Physica A 400 (2014) 159–167

Contents lists available at ScienceDirect

Physica A

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

On the computational complexity of the empirical mode decomposition algorithm

Yung-Hung Wang [a], Chien-Hung Yeh [a,b], Hsu-Wen Vincent Young [a], Kun Hu [c], Men-Tzung Lo [a,*]

[a] Research Center for Adaptive Data Analysis, National Central University, Chungli, Taiwan, ROC

[b] Department of Electrical Engineering, National Central University, Chungli, Taiwan, ROC

[c] Medical Biodynamics Program, Division of Sleep Medicine, Brigham and Women's Hospital, Harvard Medical School, Boston, MA 02115, United States

HIGHLIGHTS

- The order of the computational complexity of the EMD is equivalent to FFT.
- Optimized program is proposed to speed up the computation of EMD up to 1000 times.
- Fast HHT with optimized EMD/EEMD algorithm can operate in real-time.

ARTICLE INFO

Article history:
Received 30 October 2013
Received in revised form 30 December 2013
Available online 21 January 2014

Keywords:
EMD
EEMD
Time
Space
Complexity

ABSTRACT

It has been claimed that the empirical mode decomposition (EMD) and its improved version the ensemble EMD (EEMD) are computation intensive. In this study we will prove that the time complexity of the EMD/EEMD, which has never been analyzed before, is actually equivalent to that of the Fourier Transform. Numerical examples are presented to verify that EMD/EEMD is, in fact, a computationally efficient method.

1. Introduction

EMD [1] is a nonlinear and nonstationary time domain decomposition method. It is an adaptive, data-driven algorithm that decomposes a time series into multiple empirical modes, known as intrinsic mode functions (IMFs). Each IMF represents a narrow band frequency–amplitude modulation that is often related to a specific physical process. For signals with intermittent oscillations, one intrinsic mode can comprise oscillations with a variety of wavelengths at different temporal locations. Collectively, the simultaneous exhibition of these disparate oscillations is known as the mode mixing phenomenon, which can complicate analyses and obscure physical meanings. To overcome this problem, the EEMD algorithm [2] and the noise-assisted MEMD [3] have been proposed. In particular, the EEMD has drawn a lot of attention

Abbreviations: EMD, empirical mode decomposition; EEMD, ensemble empirical mode decomposition; IMFs, intrinsic mode functions; MEMD, multivariate empirical mode decomposition; ADD, addition; MUL, multiplication; DIV, division; COMP, comparison; BFV, blood flow velocity signal.

* Correspondence to: Research Center for Adaptive Data Analysis/National Central University, No. 300, Jhongda Rd., Jhongli City, Taoyuan County 32001, Taiwan, ROC. Tel.: +886 3 426 9734; fax: +886 3 426 9736.

E-mail address: mzlo@ncu.edu.tw (M.-T. Lo).

0378-4371/$ – see front matter
http://dx.doi.org/10.1016/j.physa.2014.01.020

Table 1
EMD.

$[c_m] = Main\ (NS, n_m, y_0)$

$r(t) = y_0(t)$
for $(m = 1 : n_m)\{$ % extract each IMF
  $x(t) = r(t)$ % $x(t)$ is proto-IMF
  for $(s = 1 : NS)\{$ % sifting iteration:
    $[t_{\max}, x_{\max}, t_{\min}, x_{\min}] = identify\_extrema(x)$;
    $U(t) = spline(t_{\max}, x_{\max})$; % Upper envelope
    $L(t) = spline(t_{\min}, x_{\min})$; % Lower envelope
    $x(t) = x(t) - (U(t) + L(t))/2$; % detail extraction (1)
  }
  $c_m(t) = x(t)$ (2)
  $r(t) = r(t) - x(t)$ (3)
}

during the past few years. During the last decade, the EMD/EEMD was shown to be more effective than the traditional Fourier method in many problems from various fields such as physics [4], biomedicine [5–8], mechanical health diagnosis [9], image analysis [10] and others.

