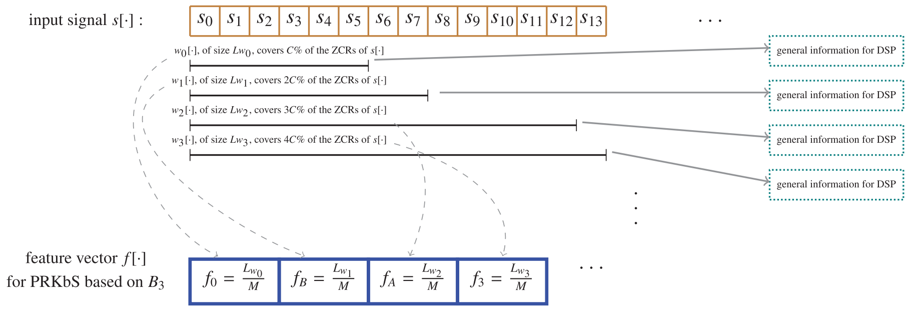
End. Figs. 9 and 10, and algorithms 13 and 14[2], complement my explanations regarding B_3, for both 1D and 2D, respectively.

2.4. ZCRs are neurocomputing agents

In this subsection, the trail for an interesting point-of-view is explained. Additionally to Eq. 1, ZCRs may also be counted based

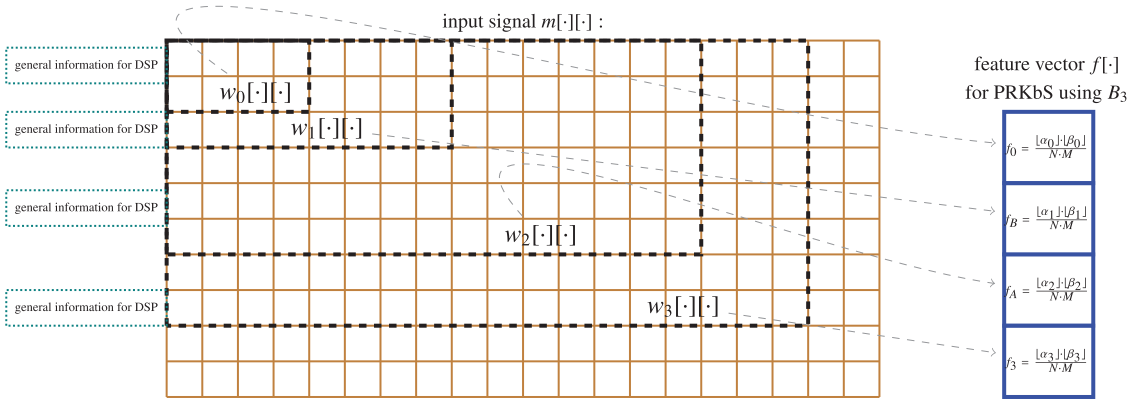
β_i themselves, as originally documented in that paper. A corrigendum is available on-line at http://dx.doi.org/10.1016/j.neucom.2016.04.001 with details.

[2] The algorithm for A_3 in 2D, originally described in [1]-pp.273, also requires the same corrections I mentioned in the previous footnote, as described in the corrigendum.



input signal $s[\cdot]$:
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12} s_{13} . . .
$w_0[\cdot]$, of size Lw_0, covers C% of the ZCRs of $s[\cdot]$
$w_1[\cdot]$, of size Lw_1, covers 2C% of the ZCRs of $s[\cdot]$
$w_2[\cdot]$, of size Lw_2, covers 3C% of the ZCRs of $s[\cdot]$
$w_3[\cdot]$, of size Lw_3, covers 4C% of the ZCRs of $s[\cdot]$
general information for DSP
general information for DSP
general information for DSP
general information for DSP
feature vector $f[\cdot]$ for PRKbS based on B_3
$f_0 = \frac{L_{w_0}}{M}$
$f_B = \frac{L_{w_1}}{M}$
$f_A = \frac{L_{w_2}}{M}$
$f_3 = \frac{L_{w_3}}{M}$

Fig. 9. 1D example for B_3, where L_{w_i} represents the length of the window $w_i[\cdot]$, for $i = 0, 1, 2, 3, ..., T-1$.



input signal $m[\cdot][\cdot]$:
general information for DSP
general information for DSP
general information for DSP
general information for DSP
$w_0[\cdot][\cdot]$
$w_1[\cdot][\cdot]$
$w_2[\cdot][\cdot]$
$w_3[\cdot][\cdot]$
feature vector $f[\cdot]$ for PRKbS using B_3
$f_0 = \frac{\lfloor\alpha_0\rfloor\cdot\lfloor\beta_0\rfloor}{N\cdot M}$
$f_B = \frac{\lfloor\alpha_1\rfloor\cdot\lfloor\beta_1\rfloor}{N\cdot M}$
$f_A = \frac{\lfloor\alpha_2\rfloor\cdot\lfloor\beta_2\rfloor}{N\cdot M}$
$f_3 = \frac{\lfloor\alpha_3\rfloor\cdot\lfloor\beta_3\rfloor}{N\cdot M}$

Fig. 10. 2D example for B_3, where $w_i[\cdot]$ corresponds to the ith window, and α_i and β_i represent, respectively, its height and width, for $i = 0, 1, 2, 3, ..., TP-1$.

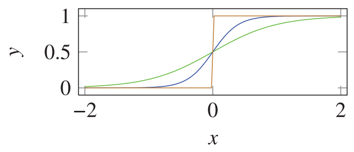


Fig. 11. The sigmoide function, $y = \frac{1}{1+e^{-\gamma\cdot x}}$, exemplified for different values of γ: 2, 5 and 1000, respectively drawn in green, blue and brown. The proposed strategy requires $\gamma >> 0$ aiming at a response as the one drawn in brown.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

on a different strategy, which is the one I use in my algorithms: two adjacent samples of a discrete-time signal, lets say s_i and s_{i+1}, cross zero whenever their product is negative. Thus,

$$\frac{s_i \cdot s_{i+1}}{|s_i \cdot s_{i+1}|} = \begin{cases} -1 & \text{if there is a zero-crossing between } s_i \text{ and } s_{i+1} \\ 1 & \text{otherwise} \end{cases},$$

being the denominator used for normalisation.

Purposely, I am inverting the polarities hereafter so that $-\frac{s_i \cdot s_{i+1}}{|s_i \cdot s_{i+1}|}$ becomes either 1 or −1, respectively, in response to the presence or absence of a zero-crossing. Furthermore, despite the fact that $\frac{1}{|s_i \cdot s_{i+1}|}$ is the simplest existing normalisation, I am going to replace it by a more convenient formulation to reach my objective: the sigmoide function parametrised with a slope $\gamma >> 0$, as shown in Fig. 11. We therefore have

$$\frac{1}{1+e^{-\gamma(-s_i \cdot s_{i+1})}} = \begin{cases} 1 & \text{if there is a zero-crossing between} \\ & \quad s_i \text{ and } s_{i+1} \\ 0 & \text{otherwise} \end{cases}.$$

In order to traverse a window of length L and count its ZCRs, the summation $\sum_{i=0}^{L-2} \frac{1}{1+e^{-\gamma(-s_i \cdot s_{i+1})}}$ is adopted. The readers may have learnt that, for FE, we are interested in the **normalised** number of ZCRs instead of its raw amount. Thus,

$$ZCR(s[\cdot]) = \frac{1}{\beta} \cdot \sum_{i=0}^{L-2} \frac{1}{1+e^{-\gamma(-s_i \cdot s_{i+1})}} = \sum_{i=0}^{L-2} \frac{1}{\beta} \cdot \frac{1}{1+e^{-\gamma(-s_i \cdot s_{i+1})}}$$

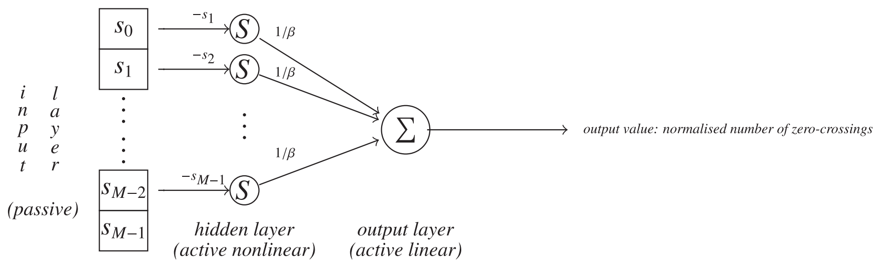
is the simplest possibility to obtain a bounded outcome within the range from 0 to 1. Particularly, TA, PA and EA can be addressed as a function of β, respectively, by letting it be equal to $\sum_{k=0}^{T-1} ZCR(w_k[\cdot])$, $L-1$ and $ZCR(w_h[\cdot])$, as I defined previously.

Clearly, the structure I propose corresponds to the original multilayer perceptron defined by Frank Rosenblatt [34], as shown in Fig. 12, with some peculiarities. Its ith input, ith weight between the input and the hidden layers, and ith weight between the hidden and the output layers are, respectively, s_i, $-s_{i+1}$ and $\frac{1}{\beta}$. Moreover, the ith neuron of the input layer connects forward only with the ith of the hidden one. Another possible interpretation for the proposed structure is that of a weightless neural network, also known as random access memory (RAM) network [35–37], so that there are weights albeit pre-defined, implying that there is no learning procedure.

Concluding, when we are counting the normalised ZCRs of a certain signal, we are somehow neurocomputing it, moreover, on the basis of neurons which were “born with a pre-established knowledge”. The potential of ZCRs awakens deeper attraction upon their characterisation as being specific neurocomputing agents, thus, my expectation is that the interdisciplinary community interested in PRKbS, FE, computational intelligence, artificial neural networks, DSP and related fields will frequently take advantage of the methods I present.

3. Numerical examples

In order to shed some light on the proposed approaches, one numerical example follows for each case: methods B_1, B_2 and B_3,



s_0
s_1
s_{M-2}
s_{M-1}
input layer
(passive)
$-s_1$
$-s_2$
$-s_{M-1}$
S
$1/\beta$
hidden layer (active nonlinear)
Σ
output layer (active linear)
output value: normalised number of zero-crossings

Fig. 12. The proposed structure, with pre-defined weights.

Algorithm 6 : fragment of C++ code for method B_1 in 2D, adopting the normalisation EA.

```
// ensure that m[·][·], with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean+ = m[p][q]/(double)(M*N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L = /* the desired positive value, not higher than the higher between M and N */;
int V = /* the desired positive value, lower than 100 */;
int T = (int)((100*M − L*V)/((100 − V)*L));
int P = (int)((100*N − L*V)/((100 − V)*L));
int highest_ZCR = 0; // ZCR is the total number of zero-crossings over all the window placements, required for normalisation
double *f = new double[T*P]; // dynamic vector declaration
for(int k = 0; k < T*P; k++)
        {
        f[k] = 0;
         for(int i = k*((int)(((100 − V)/100.0)*L)); i < k*((int)(((100 − V)/100.0)*L)) + L; i++)
                   for(int j = k*((int)(((100 − V)/100.0))*L); j < k*((int)(((100 − V)/100.0))*L) + L − 1; j++)
                   f[k]+ = (m[i][j]*m[i][j + 1] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
         for(int i = k*((int)(((100 − V)/100.0)*L)); i < k*((int)(((100 − V)/100.0)*L)) + L − 1; i++)
                   for(int j = k*((int)(((100 − V)/100.0))*L); j < k*((int)(((100 − V)/100.0))*L) + L; j++)
                   f[k]+ = (m[i][j]*m[i + 1][j] < 0)?1:0; /* multiplying subsequent samples results in a negative value if they are between 0. This is equivalent to the theoretical procedure described in the text and based on equation 1. */
        if (f[k] > highest_ZCR)
                highest_ZCR = f[k];
        }
for(int k = 0; k < T*P; k++)
        f[k]/ = (double)(highest_ZCR); the casting, i.e., the explicit conversion of highest_ZCR from int to double is, theoretically, not required, however, some C/C++ compilers have presented problems when a double-precision variable is divided by an int one, resulting in 0. To avoid this issue, the casting is used. */
// at this point, the feature vector, f[·], is ready
```

both in 1D and 2D, assuming the normalisations previously described and based on hypothetical data.