It seems a common belief that a major drawback of the EMD/EEMD is that they require a long computation time. For example, in Ref. [11], the authors report that it takes more than an hour to perform EMD on a signal with 30,000 data points using a modern personal computer. The enormous amount of computation time hampers the applications of EMD/EEMD in analyzing long data and pseudo real-time hardware applications. For this reason, parallel computing such as Cuda [12], Open MP [13] and hardware implementations like VLSI [14] and FPGA [15] have been recently adopted to improve the execution time.

In this study, we proved that the EMD/EEMD is actually not computation intensive as has been claimed in the past. We first analyze the time and storage complexity for EMD/EEMD and their variants in different applications under an appropriate implementation, which is based on single core, sequential implementation without any approximations. Finally, numerical examples are presented to verify the performance of EMD/EEMD.

The remainder of this article is divided into the following sections: in Section 2, we briefly review the EMD method. In Section 3 we present an implementation of the EMD/EEMD method and then analyze their time and storage complexities. Section 4 illustrates the efficacy of the proposed method with experimental results and we conclude our study in Section 5.

2. The EMD algorithm

Given an input signal $y_0(t),\ y_0 \in R,\ t \in Z$ and $t = [1 : n]$, the EMD decomposes a signal $y_0(t)$ into a series of intrinsic mode functions (IMFs) which are extracted via an iterative sifting process. First, the local maxima and minima of the signal are connected, respectively, by cubic splines to form the upper/lower envelopes. The average of the two envelopes is then subtracted from the original signal. This sifting process is then repeated several times (define the number of siftings as *NS*, usually 10) to obtain the first IMF that oscillates at relatively higher frequencies than does the residual signal that is obtained by subtracting the IMF from the original signal. Then, the residual is deemed as the input for a new round of iterations. In turn, subsequent IMFs with lower oscillation frequencies are derived using the same process and the newly obtained residue. The result of the EMD is a decomposition of the signal $y_0(t)$ into the sum of the IMFs and a residue $r(t)$. That is,

$$y_0(t) = \sum_{m=1}^{n_m} c_m(t) + r(t)$$

where n_m is the number of IMFs. The EMD algorithm is listed in Table 1.

3. Time and space complexity

3.1. EMD

In this section, we propose an implementation of the EMD and analyze its time and space complexity. The arithmetic operations involved include addition (*ADD*), multiplication (*MUL*), division (*DIV*) and comparison (*COMP*). Fixed-point arithmetic operations are negligible compared with the floating-point operations. To facilitate the analysis, the EMD procedure is divided into the main, extrema identification and cubic spline procedures. Capital T and M denote the time and space (storage) complexity, respectively. The EMD algorithm is listed in Table 1 and is explained in detail below.

3.1.1. Main procedure

Each of the input, residue $r(t)$, proto-IMF $x(t)$ and the upper/lower envelope $U(t)/L(t)$ is of length n and requires n floating point storage. The output signal consists of n_m IMFs each with length of n, hence it requires $n_m \cdot n$ floating-

point storage. Then $M_{main} = (5 + n_m)n$ float. The detail extraction step in Eq. (1) is applied $NS \cdot n_m$ times and requires $(2ADD + 1MUL) \cdot n$ operations for each sifting. Typically $NS = O(10) \gg 1$ and the time complexity of Eqs. (1) and (3) are negligible compared with that of Eq. (1). Therefore,

$$T_{main} = (2ADD + 1MUL) \cdot NS \cdot n_m \cdot n.$$

3.1.2. Extrema identification procedure

The definition of a local maximum in the strict sense is adopted in this study, i.e.

$$(t_{\max})_k = t_j; \qquad (x_{\max})_k = x_j \quad \text{if } (x_j > x_{j-1} \text{ and } x_j > x_{j+1}). \tag{4}$$

Similar procedure is performed for the local minima. The brute force algorithm is applied to identify the maxima/minima, hence the time complexity is proportional to n. The comparison $x_j > x_{j-1}$ has been performed at the previous point and the information can thus be applied without extra work. The detection of maxima/minima requires $2COMP$ for each point and $T_{ext} = (2COMP) \cdot NS \cdot n_m \cdot n$. Since the number of maxima $n_u(m)$ and minima $n_l(m)$ are both less than data length n, $t_{\max}$ and $x_{\max}$ can be stored in an n integer and n floating array respectively. The same storage can be applied to the minima. Therefore, $M_{ext} = 2n\ float + 2n\ integer$.

3.1.3. Spline procedure

Once $(t_{\max}, x_{\max})$ is identified, the upper/lower envelope is connected by cubic spline interpolation, which is an established method [16], so we will only sketch the steps. Basically, suppose there are $n_u(m)$ maxima and $n_l(m)$ minima for each IMF m and the number of extrema is $n_e(m) = n_u(m) + n_l(m)$, then, for each point between two consecutive maxima $t_i \le t_j \le t_{i+1}$, the upper envelope is constructed using a third order polynomial. Thus, the envelope as a curve is piecewise polynomial of the third order. Define $\tau_j = t_j - t_i$, then

$$x(t) = A_i\tau_j^3 + B_i\tau_j^2 + C_i\tau_j + D_i. \tag{5}$$

Denoting $x''(t)$ as s and enforcing the continuity of x, x' and x'' at the maxima, s can be obtained by solving the following tridiagonal system of equations.

$$a_i s_{i-1} + b_i s_i + c_i s_{i+1} = d_i. \tag{6}$$

Define $h_i = t_i - t_{i-1}$, the matrix elements are found as

$$\begin{aligned} a_i &= h_i \\ c &= h_{i+1} \\ b_i &= 2(h_i + h_{i+1}) \\ d_i &= 6[(x_{i+1} - x_i)/h_{i+1} - (x_i - x_{i-1})/h_i]. \end{aligned} \tag{7}$$

After solving for s in Eq. (6), the coefficients A_i, B_i, C_i and D_i will be obtained through the following equations:

$$\begin{aligned} A_i &= (s_{i+1} - s_i)/(6h_i) \\ B_i &= 0.5s_i \\ C_i &= (x_{i+1} - x_i)/h_i - (2s_i + s_{i+1})h_i/6 \\ D_i &= x_i. \end{aligned} \tag{8}$$

Finally, the upper envelop $U(t_j)$ is calculated in the interval i according to Eq. (5). In summary, the first step for cubic spline interpolation is to determine the tridiagonal matrix elements and the vector on the right hand side of Eq. (6), using Eq. (7). The second step is to solve Eq. (6), which has n_u unknowns and $n_u - 2$ equations. Two more equations at the left and right boundaries need to be specified in order to make the problem well-posed, and the Not-a-knot boundary conditions are chosen in this study. The third step is to compute the polynomial coefficients A_i, B_i, C_i and D_i using Eq. (8). Then, the final step is to calculate the upper envelope with Eq. (5) for each point j in interval i. The complexity of each step is analyzed in detail as follows.

3.1.3.1. Elements of the tridiagonal matrix. The calculations of elements a_i, b_i and c_i only involve fixed-point calculations and therefore can be neglected. The term $(x_{i+1} - x_i)/h_i$ has been calculated at the previous point and does not need to be re-computed. If $(s/6)$ is used in place of s for the variables in Eq. (6), the calculation of d_i then requires $(2ADD + 1DIV)$, since the number of maxima and the number of minima differ at most by one and the time complexity of calculating the lower envelope is identical to that of the upper envelope. The same procedure is repeated for each sifting and IMF, therefore $T_{element} = (2ADD + 1DIV) \cdot NS \cdot \sum_{m=1}^{n_m} n_e(m)$.

3.1.3.2. Tridiagonal solver. The tridiagonal matrix Eq. (6) is diagonal dominant and well-conditioned such that no pivoting is required. The popular Thomas algorithm [17], a simplified form of Gaussian elimination, is the most economical way to solve it. The time complexity of the Thomas algorithm is linear and equal to $(3ADD + 5MUL + 1DIV) \cdot n_u$ for the upper envelope. Therefore, $T_{sol} = (3ADD + 5MUL + 1DIV) \cdot NS \cdot \sum n_e(m)$. Because the number of maxima $n_u \leq n$, we can store a_i, b_i, c_i, d_i and s_i as a floating array of length n. Therefore, $M_{sol} = 5n$ float.