3.1. Numerical example for B$_1$ in 1D

Problem statement: Let $s[\cdot] = \{1, 2, 3, 4, 5, 5, 4, 3, 2, 1\}$, implying in $M = 10$, and $L = 4$ be the window length, with overlaps of $V = 50\%$. Obtain the feature vector, $f[\cdot]$, according to the method B_1.

Solution: First, the 1D signal mean, $\frac{1+2+3+4+5+5+4+3+2+1}{10} = \frac{30}{10} = 3 \neq 0$, is subtracted from each component of $s[\cdot]$, resulting in $\{1-3, 2-3, 3-3, 4-3, 5-3, 5-3, 4-3, 3-3, 2-3, 1-3\} = \{-2, -1, 0, 1, 2, 2, 1, 0, -1, -2\}$. The corresponding feature

Algorithm 7 : fragment of C++ code for method B_2 in 1D, adopting the normalisation TA.

```
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L; // window length
int ZCR; // ZCR represents the total ZCR over all the window positions, that is required to normalise f[·]
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of interest. It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in X[·]
        total_size_of_f+=X[i];
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum of the elements in X[·], i.e., the size of the subvector ξ1[·] plus the size of the subvector ξ2[·], plus the size of the subvector ξ3[·], ..., and so on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int j = 0; j <(int)(sizeof(X)/sizeof(int)) ; j++)
        {
        ZCR = 0;
        for(int k = 0; k < X[j]; k++)
                {
                L = (int)(M/X[j]);
                f[jump + k] = 0;
                for(int i = (k*L); i < (k*L) + L; i++)
                        f[jump + k]+ = (s[i]*s[i + 1] < 0)?1:0;
                ZCR+ = f[jump + k];
                }
        for(int k = 0; k < X[j]; k++)
                f[jump + k]/ = (double)(ZCR);
         jump+ = X[j];
        }
// at this point, the feature vector, f[·], is ready
```

vector, which has length $T = \lfloor\frac{(100 \cdot M)-(L \cdot V)}{(100-V) \cdot L}\rfloor = \lfloor\frac{(100 \cdot 10)-(4 \cdot 50)}{(100-50) \cdot 4}\rfloor = 4$, is obtained as follows:

- $w_0[\ \cdot\ ]$, which covers the sub-signal $\{-2, -1, 0, 1\}$, contains 1 zero-crossing, implying that $f_0 = 1$;
- $w_1[\ \cdot\ ]$, which covers the sub-signal {0, 1, 2, 2}, contains no zero-crossings, implying that $f_B = 0$;
- $w_2[\ \cdot\ ]$, which covers the sub-signal {2, 2, 1, 0}, contains no zero-crossings, implying that $f_A = 0$;
- $w_3[\ \cdot\ ]$, which covers the sub-signal $\{1, 0, -1, -2\}$, contains 1 zero-crossing, implying that $f_3 = 1$.

For the normalisation TA, each component of $f[\cdot]$ is divided by $\sum_{k=0}^{3} f_k = 1 + 0 + 0 + 1 = 2$. Thus, it becomes $\{\frac{1}{2}, \frac{0}{2}, \frac{0}{2}, \frac{1}{2}\} = \{\frac{1}{2}, 0, 0, \frac{1}{2}\}$. On the other hand, for PA, each component of $f[\cdot]$ is divided by the maximum number of zero-crossings, i.e., $L - 1 = 3$. Thus, it becomes $\{\frac{1}{3}, \frac{0}{3}, \frac{0}{3}, \frac{1}{3}\} = \{\frac{1}{3}, 0, 0, \frac{1}{3}\}$. Lastly, for EA, each component of $f[\cdot]$ is divided by the highest component of its unnormalised version, i.e., 1. Thus, it becomes $\{\frac{1}{1}, \frac{0}{1}, \frac{0}{1}, \frac{1}{1}\} = \{1, 0, 0, 1\}$.

Algorithm 8 : fragment of C++ code for method B_2 in 1D, adopting the normalisation PA.

```
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L; // window length
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of interest.
It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in X[·]
        total_size_of_f+=X[i];
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum of the
elements in X[·], i.e., the size of the subvector ξ1[·] plus the size of the subvector
ξ2[·], plus the size of the subvector ξ3[·], ..., and so on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int j = 0; j <(int)(sizeof(X)/sizeof(int)) ; j++)
        {
        for(int k = 0; k < X[j]; k++)
                {
                L = (int)(M/X[j]);
                f[jump + k] = 0;
                for(int i = (k*L); i < (k*L) + L; i++)
                        f[jump + k]+ = (s[i]*s[i + 1] < 0)?1:0;
                }
        for(int k = 0; k < X[j]; k++)
                f[jump + k]/ = (double)(L − 1);
        jump+ = X[j];
        }
// at this point, the feature vector, f[·], is ready
```

Algorithm 9 : fragment of C++ code for method B_2 in 1D, adopting the normalisation EA.

```
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal
is 0
int L; // window length
int highest_ZCR; // E represents the total ZCR over all the window positions,
that is required to normalise f[·]
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of
interest. It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in
X[·]
        total_size_of_f+=X[i];
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum
of the elements in X[·], i.e., the size of the subvector ξ1[·] plus the size of the
subvector ξ2[·], plus the size of the subvector ξ3[·], ..., and so on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int j = 0; j <(int)(sizeof(X)/sizeof(int)) ; j++)
        {
        highest_ZCR = 0;
        for(int k = 0; k < X[j]; k++)
                {
                L = (int)(M/X[j]);
                f[jump + k] = 0;
                for(int i = (k*L); i < (k*L) + L; i++)
                        f[jump + k]+ = (s[i]*s[i + 1] < 0)?1:0;
                if (f[jump + k] > highest_ZCR)
                        highest_ZCR = f[jump + k];
                }
        for(int k = 0; k < X[j]; k++)
                f[jump + k]/ = (double)(highest_ZCR);
        jump+ = X[j];
        }
// at this point, the feature vector, f[·], is ready
```

3.2. Numerical example for B_1 in 2D

Problem statement: Let $m[\cdot][\cdot] = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 4 & 6 \\ 7 & 8 & 9 & 10 \end{pmatrix}$, implying in $N = 3$ and $M = 4$. Assume that the square window has size $L = 2$ with overlaps of $V = 50\%$. Obtain the feature vector, $f[\cdot]$, following method B_1.

Solution: First, the 2D signal mean, $\frac{1+2+3+4+4+2+4+6+7+8+9+10}{12} = \frac{60}{12} = 5 \neq 0$, is subtracted from each component of $s[\cdot]$, resulting in $\begin{pmatrix} 1-5 & 2-5 & 3-5 & 4-5 \\ 4-5 & 2-5 & 4-5 & 6-5 \\ 7-5 & 8-5 & 9-5 & 10-5 \end{pmatrix} = \begin{pmatrix} -4 & -3 & -2 & -1 \\ -1 & -3 & -1 & 1 \\ 2 & 3 & 4 & 5 \end{pmatrix}$. Then, the feature vector with length $T \cdot P = \lfloor \frac{(100 \cdot M)-(L \cdot V)}{(100-V) \cdot L} \rfloor \cdot \lfloor \frac{(100 \cdot N)-(L \cdot V)}{(100-V) \cdot L} \rfloor = \lfloor \frac{(100 \cdot 4)-(2 \cdot 50)}{(100-50) \cdot 2} \rfloor \cdot \lfloor \frac{(100 \cdot 3)-(2 \cdot 50)}{(100-50) \cdot 2} \rfloor = 3 \cdot 2 = 6$ is obtained as follows:

- $w_0[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -4 & -3 \\ -1 & -3 \end{pmatrix}$, which contains no zero-crossings, implying that $f_0 = 0$;
- $w_1[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -3 & -3 \\ -2 & -1 \end{pmatrix}$, which contains no zero-crossings, implying that $f_B = 0$;
- $w_2[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -2 & -1 \\ -1 & 1 \end{pmatrix}$, which contains 2 zero-crossings, implying that $f_A = 2$;
- $w_3[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -1 & -3 \\ 2 & 3 \end{pmatrix}$, which contains 2 zero-crossings, implying that $f_3 = 2$;
- $w_4[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -3 & -1 \\ 3 & 4 \end{pmatrix}$, which contains 2 zero-crossings, implying that $f_4 = 2$;
- $w_5[\cdot][\cdot]$ covers the sub-matrix $\begin{pmatrix} -1 & 1 \\ 4 & 5 \end{pmatrix}$, which contains 2 zero-crossings, implying that $f_5 = 2$.

For the normalisation TA, each component of $f[\cdot]$ is divided by $\sum_{k=0}^{5} f_k = 0+0+2+2+2+2 = 8$. Thus, it becomes $\{\frac{0}{8}, \frac{0}{8}, \frac{2}{8}, \frac{2}{8}, \frac{2}{8}, \frac{2}{8}\} = \{0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$. On the other hand, for PA, each component of $f[\cdot]$ is divided by the maximum number of zero-crossings, i.e., $(4-1) \cdot 3 + (3-1) \cdot 4 = 9 + 8 = 17$. Thus, it becomes $\{\frac{0}{17}, \frac{0}{17}, \frac{2}{17}, \frac{2}{17}, \frac{2}{17}, \frac{2}{17}\}$. Lastly, for EA, each component of $f[\cdot]$ is divided by the highest component of its unnormalised version, i.e., 2. Thus, it becomes $\{\frac{0}{2}, \frac{0}{2}, \frac{2}{2}, \frac{2}{2}, \frac{2}{2}, \frac{2}{2}\} = \{0, 0, 1, 1, 1, 1\}$.

3.3. Numerical example for B_2 in 1D

Problem statement: Let $s[\cdot] = \{1, 2, 4, 6, 6, 6, 6, 5, 3, 1\}$, implying in $M = 10$. Assuming that $Q = 3$, with no overlaps between window positions, obtain the feature vector, $f[\cdot]$, following the method B_2.

Solution: First, the 1D signal mean, $\frac{1+2+4+6+6+6+6+5+3+1}{10} = \frac{40}{10} = 4 \neq 0$, is subtracted from each component of $s[\cdot]$, resulting in $\{1-4, 2-4, 4-4, 6-4, 6-4, 6-4, 6-4, 5-4, 3-4, 1-4\} = \{-3, -2, 0, 2, 2, 2, 2, 1, -1, -3\}$. The feature vector is composed by the concatenation of $Q = 3$ sub-vectors, i.e., $f[\cdot] = \{\xi_1[\cdot]\} \cup \{\xi_2[\cdot]\} \cup \{\xi_3[\cdot]\}$, which are obtained as follows:

- The first subvector, $\xi_1[\cdot]$, comes from two non-overlapping windows, $w_0[\cdot] = \{-3, -2, 0, 2, 2\}$ and $w_1[\cdot] = \{2, 2, 1, -1, -3\}$, which are positioned over $s[\cdot]$. The corresponding results are:
  $\xi_{10} = 1 \quad ; \quad \xi_{11} = 1$.
- The second subvector, $\xi_2[\cdot]$, comes from three non-overlapping windows, $w_0[\cdot] = \{-3, -2, 0\}$, $w_1[\cdot] = \{2, 2, 2\}$ and $w_2[\cdot] = \{2, 1, -1\}$, which are positioned over $s[\cdot]$, discarding its last element, i.e., the amplitude -3. The corresponding results are:

Algorithm 10 : fragment of C++ code for method B_2 in 2D, adopting the normalisation TA.

```
// ensure that $m[\cdot][\cdot]$, with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean+ = m[p][q]/(double)(M*N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q]− = mean; //at this point, the arithmetic mean of the input signal is 0
double ZCR; // represents the total ZCR over all the window positions, that is required to normalise f[·]
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of interest. It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in X[·]
        total_size_of_f+=pow(X[i], 2);
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum of the squares of the elements in X[·], i.e., the size of the subvector ξ1[·] plus the size of the subvector ξ2[·], plus the size of the subvector ξ3[·], ..., and so on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int i = 0;i <total_size_of_f;i++)
        f[i] = 0;
int L1, L2;
for(int k = 0;k <(int)(sizeof(X)/sizeof(int));k++)
        {
        ZCR = 0;
        L1 = (int)(N/X[k]);
        L2 = (int)(M/X[k]);
        for(int i = 0;i <((int)(N/X[k]))*X[k] - 1;i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k];j++)
                        {
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] * m[i+1][j] < 0)?1:0;
                        ZCR+ = (m[i][j] * m[i+1][j] < 0)?1 : 0;
                        }
        for(int i = 0;i <((int)(N/X[k]))*X[k];i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k] - 1;j++)
                        {
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] * m[i][j+1] < 0)?1:0;
                        ZCR+ = (m[i][j] * m[i][j+1] < 0)?1 : 0;
                        }
        for(int i =jump;i <jump+pow(X[k],2);i++)
                f[i]/ = (double)(ZCR);
        jump+=pow(X[k],2);
        }
// at this point, the feature vector, f[·], is ready.
```

$\xi_{20} = 1$; $\xi_{21} = 0$; $\xi_{22} = 1$.

- The third subvector, $\xi_3[\cdot]$, comes from five non-overlapping windows, $w_0[\cdot] = \{-3, -2\}$, $w_1[\cdot] = \{0, 2\}$, $w_2[\cdot] = \{2, 2\}$, $w_3[\cdot] = \{2, 1\}$ and $w_4[\cdot] = \{-1, -3\}$, which are positioned over $s[\cdot]$. The corresponding results are:
  $\xi_{30} = 0$; $\xi_{31} = 0$; $\xi_{32} = 0$; $\xi_{33} = 0$; $\xi_{34} = 0$.

The concatenation of the three sub-vectors produce $f[\cdot] = \{1, 1, 1, 0, 1, 0, 0, 0, 0, 0\}$. Now, each sub-vector is normalised separately. Considering TA, each component of ξ_1 in $f[\cdot]$ is divided by $\sum_{k=0}^{1} \xi_{1k} = 1 + 1 = 2$; each component of ξ_2 in $f[\cdot]$ is divided by $\sum_{k=0}^{2} \xi_{2k} = 1 + 0 + 1 = 2$; and each component of ξ_3 in $f[\cdot]$ keeps unchangeable because $\sum_{k=0}^{4} \xi_{3k} = 0$. Thus, $f[\cdot]$ becomes $\{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{0}{2}, \frac{1}{2}, 0, 0, 0, 0, 0\} = \{\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, 0, 0, 0\}$.

On the other hand, considering PA, each component of ξ_1 in $f[\cdot]$ is divided by the maximum number of zero-crossings possible for the window that originated it, i.e., $L - 1 = 5 - 1 = 4$. Equally, each component of ξ_2 in $f[\cdot]$ is divided by $L - 1 = 3 - 1 = 2$, and each component of ξ_3 in $f[\cdot]$ is divided by $L - 1 = 2 - 1 = 1$. Thus, $f[\cdot]$ becomes $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{0}{2}, \frac{1}{2}, \frac{0}{1}, \frac{0}{1}, \frac{0}{1}, \frac{0}{1}, \frac{0}{1}\} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, 0, 0, 0\}$.

Algorithm 11 : fragment of C++ code for method B_2 in 2D, adopting the normalisation PA.

```
// ensure that $m[\cdot][\cdot]$, with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean+ = m[p][q]/(double)(M*N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q]− = mean; //at this point, the arithmetic mean of the input signal is 0
int L1, L2;
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of interest. It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in X[·]
        total_size_of_f+=pow(X[i], 2);
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum of the squares of the elements in X[·], i.e., the size of the subvector ξ1[·] plus the size of the subvector ξ2[·], plus the size of the subvector ξ3[·], ..., and so on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int i = 0;i <total_size_of_f;i++)
        f[i] = 0;
for(int k = 0;k <(int)(sizeof(X)/sizeof(int));k++)
        {
        L1 =(int)(M/X[k]);
        L2 =(int)(N/X[k]);
        for(int i = 0;i <((int)(N/X[k]))*X[k] - 1;i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k];j++)
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] * m[i+1][j] < 0)?1:0;
        for(int i = 0;i <((int)(N/X[k]))*X[k];i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k] - 1;j++)
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] * m[i][j+1] < 0)?1:0;
        for(int i =jump;i <jump+pow(X[k],2);i++)
                f[i]/ = (2*L1*L2 − L1 − L2);
        jump+=pow(X[k],2);
        }
// at this point, the feature vector, f[·], is ready.
// . . .
```

Lastly, considering EA, each component of ξ_1 in $f[\cdot]$ is divided by the highest component in it, i.e., 1; each component of ξ_2 in $f[\cdot]$ is divided by the highest component in it, i.e., 1; and each component of ξ_3 in $f[\cdot]$ keeps unchangeable because its highest component is 0. Thus, $f[\cdot]$ becomes $\{\frac{1}{1}, \frac{1}{1}, \frac{1}{1}, \frac{0}{1}, \frac{1}{1}, 0, 0, 0, 0, 0\} = \{1, 1, 1, 0, 1, 0, 0, 0, 0, 0\}$.

3.4. Numerical example for B$_2$ in 2D

Problem statement: Let $m[\cdot][\cdot] = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 53 \\ 3 & 0 & 1 & 0 \end{pmatrix}$, implying in $N = 4$ and $M = 4$. Assume that $Q = 2$ with no overlaps between windows. Obtain the feature vector, $f[\cdot]$, following method B_2.

Solution: First, the 2D signal mean, $\frac{0+1+2+3+4+5+6+7+8+9+10+53+3+0+1+0}{16} = \frac{112}{16} = 7 \neq 0$, is subtracted from each component of $m[\cdot][\cdot]$, resulting in $\begin{pmatrix} 0-7 & 1-7 & 2-7 & 3-7 \\ 4-7 & 5-7 & 6-7 & 7-7 \\ 8-7 & 9-7 & 10-7 & 53-7 \\ 3-7 & 0-7 & 1-7 & 0-7 \end{pmatrix} = \begin{pmatrix} -7 & -6 & -5 & -4 \\ -3 & -2 & -1 & 0 \\ 1 & 2 & 3 & 46 \\ -4 & -7 & -6 & -7 \end{pmatrix}$. The feature vector is composed by the concatenation of $Q = 2$ sub-vectors, i.e., $f[\cdot] = \{\xi_1[\cdot]\} \cup \{\xi_2[\cdot]\}$. They are obtained as follows:

- for $\xi_1[\cdot]$, a total of $2 \cdot 2 = 4$ non-overlapping windows, $w_0[\cdot][\cdot] = \begin{pmatrix} -7 & -6 \\ -3 & -2 \end{pmatrix}$, $w_1[\cdot][\cdot] = \begin{pmatrix} -5 & -4 \\ -1 & 0 \end{pmatrix}$, $w_2[\cdot][\cdot] = \begin{pmatrix} 1 & 2 \\ -4 & -7 \end{pmatrix}$ and $w_3[\cdot][\cdot] = \begin{pmatrix} 3 & 46 \\ -6 & -7 \end{pmatrix}$, are positioned over $m[\cdot][\cdot]$. The result is:
  $\xi_{10} = 0$; $\xi_{11} = 2$; $\xi_{13} = 2$; $\xi_{14} = 2$.

Algorithm 12 : fragment of C++ code for method B_2 in 2D, adopting the normalisation EA.

```
// ensure that m[·][·], with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean+ = m[p][q]/(double)(M * N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q]− = mean; //at this point, the arithmetic mean of the
input signal is 0
int X[] = {2, 3, 5, 7, 9, 11, 13, 17}; /* vector containing the prime numbers of
interest. It can be changed according to the experiment */
int total_size_of_f = 0;
for(int i = 0; i <(int)(sizeof(X)/sizeof(int));i++) // number of elements in
X[·]
        total_size_of_f+=pow(X[i], 2);
double *f = new double[total_size_of_f]; /* The total size of f[·] is the sum
of the squares of the elements in X[·], i.e., the size of the subvector ξ1[·] plus
the size of the subvector ξ2[·], plus the size of the subvector ξ3[·], ..., and so
on */
int jump = 0; // helps to control the correct positions to write in f[·]
for(int i = 0;i <total_size_of_f;i++)
        f[i] = 0;
int L1, L2;
for(int k = 0;k <(int)(sizeof(X)/sizeof(int));k++)
        {
        L1 = (int)(N/X[k]);
        L2 = (int)(M/X[k]);
        for(int i = 0;i <((int)(N/X[k]))*X[k] - 1;i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k];j++)
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] *
m[i+1][j] < 0)?1:0;
        for(int i = 0;i <((int)(N/X[k]))*X[k];i++)
                for(int j = 0;j <((int)(M/X[k]))*X[k] - 1;j++)
                        f[jump+(((int)(i/L2))*(X[k]))+((int)(j/L1))]+ =(m[i][j] *
m[i][j+1] < 0)?1:0;
        jump+=pow(X[k],2);
        }
jump=0;
double highest_ZCR;
for(int k = 0;k <(int)(sizeof(X)/sizeof(int));k++)
        {
        highest_ZCR = 0;
        for(int l = 0;l <pow(X[k], 2);l++)
                if(f[jump+l]> highest_ZCR)
                        highest_ZCR = f[jump+l];
        for(int l = 0;l <pow(X[k], 2);l++)
                f[jump+l]/ = highest_ZCR;
        jump+=pow(X[k], 2);
        }
// at this point, the feature vector, f[·], is ready.
```

Algorithm 13 : fragment of C++ code for method B_3 in 1D.

```
// ensure that s[·], of length M, is available as input
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal
is 0
int L = 0; // partial lengths
double C =; // the desired value, being 0 < C < 100
int T = ((100/C) − ((int)(100/C)) == 0)?(100/C) − 1 : (int)(100/C) ; //
the number of elements in T
double *f = new double[T]; // dynamic vector declaration
double z = zcr(&s[0], M)*((double)(C)/100);
for(int k = 0; k < T; k++)
        {
        while(zcr(&s[0], L)<((k+1) * z))
                L++;
        f[k] =(double)(L)/(double)(M);
        }
// at this point, the feature vector, f[·], is ready
// . . .
// function zcr
double zcr(double* input_vector, int length)
{
double z = 0;
for(int i=0;i<length - 1;i++)
        z +=(input_vector[i] * input_vector[i+1] < 0)?1:0;
return(z);
}
```

Algorithm 14 : fragment of C++ code for method B_3 in 2D.

```
// ...
// ensure that m[·][·], with height N and width M, is available as input
double mean = 0;
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                mean+ = m[p][q]/(double)(M * N);
for(int p = 0; p < N; p++)
        for(int q = 0; q < M; q++)
                m[p][q]− = mean; //at this point, the arithmetic mean of the
input signal is 0
double alpha = 0; // partial height
double beta = 0; // partial width
double C =; // the desired value, being 0 < C < 100
int T = ((100/C) − ((int)(100/C)) == 0)?(100/C) − 1 : (int)(100/C) ; //
the number of elements in T
double *f = new double[T]; // dynamic vector declaration
double z = zcr(m, N, M)*((double)(C)/100.0);
for(int k = 0; k < T; k++)
        {
        while(zcr(&m[0][0],(int)(alpha),(int)(beta))<((k+1) * z))
                if (N == M)
                        {
                        alpha++;
                        beta++;
                        }
                else if (N > M)
                        {
                        alpha++;
                        beta+=(double)(M)/(double)(N);
                        }
                else
                        {
                        alpha+=(double)(N)/(double)(M);
                        beta++;
                        }
        f[k] =((alpha * beta)<=(N * M))?((alpha * beta)/(N * M)):(1); // ex-
ceptionally, as alpha or beta increases, f[k] > 1 for k close to T, so this is
a correction.
        }
// at this point, the feature vector, f[·], is ready
// . . .
// function zcr in 2D
double zcr(double** input_matrix, int height, int width)
{
double z = 0;
for(int i=0;i<height - 1;i++)
        for(int j=0;j<width;j++)
                z +=(input_matrix[i][j] * input_matrix[i+1][j] < 0)?1:0;
for(int i=0;i<height;i++)
        for(int j=0;j<width - 1;j++)
                z +=(input_matrix[i][j] * input_matrix[i][j+1] < 0)?1:0;
return(z);
}
// ...
```

- for $\xi_2[\cdot]$, a total of $3 \cdot 3 = 9$ non-overlapping windows, $w_0[\cdot][\cdot] = (-7)$, $w_1[\cdot][\cdot] = (-6)$, $w_2[\cdot][\cdot] = (-5)$, $w_3[\cdot][\cdot] = (-3)$, $w_4[\cdot][\cdot] = (-2)$, $w_5[\cdot][\cdot] = (-1)$, $w_6[\cdot][\cdot] = (1)$, $w_7[\cdot][\cdot] = (2)$ and $w_8[\cdot][\cdot] = (3)$, are positioned over $m[\cdot][\cdot]$, being its fourth row and fourth column discarded. The result is: $\xi_{20} = 0$; $\xi_{21} = 0$; $\xi_{22} = 0$; $\xi_{23} = 0$; $\xi_{24} = 0$; $\xi_{25} = 0$; $\xi_{26} = 0$; $\xi_{27} = 0$; $\xi_{28} = 0$.