3.1.3.3. Cubic spline polynomial coefficient. If the variable $(s/6)$ is used in lieu of s in Eqs. (6)–(8), then the calculation of A_i, B_i, C_i and D_i requires $(4ADD + 5MUL + 1DIV)$. Therefore, $T_{coef} = (4ADD + 5MUL + 1DIV) \cdot NS \cdot \sum n_e(m)$.

3.1.3.4. Cubic polynomial interpolation. After obtaining the polynomial coefficients in Eq. (8), we interpolate the upper/lower envelope for each point j in $t_i \leq t_j \leq t_{i+1}$ using Eq. (5). Since the calculations of τ^2 and τ^3 are necessary, Eq. (5) requires $(3ADD + 5MUL)$ for each j.

Here we propose novel recursive relations of $U(t_i)$ to further optimize the spline interpolation in Eq. (5). Define $\tau_k = \tau - k$ and rewrite $\tau = \tau_k + k$, then $U(\tau)$ can be expressed as

$$U(\tau) = U(\tau - 1) + f(\tau - 1) \tag{9A}$$

where $f(\tau_1)$ is a second order polynomial as follows:

$$f(\tau_1) = 3A_i\tau_1^2 + (3A_i + 2B_i)\tau_1 + (A_i + B_i + C_i). \tag{9B}$$

Likewise, $f(\tau_1)$ can be written as the sum of $f(\tau_2)$ and a first order polynomial $g(\tau_2)$ and $g(\tau_2)$ can be written as the sum of $g(\tau_3)$ and a constant p_0.

$$f(\tau - 1) = f(\tau - 2) + g(\tau - 2) \tag{10A}$$
$$g(\tau_2) = 6A_i\tau_2 + (6A_i + 2B_i) \tag{10B}$$
$$g(\tau - 2) = g(\tau - 3) + p_0 \tag{11A}$$
$$p_0 = (6A_i + 2B_i). \tag{11B}$$

In summary, the upper envelope $U(\tau)$ for $j \geq 3$ is calculated through the following recursive relation.

$$\begin{aligned} g(\tau - 2) &= g(\tau - 3) + p_0 \\ f(\tau - 1) &= f(\tau - 2) + g(\tau - 2) \\ U(\tau) &= U(\tau - 1) + f(\tau - 1). \end{aligned} \tag{12}$$

The initial conditions $U(0)$, $U(1)$ and $U(2)$ are required to initialize Eqs. (12) which are computed through Eq. (5). The terms $g(0)$ and $f(1)$ are required in computing Eq. (12) for $j = 3$. After side calculation, we obtain the following

$$\begin{aligned} U(0) &= D_i \\ U(1) &= A_i + B_i + C_i + D_i \\ U(2) &= g(0) + 2U(1) - D_i \\ g(0) &= 6A_i + 2B_i \\ f(1) &= g(0) + U(1) - D_i. \end{aligned} \tag{13}$$

We summarize the steps in calculating the upper envelope as follows.

(1) Compute $U(0)$, $U(1)$, $U(2)$, $g(0)$ and $f(1)$ through Eq. (13).
(2) Compute Eq. (12) for $j \geq 3$.

The calculation of Eq. (12) for each j requires $3ADD$ without any MUL operations. Five multiplication operations are spared, compared with applying Eq. (5) directly. The calculation of Eq. (13) for each i requires $8ADD + 3MUL$. Therefore, $T_{interp} = [(6ADD)n_m n + (5ADD + 3MUL) \sum n_e(m)] \cdot NS$. The time complexity for each step in EMD is summarized in Table 2. It is worth noting that the spline interpolation using Eqs. (12) and (13) can be easily extended to fixed point computation. Suppose the dynamic range of τ can reach 2^b and is stored in b bits. A direct application of Eq. (5) will then result in that $\max(\tau^3) = b^3$ and τ^3 must be stored in $3b$ bits. Thus, $2b$ extra bits are required to store τ^3. For example, if $b = 8$, $b^3 = 24$, then τ^3 must be stored in 24 bits instead of 8 bits to maintain the precision of the polynomial, hence it requires more storage and computation time. On the other hand, applying Eq. (12) will circumvent this difficulty.