The concatenation of both sub-vectors produce $f[\cdot] = \{0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$. As in the previous example, each sub-vector is normalised separately. Considering TA, each component of ξ_1 in $f[\cdot]$ is divided by $\sum_{k=0}^{3} \xi_{1k} = 0 + 2 + 2 + 2 = 6$; and each component of ξ_2 in $f[\cdot]$ keeps unchangeable because $\sum_{k=0}^{8} \xi_{2k} = 0$. Thus, $f[\cdot]$ becomes $\{\frac{0}{6}, \frac{2}{6}, \frac{2}{6}, \frac{2}{6}, 0, 0, 0, 0, 0, 0, 0, 0, 0\} = \{0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$.

On the other hand, considering PA, each component of ξ_1 in $f[\cdot]$ is divided by the maximum number of zero-crossings possible for the window that originated it, i.e., $1 \cdot 2 + 1 \cdot 2 = 4$, and each component of ξ_2 in $f[\cdot]$ keeps unchangeable because the subvector does not cross zero. Thus, $f[\cdot]$ becomes $\{\frac{0}{4}, \frac{2}{4}, \frac{2}{4}, \frac{2}{4}, 0, 0, 0, 0, 0, 0, 0, 0, 0\} = \{0, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$.

Lastly, considering EA, each component of ξ_1 in $f[\cdot]$ is divided by the highest component in it, i.e., 2; and each component of ξ_2 in $f[\cdot]$ keeps unchangeable because its highest component is 0. Thus, $f[\cdot]$ becomes $\{\frac{0}{2}, \frac{2}{2}, \frac{2}{2}, \frac{2}{2}, 0, 0, 0, 0, 0, 0, 0, 0, 0\} = \{0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$.

3.5. Numerical example for B$_3$ in 1D

Problem statement: Let $s[\cdot] = \{1, -1, 1, -1, 1, 1, -1, -1\}$, implying in $M = 8$, and assume that $C = 20\%$ is the critical level. Obtain the feature vector, $f[\cdot]$, according to the method B_3.

Solution: The signal mean is $1 + (-1) + 1 + (-1) + 1 + 1 + (-1) + (-1) = 0$, i.e., the ZCRs are ready to be counted. The feature vector, of length $T = \frac{100}{20} - 1 = 4$, is composed by the proportional lengths of $s[\cdot]$ required to reach $1 \cdot 20\% = 20\%$, $2 \cdot 20\% = 40\%$, $3 \cdot 20\% = 60\%$ and $4 \cdot 20\% = 80\%$ of its total number of zero-crossings. They are obtained as follows:

- $s[\cdot]$ has 5 zero-crossings;
- 20% of 5 is $0.2 \cdot 5 = 1.4$. Ceiling the result, 2 zero-crossings are required. The proportion of the length of $s[\cdot]$ to reach them, from the beginning, is $\frac{3}{8}$;
- 40% of 5 is $0.4 \cdot 5 = 2$ zero-crossings. The proportion of the length of $s[\cdot]$ to reach them, from the beginning, is, again, $\frac{3}{8}$;
- 60% of 5 is $0.6 \cdot 5 = 3$. The proportion of the length of $s[\cdot]$ to reach them, from the beginning, is $\frac{4}{8} = \frac{1}{2}$;
- 80% of 5 is $0.8 \cdot 5 = 4$. The proportion of the length of $s[\cdot]$ to reach them, from the beginning, is $\frac{5}{8}$.

Thus, $f[\cdot] = \{\frac{3}{8}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}\}$. As explained in the previous section, no further normalisation applies.

3.6. Numerical example for B$_3$ in 2D

Problem statement: Let $m[\cdot][\cdot] = \begin{pmatrix} 5 & 2 & 5 & 6 \\ 2 & 3 & 3 & 7 \\ 8 & 4 & -4 & 6 \\ 1 & 4 & 12 & 0 \end{pmatrix}$, implying in $N = 4$ and $M = 4$. Assume that $C = 25\%$ is the critical level. Obtain the feature vector, $f[\cdot]$, according to the method B_3.

Solution: The signal mean is $\frac{5+2+5+6+2+3+3+7+8+4-4+6+1+4+12+0}{16} = \frac{64}{16} = 4 \neq 0$. Thus, $m[\cdot][\cdot]$ is translated so that it becomes $\begin{pmatrix} 5-4 & 2-4 & 5-4 & 6-4 \\ 2-4 & 3-4 & 3-4 & 7-4 \\ 8-4 & 4-4 & -4-4 & 6-4 \\ 1-4 & 4-4 & 12-4 & 0-4 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 1 & 2 \\ -2 & -1 & -1 & 3 \\ 4 & 0 & -8 & 2 \\ -3 & 0 & 8 & -4 \end{pmatrix}$. The feature vector, of length $T = \frac{100}{25} - 1 = 3$, is composed of the proportional areas of $m[\cdot][\cdot]$ required to reach $1 \cdot 25\% = 25\%$, $2 \cdot 25\% = 50\%$ and $3 \cdot 25\% = 75\%$ of its total ZCR. They are obtained as follows:

- $m[\cdot][\cdot]$ has 13 zero-crossings. Its area is $N \cdot M = 4 \cdot 4 = 16$;
- 25% of 13 is $0.25 \cdot 13 = 3.25$. Ceiling it, 4 zero-crossings are considered. Thus, the sub-matrix covered by the $(N = 3)\ x\ (M = 3)$ rectangle, from $m_{0,0}$, are required to reach at least 4 zero-crossings. Since $N \cdot M = 3 \cdot 3 = 9$, the proportion of $m[\cdot][\cdot]$ area covered is $\frac{9}{16} = 0.5625$;
- 50% of 13 is $0.5 \cdot 13 = 6.5$. Ceiling it, 7 zero-crossings are considered. Thus, the sub-matrix covered by the $(N = 4)\ x\ (M = 4)$ rectangle, from $m_{0,0}$, are required to reach at least 7 zero-crossings. Since $N \cdot M = 4 \cdot 4 = 16$, the proportion of $m[\cdot][\cdot]$ area covered is $\frac{16}{16} = 1$;
- 75% of 13 is $0.75 \cdot 13 = 9.75$. Ceiling it, 10 zero-crossings are considered. Thus, the sub-matrix covered by the $(N = 4)\ x\ (M = 4)$ rectangle, from $m_{0,0}$, are required to reach at least 10 zero-crossings. Since $N \cdot M = 4 \cdot 4 = 16$, the proportion of $m[\cdot][\cdot]$ area covered is $\frac{16}{16} = 1$;

Thus, $f[\cdot] = \{0.5625, 1, 1\}$. As in the previous example, no further normalisation applies.

4. Example applications

In this section, example applications involving 1D and 2D real-life data are shown to consolidate the proposed approaches, demonstrating their usability.

4.1. Speech classification and segmentation

There are many subclassifications for speech data, however, *voiced, unvoiced* and *silent*, respectively originated from quasi-periodic, non-periodic and inactive sources, are the root ones [38]-pp.77, 78. Usual applications in which the differentiation between voiced, unvoiced and silent segments (VUSS) is relevant include large-vocabulary speech recognition [39], speaker identification [40], voice conversion [41] and speech coding [42]. Thus, I dedicate this section to initially present a ZCR-based algorithm for the distinction among VUSS and, upon taking advantage of that formulation, to introduce my proposal for isolated-sentence word segmentation.

Neither B_2 nor B_3 can be used in this experiment, because, a priori, there is an unknown number of VUSS, implying that the feature vector generated to store their positions has a variable length that depends not only on the duration but also on the content of the spoken words. Thus, B_1 is the only adequate method, among the three ones I presented, to carry out this task. Complementarily, VUSS are classified according to the characteristics of each speech frame, in isolation, i.e., disregarding the remaining of the signal, suggesting that PA is the proper normalisation.

As reported in [43], ZCRs are usually associated with signal energy [1] to allow the implementation of accurate algorithms for the detection of VUSS. Thus, B_1 normalised with PA was associated with A_1, fully described in [1]. Independently, B_1 and A_1 were applied to the input speech signal $s[\cdot]$, respectively, producing the feature vectors I named as $f_B[\cdot]$ and as $f_A[\cdot]$, with the same variable length. Consequently, for each window placement, there are two types of information: spectral, based on the normalised ZCRs available in $f_B[\cdot]$, and temporal, based on the normalised energies contained in $f_A[\cdot]$. Considered in conjunction, the feature sets produce a variable-length description of $s[\cdot]$ so that there is a correspondence between this signal and those vectors for each transition between different types of segments. Notably, the frontiers which delimit VUSS could not be easily found with basis on the direct inspection of $s[\cdot]$.

Fig. 13 shows the input signal, $s[\cdot]$, that I use to explain the proposed approach. It was digitalised at 16,000 samples per second, 16-bit, corresponding to the 45,520 sample-long raw speech data extracted from the sentence *sa1* contained in the directory */test/dr1/mdab0/* of the TIMIT speech corpus [44], which reads as "*She had your dark suit in greasy wash water all year*". In the figure, the horizontal axis contains the tags J_k, $(0 \le k \le 37)$, which correspond to the transitions between consecutive phonemes of *sa1*, as described in the respective *phn* file included in the above-mentioned directory. In addition to the silent periods, labelled as SIL and indicated in orange, the exact voiced and unvoiced intervals, respectively designated as VOI and UNV, are shown in teal and violet, in accordance with the documentation found in [44] and [45].

The value I chose for L, equivalent to 32 ms of speech, is of 512 samples. As documented in [12]-pp.32 and [46]-pp.25, the speech

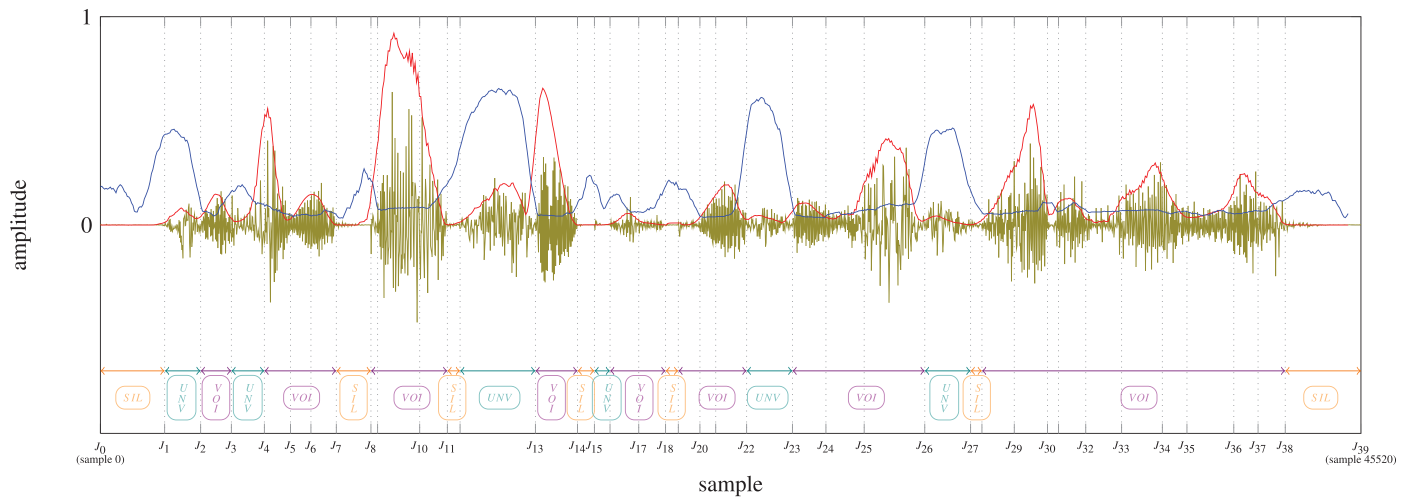


Fig. 13. The speech voiced/unvoiced/silent decision experiment. SIL, UNV and VOI, respectively, mean silent, unvoiced and voiced. Olive, blue and red, respectively, are the colors used to plot the input speech signal $s[\cdot]$, i.e., the file */test/dr1/mdab0/sa1.wav* from TIMIT, the ZCR feature vector $f_B[\cdot]$ and the energy feature vector $f_A[\cdot]$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

processing community considers this as being adequate to analyse speech data such as $s[\cdot]$, the signal that appears as an olive curve in Fig. 13. Furthermore, $V = 50\%$ follows the usual procedure adopted during the short-time analysis of speech signals, albeit this is not a critical choice. Additionally, the options for L and V allowed a comfortable visualisation of $f_B[\cdot]$ and $f_A[\cdot]$, plotted respectively in red and blue in Fig. 13, in which the three signals are time-aligned so that the kth sample of $f_B[\cdot]$ and of $f_A[\cdot]$, $(0 \leqslant k \leqslant T-1)$, correspond to the window placement that covers the interval from $s_{k\lfloor L\frac{100-V}{100}\rfloor}$ to $s_{k\lfloor L\frac{100-V}{100}\rfloor+L}$, i.e., from s_{256K} to $s_{256K+512}$. Since $T = \lfloor\frac{(100\cdot M)-(L\cdot V)}{(100-V)\cdot L}\rfloor = \lfloor\frac{(100\cdot 45520)-(512\cdot 50)}{(100-50)\cdot 512}\rfloor = 176$, both $f_B[\cdot]$ and $f_A[\cdot]$, indexed from 0 to 175, were dilated on the horizontal axis prior to be plotted, allowing a better visualisation, comparison and understanding of the proposed ideas.

On one hand, each unvoiced segment of $s[\cdot]$ matches a region with two characteristics: an inconstant and high ZCR in $f_B[\cdot]$, and a high energy in $f_A[\cdot]$. On the other hand, voiced parts of $s[\cdot]$ are the ones that correspond to low and relatively constant ZCRs in $f_B[\cdot]$ associated with high energies in $f_A[\cdot]$. Finally, silent segments correspond to those with low energies in $f_A[\cdot]$, disregarding $f_B[\cdot]$. To be considered either high or low, ZCRs and energies obey the hard thresholds respectively named as H_B and H_A. Summarising, the strategy is:

```
For the kth sample of $f_B[\cdot]$ and $f_A[\cdot]$, $(0 \le k \le T-1)$:
  {
  If ($f_{A_k} < H_A$)
      the corresponding region in $s[\cdot]$ is silent ;
  else if ($f_{B_k} < H_B$)
      the corresponding region in $s[\cdot]$ is voiced ;
  else
      the corresponding region in $s[\cdot]$ is unvoiced.
  }
```

As documented in [12]-pp.34, ZCR measures for voiced and unvoiced speech are, respectively, close to 1400 and 4900 zero-crossings per second. Thus, a reasonable value for H_B is the mean, i.e., $\frac{1400+4900}{2} = 3150$ zero-crossings per second. For 32 mili-seconds (ms), $H_B = 3150 \cdot 0.032 = 100.8$ zero-crossings per 512 samples. Since PA was applied, H_B becomes $\frac{100.8}{L-1} = \frac{100.8}{511} \approx 0.197$, the value I adopted.

For energy, contrastingly, I observe that the threshold of hearing for human beings is 0 dB at the pressure intensity of 20µPa, i.e., 20 micro-Pascal micro-Pascal, at 1000 Hz and 25 °C, as documented in [71]-pp.150–155. In order to compare a spoken signal with the threshold of hearing, its specific playback level should be known but this is not simple in practice. Notwithstanding, an usual assumption is to consider such a level as being the smallest possible signal represented by means of the speech coding system defined at the time the signal was digitalised and quantised. Equivalently, the fairly flat bottom of the threshold of hearing, for the frequencies within the main range of speech, is simply aligned to the energy level represented by the least significant coding bit. TIMIT speech files were quantised with 16 bits, with one of them reserved for signalling, i.e., positive or negative, and the remaining ones for amplitude description, hence, the amplitude axis, for both positive and negative amplitudes, varies at each $\frac{1}{2^{16-1}} = \frac{1}{2^{15}}$. Considering each window placement covers $L = 512$ samples, the normalised level is $512 \cdot \frac{1}{2^{15}} = \frac{2^9}{2^{15}} = 2^{-6} = 0.015625 \approx 0.016$, which is the value I chose for H_A.

If the readers repeat this experiment for the entire TIMIT corpus, which contains 630 spoken sentences, the same successful style of discrimination observed in Fig. 13 will be obtained. In fact, many similar descriptions for the distinction among VUSS based on signal energy and ZCRs have already been documented in the literature, such as [47], published forty years ago, and [23], that is a more recent work. Aiming to offer the readers a more attractive and novel classification scheme, taking advantage of the previous formulation, I shall describe my proposal for isolated-sentence word segmentation.

According to [48]-pp.125, no known detail contained in a raw speech waveform corresponds directly to the white spaces that separate two words in written form. In fact, as I have observed through the years, the raw waveform rarely presents distinct and clear silent spaces between words, making this kind of segmentation a hard and complex challenge. In some particular cases, only the context can serve as the basis to find whether or not a boundary exists. One example refers to the underlined words in the next pair of sentences:

"... *Is there a good wine ? Yes, there is some. What age is it ?* ..."

"... *Is there a good wine ? Yes, there is. Somewhat aged ?* ..."
.

Surely, not only an isolated speech fragment has to be taken into account to solve this issue but also the entire sentence. Notably, the authors of paper [50] have already shown the advantages

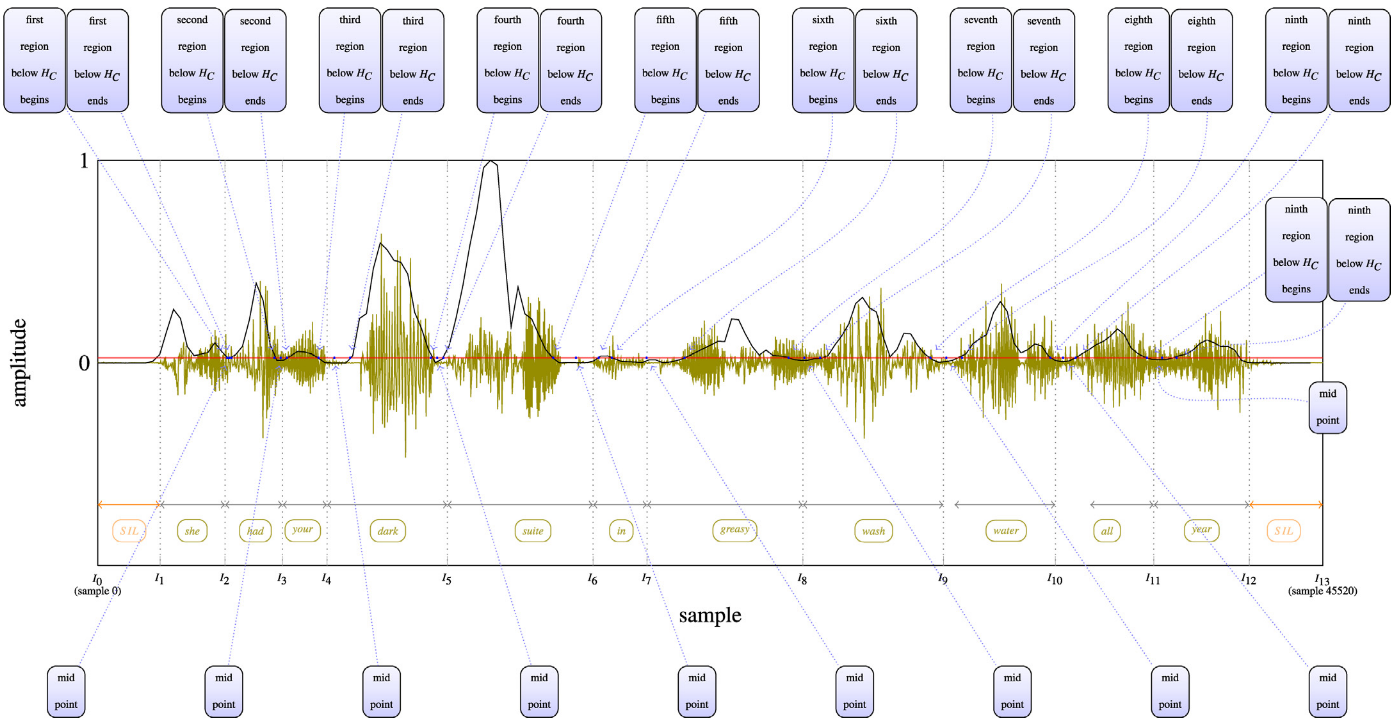


Fig. 14. The speech segmentation experiment, with the tags I_k, $(0 \le k \le 13)$, delimiting each spoken word. Olive and black, respectively, are the colors used to plot the input speech signal $s[\cdot]$, i.e., the file */test/dr1/mdab0/sa1.wav* from TIMIT, and the feature vector $f_C[\cdot]$. The red horizontal line corresponds to the threshold H_C. As in the previous figure, SIL identifies the silent periods.(For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article).

of using a few seconds of bootstrapping data, instead of particular speech segment only, for word segmentation. Inspired by such formulations, my experimental proposal is

$$f_{Ck} = f_{Bk} \cdot f_{Ak}, \qquad (0 \leqslant k \leqslant T-1),$$

which reflects a ZCR-based weighted set of energy components. Although $f_A[\cdot]$, originally described in [1], is normalised considering the whole input signal energy that is distributed along the time so that its kth position is influenced by the entire speech, $f_B[\cdot]$ associated with PA contains only local spectral information. As a result, global temporal workload [1] is weighted on the basis of local frequency content, semantically characterising the intended physical principle. Once $f_C[\cdot]$ is obtained, the strategy for decision becomes:

```
For $(1 \leqslant k \leqslant T-2)$:
  If ($f_{Ck}$ is the mid-point between the beginning and the end of a region below $H_C$)
      $s_{k \cdot \lfloor L \frac{100-V}{100} \rfloor}$ is defined as being a word boundary.
```

$H_C = H_B \cdot \frac{1}{T}\sum_{k=0}^{T-1} f_{Ck} = \frac{0.197}{T}\sum_{k=0}^{T-1} f_{Ck}$, which represents a hard threshold for usage in conjunction with the new feature vector $f_C[\cdot]$, is formed on the basis of two contributions, just as that vector was: the local spectral threshold, H_B, and the temporal vector mean, which represents a distributed piece of information regarding it.

Fig. 14 illustrates $s[\cdot]$, $f_C[\cdot]$, H_C and the exact word boundaries, according to the corresponding *wrd* text file contained in the same TIMIT folder that includes $s[\cdot]$. The close, and certainly acceptable, matches between the points that satisfy the condition imposed by the proposed strategy and the real boundaries are clearly perceptible. Particularly, the highest positive spike in the black curve of Fig. 14, close to the position of I_5, expresses the most improbable point for a word boundary in $s_C[\cdot]$, since it represents a considerable activity of the speaker's vocal apparatus. Consequently and in opposition, its lowest negative spikes, the ones we are interested in, indicate the most probable points for the word boundaries in $s_C[\cdot]$, because they represent a more relaxed speaker's work to utter, in view of a semantic shift in the way of speaking.

Algorithm 15 presents the complete source-code that implements the procedures described for isolated-sentence word segmentation, so that the readers can easily replicate the experiments.

The global accuracy of the proposed approach was evaluated on the basis of a recent tool called *R-value*, explained in [49] and defined as being

$$R = 1 - \frac{|r_1| + |r_2|}{200},$$

considering

$$r_1 = \sqrt{(100 - HR)^2 + (OS)^2} \qquad \text{and} \qquad r_2 = \frac{-OS + HR - 100}{\sqrt{2}}.$$

OS and *HR* are respectively known as *over-segmentation* and *hit-rate*. The former indicates the percentage of the number of boundaries correctly detected in relation to the total number of existing ones. On the other hand, the latter corresponds to the percentage of the total number of boundaries detected, both correctly and incorrectly, in relation to the total number of existing ones. Prominent results, fully acceptable in terms of the points labelled as being word boundaries, were observed when the proposed approach was applied to all the 630 sentences *sa1* spoken by the TIMIT speakers. Disregarding their borders, i.e., their start and end points, there are a total of $10 \cdot 630 = 6300$ boundaries between the $11 \cdot 630 = 6930$ existing words in those sentences. Assuming that *OS* and *HR* were counted so that the former and the latter are respectively equal to 1.7378% and 93.4755%, then $R = 0.9379 = 93.70\%$, consolidating the hypothesis that $f_C[\cdot]$ contains useful information for word segmentation and motivating its further investigation.