The total storage for the EMD algorithm is obtained by summing up all steps in Table 2. Without loss of generality, we can assume $M_{int} = M_{float}/2$. Then,

$$M = (13 + n_m) \cdot n \text{ float}. \tag{14}$$

Since it is known that the EMD behaves as a dyadic filter bank [18] the number of IMFs is pre-assigned to be [2,19]

$$n_m = \log_2 n. \tag{15}$$

Table 2
Time and space complexity for each step.

Function	T	M
main	$(2ADD + 1MUL) \cdot NS \cdot n_m \cdot n$	$(5 + n_m)n$ *float*.
extrema	$(2COMP) \cdot NS \cdot n_m \cdot n$	$2n$ *float* $+ 2n$ *integer*.
matrix elements	$(2ADD + 1DIV) \cdot NS \cdot \sum n_e(m)$	
Tridiagonal solver	$(3ADD + 5MUL + 1DIV) \cdot NS \cdot \sum n_e(m)$	$5n$ *float*.
spline Coefficient	$(4ADD + 5MUL + 1DIV) \cdot NS \cdot \sum n_e(m)$.	
polynomial Interpolation	$[(6ADD)n_m n + (5ADD + 3MUL) \sum n_e(m)]$	

Table 3
The EEMD algorithm.

$[c_m] = \text{EEMD}(NE, NS, n_m, \varepsilon, y_0)$
  $c_m(t) = 0$; % initialize *IMF*
  for ($trial = 1 : NE$) {
    $y(t) = y_0(t) + \varepsilon w_k(t)$; % add white noise w to input signal
    $\tilde{c}_m(t) = \text{EMD}(NS, n_m\, y)$
    $c_m(t) = \tilde{c}_m(t) + x(t)$
  }
  $c_m(t) = c_m(t)/NE$; % ensemble average

The space complexity for extracting all IMFs is then obtained by substituting n_m in Eq. (15) into Eq. (14).

$$M = (13 + \log_2 n) \cdot n \text{ float}. \tag{16}$$

The CPU time for the *ADD*, *MUL*, *DIV* and *COMP* depends on the hardware used. In general, *DIV* takes much longer time than others. To simplify the analysis, we assume that all operations require the same number of flops. The total operation count is then simplified to

$$T(n) = NS \cdot \left[11n_m \cdot n + 30 \sum n_e(m)\right]. \tag{17}$$

Eq. (17) consists of terms involving *NS* and, in particular, $n_e(m)$, which still needs to be determined. The upper bound of $n_e(m)$ is n, then the time complexity for extracting n_m IMF is

$$T(n) = 41 \cdot NS \cdot n_m \cdot n. \tag{18}$$

As discussed in Refs. [2,19], a constant *NS* will generally produce better results than using other stoppage criteria in the EMD algorithm. *NS* about 10 [2,19] would render EEMD an almost perfect dyadic filter for noise while keeping the upper and lower envelopes of IMFs almost symmetric with respect to the zero line. If the chosen sifting number is too large, the EMD procedure then tends to over-decompose a physical component, which would contaminate other parts of the signal and spread out its component over adjacent IMFs [20]. Based on the above discussion and using Eqs. (15) and (18), the time complexity, with a fixed *NS*, for extracting all IMFs is given as

$$T(n) \leq 41NS \cdot n(\log_2 n) = O(n \log n). \tag{19}$$

Eq. (19) shows that the time complexity of the EMD is $O(n \log n)$, which is equal to that of FFT. In practice, the upper bound of $n_e(m)$ can be further reduced as follows. The EMD acts as a dyadic filter bank for a broadband white noise that has maximum number of extrema for each IMF. In this case, the number of extrema reduces approximately by half as the IMF number m increases by one. That is, $n_e(m) \approx 2n_e(m + 1)$. In addition, since $n_e(m = 1) \leq n$, it follows that $\sum n_e(m) \leq 2n$ and Eq. (17) can be further reduced to

$$T(n) \leq (11n_m + 60) \cdot NS \cdot n. \tag{20}$$

3.2. EEMD

The EEMD first generates an ensemble of data sets by adding different realizations of a white noise with finite amplitude ε to the original data. EMD analysis is then applied to each data series of the ensemble. Finally, the IMFs are obtained by averaging the respective components in each realization over the ensemble. The EEMD algorithm is listed in Table 3.