In comparison with state-of-the-art procedures [50–56], the proposed algorithm is novel, introducing the applications of ZCR-based weighted energy components for isolated-sentence word

Algorithm 15 : C++ function for word segmentation

```
void word_segmentation(double* s, int M)
{
double mean = 0;
for(int k = 0; k < M; k++)
        mean+ = s[k]/(double)(M);
for(int k = 0; k < M; k++)
        s[k]− = mean; //at this point, the arithmetic mean of the input signal
is 0
int L = 512;
int V = 50;
int T = (int)((100 ∗ M − L ∗ V)/((100 − V)∗L));
double ∗f_B = new double[T];
double E = 0; // E represents the total energy over all the positions of the
window, that is required to normalise f_A[·]
double ∗f_A = new double[T];
for(int k = 0; k < T; k++)
        {
        f_B[k] = 0;
        f_A[k] = 0;
         for(int  i = k ∗ ((int)(((100 − V)/100.0) ∗ L)); i < k ∗ ((int)(((100 −
V)/100.0) ∗ L)) + L − 1; i++)
                {
                f_B[k]+ = (s[i] ∗ s[i + 1] < 0)?1:0;
                f_A[k]+ = pow(s[i], 2);
                }
        f_B[k]/ = (double)(L − 1);
        E+ = f_A[k];
        }
for(int k = 0; k < T; k++)
        f_A[k]/ = E; // normalisation
double ∗f_C = new double[T];
for(int k = 0; k < T; k++)
        f_C[k] = f_B[k] ∗ f_A[k];
mean=0;
for(int k = 0; k < T; k++)
        mean + = f_C[k];
mean/ = (double)(T);
double H_C = 0.197∗mean;
int k = 0;
int start_region, end_region;
while ((k < T)&&(f_C[K] < H_C))
        k++;
do // search for regions below H_C
        {
        while ((k < T)&&(f_C[K] > H_C))
                k++;
        start_region = k;
        while ((k < T)&&(f_C[K] < H_C))
                k++;
        end_region = k;
        if (k < T)
                   printf("\n    Word    boundary    detected    at    s[%d].",
(int)((end_region+start_region)/2.0) * (int)(((100-V)/(100.0)) * L));
        } while (k < T);
}
```

segmentation. Furthermore, it presents the following advantages. First, it is based on a simple inspection of a feature vector originated from the most humble concepts involving spectral and temporal analysis, i.e., ZCRs and signal energy, respectively. Second and opposed to some of the usual procedures just cited, its order of computational complexity, both in terms of space and time, is linear ($O(M)$) in relation to the input signal length (M), allowing real-time implementations. This linearity is a direct consequence of the strategies adopted to define B_1, A_1 [1] and algorithm 15, because all of them work based on a direct traverse of the input signal under analysis. Lastly, saving the appropriate proportions, it is as accurate as the above-referenced similar techniques, which report their successful results within the range from 54 to 95% for English, French, Chinese and also ancient language corpora.

Obviously, the strategy I have just presented does not take into account complex lexical issues and cannot be determined as being the definite algorithm for word segmentation, nevertheless, it balances creativity, simplicity and accuracy, satisfying the objective of this study and providing the initial insights for additional research in the same direction.

4.2. Image analysis and re-synthesis for border extraction

At an early stage, most of the image processing algorithms for PRKbS have to deal with a segmentation step [57–59], which refers to the analysis of the input signal aiming to highlight meaningful areas of interest. Usually, such a process is based on border extraction, a strategy that identifies the pixels responsible for defining the boundaries between objects or specific parts of the image. Particular regions in which the fundamental frequencies are relatively high are most likely to represent the borders, thus, ZCRs, capturing those spectral components, are considered in this experiment instead of the usual methods, such as watershed [57] and filtering operations [60].

In order to establish an overall comparison, I state that ZCRs have one great advantage over the usual techniques: they do not take into account the frequencies higher than the fundamental one, completely disregarding minor variabilities among neighbour pixels which could hinder a precise characterisation of the borders. The direct consequence is that filtering approaches, such as those detailed in [60–63], are not better than ZCRs for border extraction because filters are never ideal in practice, i.e., contaminations resulting from imperfect filtering are not present when ZCRs are used. Thus, the proposed approach, which might definitively **not** be understood as a filtering operation, consists of an FE procedure for PRKbS playing the role of a DSP algorithm in which convolutional filtering is advantageously replaced by an analysis followed by re-synthesis of the input 2D signal.

My strategy requires ZCRs from all the minor constant-area regions over the input image to be interrelated. Neither B_1 nor B_3 are suitable to perform this task. Instead, B_2 particularly associated with EA is ideal and so became the choice. The detailed scheme follows, where the ideal value for Δ is determined empirically. Obviously, it is at least equal to four, otherwise, no ZCR could be counted. My tests have shown that sixteen is a great option for synthesising target images of different resolutions, allowing a reasonable visual quality.

```
Extract the raw data from the input gray-scaled image, storing it in the NxM
matrix $m[\cdot][\cdot]$;
Apply $B_2$ normalised with EA, choosing Q so that $\xi_Q$ contains elements
extracted from non-overlapping rectangles with about $\Delta$ pixels;
Select either $\xi_1$, or $\xi_2$, or ..., or $\xi_Q$ to synthesise the target image, being $\xi_1$
and $\xi_Q$, respectively, the options for worst and best resolutions;
Synthesise the target image, $\overline{m}[\cdot][\cdot]$, as follows:
  For the $k^{th}$ sample of $\xi_Q[\cdot]$, $(0 \leqslant k < X^2)$:
    draw a $\lfloor \frac{N}{X} \rfloor x \lfloor \frac{M}{X} \rfloor$ rectangle, using the color
((maximum_level_of_black_color)$\cdot(\xi_{Q_k})$), from the point $(\frac{k}{X}, k\%X)$ to the point
$(\frac{k}{X} + \lfloor \frac{N}{X} \rfloor, k\%X + \lfloor \frac{M}{X} \rfloor)$, being
    % the remainder of the integer division, just as in C/C++ programming
language.
```

The strategy adopted to define the color of each rectangle consists of a simple linear proportion of the maximum level of black color, that is usually equal to 255 for 8-bit images [60], based on the corresponding kth value of $\xi[\cdot]$, which varies from 0 to 1. The former extreme forces a white rectangle to be plotted, which corresponds, in practice, to the absence of plot, since a white background is assumed. At the same time, the latter creates a black rectangle. Intermediary gray colors, produced for $0 < k < 1$, usually appear close to the black ones, stimulating the completion effects during the paintings.

Aiming to exemplify the proposed approach, Fig. 15(a) shows the digit "5", handwritten and digitalised as being a matrix with $N = 508$ rows and $M = 508$ columns. The image was analysed at the resolution provided by using $Q = 31$, which produces a feature vector containing $2^2 + 3^2 + 5^2 + ... + (113)^2 + (127)^2$ elements, according to Section 2. Then, sub-vectors ξ_1, ξ_2, ξ_7, ξ_{11}, ξ_{18} and ξ_{31},

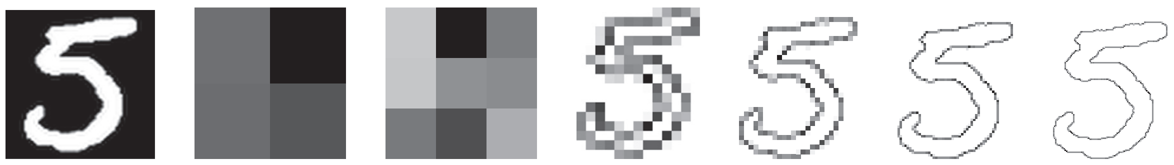

(a) the original 508 *x* 508 image. (b) the synthesised image based on ξ_1. (c) the synthesised image based on ξ_2. (d) the synthesised image based on ξ_7. (e) the synthesised image based on ξ_{11}. (f) the synthesised image based on ξ_{18}. (g) the synthesised image based on ξ_{31}.

Fig. 15. The input image and its synthesised versions.

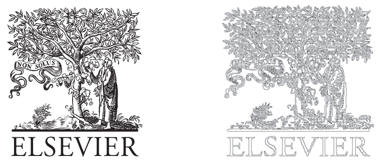

(a) the original 508 *x* 508 picture. (b) the synthesised image based on ξ_{55}.

Fig. 16. (a): The Elsevier logo; (b): corresponding image containing only borders.

Algorithm 16 : C++ code for image border extraction. The function receives three parameters, i.e., $m[\cdot][\cdot]$ and the addresses of M and N, modifying all of them: the first one, so that it becomes the synthesised squared image ($\overline{m}[\cdot][\cdot]$), and the second and third ones, so that a new size can be set.

```
void border_extraction(double** m, int* N, int* M)
{
double mean = 0; // adjust mean to be zero
for(int p = 0; p < (*N); p++)
        for(int q = 0; q < (*M); q++)
                mean+ = m[p][q]/(double)((*M) * (*N));
for(int p = 0; p < (*N); p++)
        for(int q = 0; q < (*M); q++)
                m[p][q]- = mean;
int X =; // the desired value, i.e., number of image divisions to produce ξQ.
In the example of Figure˜??, it was set to 127
int total_size_of_f = X * X;
double *f = new double[total_size_of_f];
for(int i = 0;i <total_size_of_f;i++)
        f[i] = 0;
int L1 = (int)((*N)/(double)(X));
int L2 = (int)((*M)/(double)(X));
for(int i = 0;i <((int)((*N)/(double)(X)))*X - 1;i++)
        for(int j = 0;j <((int)((*M)/(double)(X)))*X;j++)
                   f[(((int)(i/L2))*(X))+((int)(j/L1))]+ =(m[i][j] * m[i + 1][j] <
0)?1:0;
for(int i = 0;i <((int)((*N)/(double)(X)))*X;i++)
        for(int j = 0;j <((int)((*M)/(double)(X)))*X - 1;j++)
                   f[(((int)(i/L2))*(X))+((int)(j/L1))]+ =(m[i][j] * m[i][j + 1] <
0)?1:0;
double highest_ZCR = 0; // image normalisation
for(int l = 0;l <total_size_of_f;l++)
        if(f[l]> highest_ZCR)
                highest_ZCR = f[l];
for(int l = 0;l <total_size_of_f;l++)
        f[l]/ = highest_ZCR;
for(int k = 0;k <total_size_of_f;k++) // image synthesis
           for(int     p = (int)(k/(double)(X)); p < (int)(k/(double)(X)) +
(int)((*N)/(double)(X)) - 1; p++)
                 for(int q = k%X; q < k%X + (int)((*M)/(double)(X)) - 1; q +
+)
                      m[p][q] = 255 * f[k];
*N = X; // adjust size
*M = X; // adjust size
}
```

composed respectively of $X = (2)^2 = 4$, $X = (3)^2 = 9$, $X = (17)^2 = 289$, $X = (31)^2 = 961$, $X = (61)^2 = 3721$ and $X = (127)^2 = 16129$ elements, were separately used to synthesise the target images shown in Fig. 15(b)–(g). Clearly, the higher the resolution is, the better the characterisation of the border will be.

When performed using the entire database of handwritten digits downloaded from [64], this experiment succeeds. Complementarily, Fig. 16(a) and (b) shows, respectively, the 508 *x* 508 Elsevier logo and its corresponding borders extracted on the basis of the proposed approach adopting ξ_{55}. All the tests performed allow me to state that the proposed approach produces equivalent perceptual results in comparison with those obtained when algorithms of cutting-edge nature, such as [65–67]. are adopted. Furthermore, in addition to the advantages discussed above, it also presents an attractive order of time and space complexities [15]. Finally, based on the fact that my technique completely differs from the current ones, in its nature and essence, I refrain from establishing more detailed analogies. ZCR-based algorithms for 2D signal processing cannot even be found in the literature, especially when it comes to border extraction.

Algorithm 16 contains a C/C++ function that receives $m[\cdot][\cdot]$ and its dimensions as input, modifying them so that the border image is synthesised. Basically, the function corresponds to algorithm 12 adapted so that $f[\cdot]$ stores just the analysis of $m[\cdot][\cdot]$ at one particular resolution, i.e., $f[\cdot] = \xi_Q[\cdot]$, implying that X becomes an unique integer number, instead of being a vector. For instance, if $m[\cdot][\cdot]$ corresponds to the raw data extracted from the image shown in Fig. 15(a), then, X is set to 127 and $f[\cdot] = \xi_{31}[\cdot]$ becomes a $(127)^2 = 16,129$ sample-long vector. In the specific case treated in this example application, for which $f[\cdot]$ corresponds exactly to ξ_Q, X may also assume values of non-prime numbers. An intelligent choice is to set it as being a multiple of both the original N and M in order to avoid some parts of the image to be discarded, as exemplified in Fig. 8b of Section 2.

4.3. Biomedical signal analysis

In one of my previous works from 2007 [68], an algorithm to distinguish between healthy speech (HS) and pathologically-affect speech (PAS) was presented. The former and the latter encompass, respectively, the individuals with no abnormality in their vocal apparatus and the ones with a pathology in their larynxes. In that study, time-frequency features [69] served as input to a Support Vector Machine (SVM) [70] dedicated to examine the four-second sustained /a/ vowel sounds, as in the word "*dogma*", emitted by people enrolled in our system. Similarly, in this experiment, I use speech data digitalised at 22, 050 Hz, 16-bit [71], using a wideband microphone in a sound cabinet, from fourteen healthy subjects and from fourteen individuals with Reinke's edema in their larynx. All the twenty-eight voices were accredited by medical professionals based on specific hardware tools which allow precise image examinations and detailed vocal analyses.

Initially, the radiation effects from the speakers' lips were removed, as a pre-processing stage known as pre-emphasis [48]-pp.168 [72]-pp.25, before applying the proposed approach. The first-order finite impulse response (FIR) high-pass filter [13] whose coefficients are $g[\cdot] = \{1, -0.95\}$ was used, via convolution [13], to perform this task. Considering $s[\cdot]$, of length M, as being an input

Table 1
Results obtained from the experiment on biomedical signal analysis.

Value chosen for C	19.9%	24.9%	33.3%
Corresponding value of *T*	$\lfloor \frac{100}{C} \rfloor = \lfloor \frac{100}{19.9} \rfloor = 5$	$\lfloor \frac{100}{C} \rfloor = \lfloor \frac{100}{24.9} \rfloor = 4$	$\lfloor \frac{100}{C} \rfloor = \lfloor \frac{100}{33.3} \rfloor = 3$
Corresponding confusion matrix	$\begin{array}{ccc} & PAS & HS \\ PAS & 7 & 0 \\ HS & 0 & 7 \end{array}$	$\begin{array}{ccc} & PAS & HS \\ PAS & 7 & 0 \\ HS & 2 & 5 \end{array}$	$\begin{array}{ccc} & PAS & HS \\ PAS & 6 & 1 \\ HS & 1 & 6 \end{array}$
Resulting matrix sensibility **(% of accuracy)**	$\frac{7+7}{14} = 100\%$	$\frac{7+5}{14} = 85.71\%$	$\frac{6+6}{14} = 85.71\%$

speech signal, the procedure is:

$$s_k \leftarrow s_k - (0.95 \cdot s_{k-1}) \quad , \quad (1 \leqslant k < M) \quad .$$

Likewise any usual classification scheme, this approach intends to offer the classifier a set of fixed-length feature vectors, implying that B_1 is useless. On the other hand, both B_2 and B_3 output a set of T elements regardless of M, being the latter a preferable method because it allows the regularity of ZCRs to be analysed along a period. From Section 1.2 we have learnt that ZCRs are more likely to capture the fundamental frequencies of the speech signals, i.e., their pitch [12] (F_0), disregarding the resonances of the vocal tract, also known as being the formants (F_1, F_2, ...), which correspond to their higher frequencies. Complementarily, based on the fact that B_3 intrinsically normalises the frequencies it captures, the proposed approach registers only *the way* pitches vary, disregarding their values themselves. Thus, for any speaker, HS is expected to keep an almost linear variation in response to the regular and non-excessive effort performed by the vocal folds and associated organs during vibration. On the other hand, PAS usually shows irregular variations due to an excessive, and sometimes irresponsive, effort performed by the speakers' vocal system.

Assuming that seven signals from each class were adopted as being their representative models and the other seven were used to test the proposed approach, an ordinary absolute distance (AD) measurement [73] was selected to serve as the classifier. The distances from each testing vector to all the template models are registered, then, the class for which the lowest one belongs to, guides the assignment. Table 1 shows the best results obtained for all the possible $\binom{14}{7}^2 = (\frac{14!}{7!(14-7)!})^2 = (3432)^2$ hold-out cross-validations procedures [74], considering different options for C.

The respective accuracies suggest that a finer analysis, i.e., $C = 19.9\%$, is required to characterise important data. For $C = 33.3\%$, excessive information is grouped together in each of the $T = 3$ coefficients of the feature vectors, causing the misclassification of one PAS member that was labelled as being an HS one. Although for $C = 24.9\%$, there are also incorrect assignments, HS members labelled as being PAS do not cause serious consequences, as in the previous case in which the opposite occurs.

The above-mentioned characteristics of HS and PAS allow a relative generalisation of the results presented in Table 1, despite the modest size of the database, for which the lack of volunteers and the rigorous accreditations prevented further expansion. Thus, I consider a meaningful and relevant outcome was obtained. Detailed comparisons with similar state-of-the-art algorithms, such as those documented in [68,75–79], were avoided because their databases and pathologies differ, however, the proposed approach is overall as accurate as, and much simpler than, those strategies. Furthermore, AD was purposely selected to play the role of the classifier just to emphasise the relevance of the ZCR-based features, i.e., due to the potential solution offered by the latter, the former, consisting of the simplest existing possibility, performs an effortless job.

5. Conclusions

This study, dedicated to our neurocomputing community, was carefully written, polished and reviewed to serve as a tutorial on ZCRs for both 1D and 2D DSP applications designed for PRKbS. All the concepts I described correspond to the outcome of a wide and detailed research work. The readers can observe that, despite the fact that ZCRs are well known in the literature, the different ways I show their applicability, majorly concerning the 2D cases, are totally new.

Specifically, three methods for feature extraction based on ZCRs were presented just after the literature review section: B_1, which is the simplest one and is intended to produce variable-length feature vectors, is useful to search for a specific event or characteristic in a digital signal, such as word segmentation or the distinction between voiced and unvoiced frames of a speech signal. B_2, on the other hand, for which an application on image border extraction based on analysis and re-synthesis was exhibited, is adopted to inspect how ZCRs are distributed along the time, or space, in different levels of resolution. Finally, B_3, exemplified for the distinction of healthy and pathologically-affected biomedical signals, is usefull in searching for the possible variabilities of ZCRs as time or space advances. Three different types of normalisations, named TA, PA and EA, were also designed to work together with B_1 and B_2. Furthermore, sixteen algorithms, sixteen figures, six numerical examples and one table were included in the text.

The readers may have learnt from many references, such as [1], that the ordinary PRKbS require a classifier to be associated with the features extracted from raw data. The more such features linearly separate the mixed data from different classes in a correct manner, the less exquisite the classifier should be, and vice-versa. Particularly based on the example applications described in Section 4, I highlight one aspect of the proposed approaches: the simplest existing classifiers, i.e., HT and AD, were used, implying that the ZCR-based features brightly performed their work. Possibly, this is due to the fact that ZCRs are, by themselves, neurocomputing agents, as shown in Section 2. To treat much more complex problems, the potential of ZCRs may be enlarged by associating them with different types of neural networks [80], SVMs [[81–83]], hidden markov models (HMMs) [84], paraconsistent [85], and others.

In relation to noisy inputs, B_3 is less influenced than B_1 and B_2 when the noise, regardless of being white, pink, red, and so on [7,13], is uniformly distributed along the signal under analysis. This is because that method describes *the way* ZCRs vary, instead of counting them, implying that the artifacts introduced affect the entire signal more or less in the same manner, vanishing their effect over B_3.

Concluding, the proposed approaches provide a valid contribution for both young researchers, who are expected to take advantage of the fundamental concepts and basic inputs drawn from DSP and PRKbS theory, and experienced professionals, for whom this text may serve as an initial insight to the project of fruitful and prominent algorithms. As mentioned in [1], this study also draws the DSP and PRKbS scientific communities' attention to consider the use of ZCRs, somewhen in conjunction with signal energy [1] or other features, to conform *creativity, simplicity*, and *accuracy*.

All the data used during the experiments, excluding TIMIT which is controlled by the Linguistic Data Consortium (LDC), are available to the scientific community upon prior request [3] so that the procedures could be reproduced. Further research related with ZCRs focuses both on minor changes in the proposed approaches

[3] Please, send requests to *guido@ieee.org*

so that more specific issues are properly addressed. An intriguing open question: are there humbler features than ZCRs which are capable of achieving similar or better results for basic spectral signal description?

Acknowledgements

I am very grateful to CNPQ - "Conselho Nacional de Pesquisa e Desenvolvimento", in Brazil, for the grants provided, through the process 306811/2014-6, to support this research.