In Table 3, $\varepsilon = \varepsilon_0 std(y_0)$ and ε_0 is the input noise level, std stands for the standard deviation and *NE* is the ensemble number. The averaging effect of the assisted white noise—the residual noise f will decrease as [2,19]

$$\varepsilon_f = \varepsilon/\sqrt{NE}. \tag{21}$$

In Eq. (21), ε_f depends on *NE* only and is unrelated to the data length n since EMD is a local algorithm [1]. Theoretically, *NE* should approach infinity in order to smooth out the assisted white noise. In practice, ε_0 is chosen in the range of 0.1–0.4,

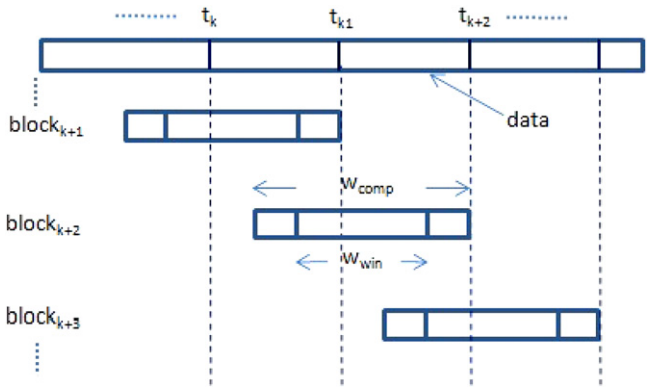

Fig. 1. The diagram of the on-line EEMD algorithm.

Table 4
Comparison of the time complexities of EMD and its variants.

Algorithm	T
EMD	$NS \cdot (41n_m \cdot n)$
EEMD	$NE \cdot NS \cdot (41n_m \cdot n)$
Real time EEMD	$(1+2OP) \cdot NE \cdot NS \cdot (41n_m \cdot n)$
MEEMD	$41(1+n_m)n_m \cdot NE \cdot NS \cdot n$

and NE of the order of 100 will generally produces satisfactory results and render the residual noise less than a fraction of 1% of error. Based on Table 3, the space complexity of EEMD is equal to that of the EMD and the time complexity of EEMD is equal to that of EMD multiplied by NE. That is,

$$T_{EEMD} = NE \cdot NS \cdot (41n_m \cdot n). \tag{22}$$

3.3. Real-time EEMD

The EEMD can be modified to (pseudo-) real time application by extending the off-line EEMD to its on-line version [14] by partitioning the data series into windows W_{win}, each sub-series thus generated would then be processed by EEMD, sequentially, from left to right. To ensure the continuity of the decomposed IMFs, the computation window, W_{comp}, is thus defined by extending W_{win} on both sides with an overlap percentage OP, where $OP \geq 0$, as demonstrated in Fig. 1.

$$W_{comp} = (1 + 2OP) \cdot W_{win}\,. \tag{23}$$

Neglecting the two boundary windows and making use of Eq. (22) and the above equation, it then follows that $T_{online\_EEMD} = (n/W_{win}) \cdot [(1 + 2OP)NE \cdot NS \cdot 41n_m \cdot W_{win}]$ and is simplified to

$$T_{online\_EEMD} = (1 + 2OP) \cdot NE \cdot NS \cdot (41n_m \cdot n). \tag{24}$$

Eq. (24) indicates that the order of the time complexity of the on-line EEMD is equal to that of the EEMD but the computational time of the on-line EEMD is slightly increased by the factor of $(1+2OP)$. It is worth noting that since Eq. (22) is linear with respect to n, applying the on-line EEMD would not accelerate the computation in terms of the total computation time that is needed.