References

[1] R.C. Guido, A tutorial on signal energy and its applications, Neurocomputing 179 (2016) 264–282.
[2] J. Xu, A multi-label feature extraction algorithm via maximizing feature variance and feature-label dependence simultaneously, Knowl. Based Syst. 98 (2016) 172–184.
[3] Q. Zhou, H. Zhou, T. Li, Cost-sensitive feature selection using random forest: selecting low-cost subsets of information features., Knowl. Based Syst. 95 (2016) 1–11.
[4] L. Yijing, Adapted ensemble classification algorithm based on multiple classification systems and feature selection for classifying multi-class unbalanced data., Knowl. Based Syst. 94 (2016) 88–104.
[5] S. Garcia, J. Luengo, F. Herrera, Tutorial on practical tips of the most influential data preprocessing algorithm in data mining., Knowl. Based Syst. 98 (2016) 1–29.
[6] Y. Meng, J. Liang, Y. Qian, Comparison study of orthonormal representations of functional data in classification., Knowl. Based Syst. 97 (2016) 224–236.
[7] S.M. Alessio, Digital Signal Processing and Spectral Analysis for Scientists: Concepts and Applications, 1, Springer, 2016.
[8] B. Stroustrup, The C++ Programming Language, 4, Addison-Wesley Professional, 2013.
[9] M. Steenbeck, A contribution to the behavior of short AC arcs during the current zero crossing., Z. Phys. 65 (1-2) (1930) 88–91.
[10] F.M. Young, J.C. Grace, Zero crossing intervals of a sine wav in noise., J. Acoust. Soc. Am. 25 (4) (1953) 832.
[11] J.P. Ertl, Detection of evoked potentials by zero crossing analysis., Electroencephalogr. Clin. Neurophysiol. 18 (6) (1965) 630–631.
[12] L. Deng, D. O'Shaughnessy, Speech Processing: A Dynamic and Optimization-oriented Approach, CRC Press, 2003.
[13] A.V. Oppenheim, R.W. Schafer, Discrete-time Signal Processing, 3, Prentice-Hall, 2009.
[14] S. Haykin, B.V. Veen, Signals and Systems, 2, Wiley, 2002.
[15] S. Arora, Computational Complexity: a modern approach, Cambridge University Press, 2009.
[16] S. Goswami, P. Deka, B. Bardoloi, D. Dutta, D. Sarma, A novel approach for design of a speech enhancement system using NLMS adaptive filter and ZCR based pattern identification., in: Proceedings of the 2013 1st International Conference on Emerging Trends and Applications in Computer Science (ICETACS), 2013, pp. 125–129.13-14.
[17] H.-.M. Park, R.M. Stern, Spatial separation of speech signals using amplitude estimation based on interaural comparisons of zero-crossings., Speech Commun. 51 (1) (2009) 15–25.
[18] A. Ghosal, R. Chakraborty, R. Chakraborty, S. Haty, B.C. Dhara, S.K. Saha, Speech/music classification using occurrence pattern of ZCR and STE., in: Proceedings of the Third International Symposium on Intelligent Information Technology Application (IITA), 3, 2009, pp. 435–438.
[19] R.R. Shenoy, C.S. Seelamantula, A zero-crossing rate property of power complementary analysis filterbank outputs., IEEE Signal Process. Lett. 22 (12) (2015) 2354–2358.
[20] A.V. Levenets, C.E. Un, Method for evaluating periodic trends in measured signals based on the number of zero crossings., Meas. Tech. 58 (4) (2015) 381–384.
[21] R.R. Shenoy, C.S. Seelamantula, Spectral zero-crossings: localization properties and application to epoch extraction in speech signals., in: Proceedings of the International Conference on Signal Processing and Communications (SPCOM), 2012, pp. 1–5.
[22] M. Jalil, F.A. Butt, A. Malik, Short-time energy, magnitude, zero crossing rate and autocorrelation measurement for discriminating voiced and unvoiced segments of speech signals., in: Proceedings of the International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013, pp. 208–212.
[23] R.G. Bachu, S. Kopparthi, B. Adapa, B.D. Barkana, Voiced/unvoiced decision for speech signals based on zero-crossing rate and energy., in: K. Elleithy (Ed.), Advanced Techniques in Computing Sciences and Software Engineering, Springer, 2010, pp. 279–282.
[24] Y.-. I. Kim, H.-. Y. Cho, S.-. H. Kim, Zero-crossing-based channel attentive weighting of cepstral features for robust speech recognition: the ETRI 2011 CHiME challenge system., in: Proceedings of the Interspeech, 2011, pp. 1649–1652.
[25] S.J. An, R.M. Kil, Y.-. I. Kim, Zero-crossing-based speech segregation and recognition for humanoid robots., IEEE Trans. Consum. Electron. 55 (4) (2009) 2341–2348.
[26] A.S. Zandi, R. Tafreshi, M. Javidan, Predicting epileptic seizures in scalp EEG based on a variational bayesian gaussian mixture model of zero-crossing intervals., IEEE Trans. Biom. Eng. 60 (5) (2013) 1401–1413.
[27] M. Phothisonothai, M. Nakagawa, A complexity measure based on modified zero-crossing rate function for biomedical signal processing., in: Proceedings of the 13th International Conference on Biomedical Engineering (ICBME), 23, 2009, pp. 240–244.
[28] M.I. Khan, M.B. Hossain, A.F.M.N. Uddin, Performance analysis of modified zero crossing counts method for heart arrhythmias detection and implementation in HDL., in: Proceedings of the International Conference on Informatics, Electronics and Vision (ICIEV), 2013, pp. 1–6.
[29] C.-. H. Wu, H.-. C. Chang, P.-. L. Lee, Frequency recognition in an SSVEP-based brain computer interface using empirical mode decomposition and refined generalized zero-crossing., J. Neurosci. Methods 196 (1) (2011) 170–181.
[30] D. Guyomar, M. Lallart, K. Li, A self-synchronizing and low-cost structural health monitoring scheme based on zero crossing detection., Smart Mater. Struct. 19 (4) (2010).Article Number: 045017, 2010.
[31] L. Florea, C. Florea, R. Vranceanu, C. Vertan, Zero-crossing based image projections encoding for eye localization., in: Proceedings of the 20th European Signal Processing Conference (EUSIPCO), 2012, pp. 150–154.
[32] S. Watanube, T. Kotnatsu, T. Saito, A stabilized zero-crossing representation in the wavelet transform domain and its extension to image representation for early vision., in: IEEE TENCON - Digital Signal Processing Applications, 1996, pp. 496–501.
[33] J.G. Daugman, Pattern and motion vision without laplacian zero crossings., J. Opt. Soc. Am. A-5 (7) (1988) 1142–1148.
[34] K.-. L. Du, M.N.S. Swamy, Neural Networks and Statistical Learning, Springer, 2014.
[35] N. Nedjaha, F.M.G. Fran A§ a, M.D. Gregorio, L.M. Mourelle, Weightless neural systems., Neurocomputing 183 (2016) 1–2.
[36] H.C.C. Carneiro, F.M.G. Franca, P.M.V. Lima, Multilingual part-of-speech tagging with weightless neural networks., Neural Netw. 66 (2015) 11–21.
[37] G.G. Lockwood, I. Aleksander, Predicting the behaviour of g-RAM networks., Neural Netw. 16 (1) (2003) 91–100.
[38] T.F. Quatieri, Discrete-time Speech Signal Processing: Principles and Practice, Upper Saddle River, NJ: Prentice Hall, 2001.
[39] W. Chou, B.H. Juang, Pattern Recognition in Speech and Language Processing, Boca Raton: CRC Press, 2003.
[40] H. Beigi, Fundamentals of Speaker Recognition, New York: Springer, 2011.
[41] R.C. Guido, L.S. Vieira, S. Barbon Jr., A neural-wavelet architecture for voice conversion., Neurocomputing 71 (1-3) (2007) 174–180.
[42] T. Ogunfunmi, M. Narasimha, Principles of Speech Coding, CRC Press, 2010.
[43] K. Skoruppa, et al., The role of vowel phonotactics in native speech segmentation., J. Phonet. 49 (2015) 67–76.
[44] TIMIT speech corpus. linguistic data consortium (LDC), https://catalog.ldc.upenn.edu/LDC93S1.
[45] C. Kim, K.-. d. Seo, Robust DTW-based recognition algorithm for hand-held consumer devices., IEEE Trans. Consum. Electron. 51 (2) (2005) 699–709.
[46] X. He, L. Deng, Discriminative Learning for Speech Recognition, Morgan and Claypool Publishers, 2008.
[47] B. Atal, L. Rabiner, A pattern recognition approach to voiced-unvoiced-silence classification with applications to speech recognition., IEEE Trans. Audio, Speech, Lang. Process. 1 (24) (1976) 201–212.
[48] J. Harrington, S. Cassidy, Techniques in Speech Acoustics, The Netherlands: Kluwer Academic Publishers, 1999.
[49] O.J. Rosanen, U.K. Laine, T. Altosaar, An improved speech segmentation quality measure: the r-value, in: Proceedings of the Interspeech, 2009, pp. 1851–1854.
[50] S. Brognaux, T. Drugman, HMM-based speech segmentation: improvements of fully automatic approaches., IEEE-ACM Trans. Audio, Speech, Lang. Process. 24 (1) (2016) 5–15.
[51] A. Stan, et al., ALISA: an automatic lightly supervised speech segmentation and alignment tool., Comput., Speech Lang. 35 (2016) 116–133.
[52] R.H. Baayen, C. Shaoul, J. Willits, M. Ramscar, Comprehension without segmentation: a proof of concept with naive discriminative learning., Lang., Cognit., Neurosci. 31 (1) (2016) 106–128.
[53] F. Stahlberg, T. Schlippe, S. Vogel, T. Schultz, Word segmentation and pronunciation extraction from phoneme sequences through cross-lingual word-to-phoneme alignment., Comput., Speech, Lang. 35 (2016) 234–261.
[54] K.G. Estes, C. Lew-Williams, Listening through voices: infant statistical word segmentation and meaning acquisition through cross-situational learning., Dev. Psychol. 51 (11) (2015) 1517–1528.
[55] Ok. Rasanen, H. Rasilo, A joint model for word segmentation and meaning acquisition through cross-situational learning., Psychol. Rev. 122 (4) (2015) 792–829.
[56] L. White, S.L. Mattys, L. Stefansdottir, Beating the bounds: localised timing cues for word segmentation., J. Acoust. Soc. Am. 138 (2) (2015) 1214–1220.
[57] F. Nery, J.S. Silva, N.C. Ferreira, F. Caramelo, R. Faustino, An algorithm for the pulmonary border extraction in PET images, Proc. Technol. 5 (2012) 876–884.
[58] L.H. Son, T.M. Tuan, A cooperative semi-supervised fuzzy clustering framework for dental x-ray image segmentation, Expert Syst. Appl. 46 (2016) 380–393.
[59] X.-. Y. Wang, Pixel classification based color image segmentation using quaternion exponent moments., Neural Netw. 74 (2016) 1–13.

[60] M. Nixon, Feature Extraction & Image Processing for Computer Vision, 3, Academic Press, 2012.
[61] P. Zhang, T.D. Bui, C.Y. Suen, Wavelet feature extraction for the recognition and verification of handwritten numerals., Keynote Address at 6th International Program on Wavelet Analysis and Active Media Technology. Available at http://users.encs.concordia.ca/~bui/pdf/Keynote.pdf.
[62] S.E.N. Correia, J.M. Carvalho, R. Sabourin, On the performance of wavelets for handwritten numerals recognition., in: Proceedings of the 16th International Conference on Pattern Recognition, 2002, 3, 2002, pp. 127–130.
[63] X. You, L. Du, Y. Cheung, Q. Chen, A blind watermarking scheme using new nontensor product wavelet filter banks., IEEE Trans. Image Process. 19 (12) (2010) 3271–3284.
[64] The MNIST database of handwritten digits, Available at http://yann.lecun.com/exdb/mnist/.
[65] A. Pratondo, C.-. K. Chui, S.-.H. Ong, Robust edge-stop functions for edge-based active contour models in medical image segmentation., IEEE Signal Process. Lett. 23 (2) (2016) 222–226.
[66] Z.M. Hadrich A, A. Masmoudi, Bayesian expectation maximization algorithm by using b-splines functions: application in image segmentation., Math. Comput. Simulat. 120 (2016) 50–63.
[67] M. Liao, Automatic segmentation for cell images based on bottleneck detection and ellipse fitting., Neurocomputing 173 (2016) 615–622.
[68] E. Fonseca, R.C. Guido, P.R. Scalassara, C.D. Maciel, J.C. Pereira, Wavelet time-frequency analysis and least-squares support vector machine for the identification of voice disorders., Comput. Biol. Med. 37 (4) (2007) 571–578.
[69] P. Addison, J. Walker, R.C. Guido, Time-frequency analysis of biosignals., IEEE Eng. Biol. Med. Mag. 28 (5) (2009) 14–29.
[70] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, 2, Wiley-Interscience, 2000.
[71] M. Bossi, E. Goldberg, Introduction to Digital Audio Coding and Standards, Kluwer, 2003.
[72] F. Muller, Invariant Features and Enhanced Speaker Normalization for Automatic Speech Recognition, Logos Verlag, 2013.
[73] V. Serdarushich, Analytic Geometry, CreateSpace Independent Publishing Platform, 2015.
[74] J.H. Kim, Estimating classification error rate: repeated cross-validation, repeated hold-out and bootstrap., Comput. Stat. Data Anal. 53 (11) (2009) 3735–3745.
[75] Z. Ali, I. Elamvazuthi, M. Alsulaiman, G. Muhammad, Detection of voice pathology using fractal dimension in a multiresolution analysis of normal and disordered speech signals., J. Med. Syst. 40 (1) (2016) 1–10.
[76] D. Panek, A. Skalski, J. Gajda, R. Tadeusiewicz, Acoustic analysis assessment in speech pathology detection., Int. J. Appl. Math. Comput. Sci. 25 (3) (2015) 631–643.
[77] M. Alsulaiman, Voice pathology assessment systems for dysphonic patients: detection, classification, and speech recognition., IETE J. Res. 60 (2) (2014) 156–167.
[78] M.J. Pulga, A.C. Spinardi-Panes, S.A. Lopes-Herrera, L.P. Maximino, Evaluating a speech-language pathology technology., Telemed. e-health 20 (3) (2014) 269–271.
[79] T.L. Whitehill, S. Bridges, K. Chan, Problem-based learning (PBL) and speech-language pathology: a tutorial., Clin. Linguist. Phonet. 28 (1-2) (2014) 5–23.
[80] S. Haykin, Neural Networks and Learning Machines, 3, Prentice Hall, 2008.
[81] M. Jandel, Biologically relevant neural network architectures for support vector machines., Neural Netw. 49 (2014) 39–50.
[82] Y. Leng, Employing unlabeled data to improve the classification performance of SVM and its applications in audio event classification., Knowl. based Syst. 98 (2016) 117–129.
[83] L. Shen, Evolving support vector machines using fruit fly optimization for medical data classification., Knowl. Based Syst. 96 (2016) 61–75.
[84] A.M. Fraser, Hidden Markov Models and Dynamical Systems, Society for Industrial and Applied Mathematics, 2009.
[85] R.C. Guido, S. Barbon Jr., R.D. Solgon, K.C.S. Paulo, L.C. Rodrigues, I.N. Silva, J.P.L. Escola, Introducing the discriminative paraconsistent machine (DPM)., Inf. Sci. 221 (2013) 389–402.