3.4. MEEMD

The multi-dimensional ensemble empirical mode decomposition (MEEMD) [10] is the multi-dimension version of EEMD. Here we will restrict our consideration to the two-dimensional image applications. The MEEMD decomposes an image into many one-dimensional sub-problems. A one-dimensional EEMD is then applied to each sub-problem. The image is assumed to have n pixels, with $n = R_x R_y$ where R_x and R_y are the resolution in the x and the y directions, respectively. In order not to repeat a similar but more tedious derivation, we will directly state the time complexity of the MEEMD (see the appendix of Ref. [10] for the detailed steps)

$$T_{MEEMD} = 41 \cdot NE \cdot NS \cdot (1 + n_m)n_m \cdot n \approx n_m^2 \cdot O(n). \tag{25}$$

The time complexities of the EMD and its variants is listed in Table 4.

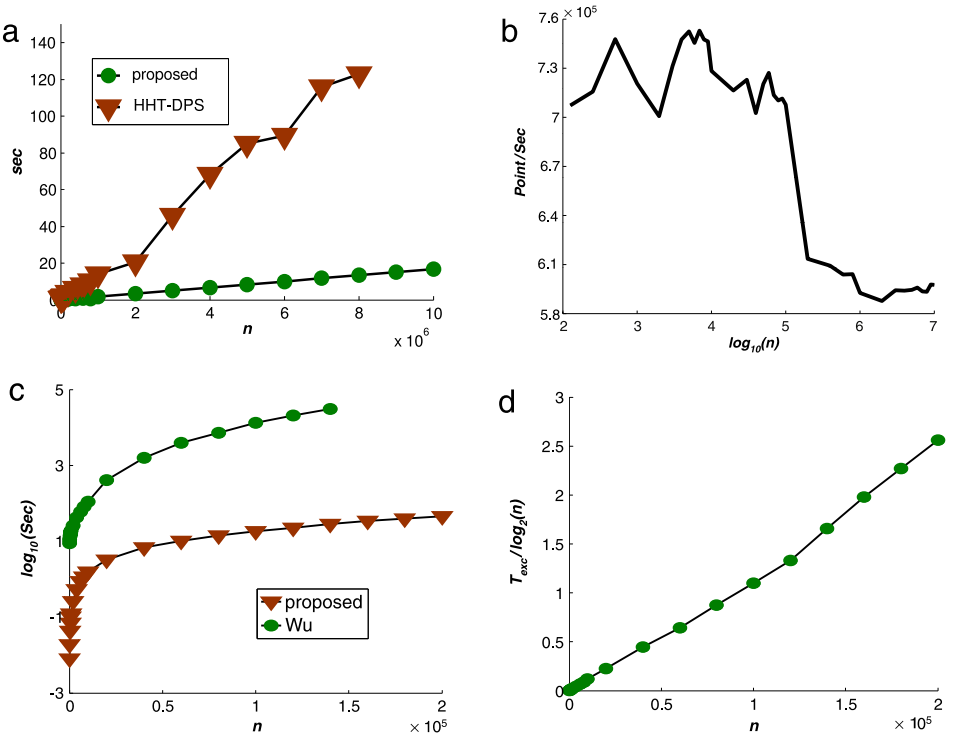

Fig. 2. The execution time for a randomly distributed white noise with $NS = 10$. (a) The computation time T_{exc} of EMD by Ref. [21] and the proposed method with $n_m = 10$. (b) The computation speed (points/s) versus $\log_{10} n$ of the proposed method with the parameters as in (a). (c) T_{exc} of EEMD by Ref. [2] and the proposed implementation with $NE = 100$, $\varepsilon = 0.2$ and $n_m = \log_2 n$. (d) $T_{exc}/\log_2 n$ of the proposed method with the parameters as in (c).

4. Numerical experiments

In this section, we perform a series of numerical experiments to verify our analysis of the time and space complexity of the EMD, EEMD, MEEMD and real-time EEMD as discussed in the previous section. We also evaluate the performance of the proposed implementation and investigate its feasibility for real-time computation. The performance can be evaluated by measuring the computation speed defined as $S = n/T_{exc}$, where T_{exc} is the execution time. The proposed method is implemented in sequential C language with double precision using a Matlab interface (http://rcada.ncu.edu.tw/research1.htm). Numerical experiments are conducted on a laptop computer with 2.4 GHz CPU, 4 GB memory and 3 M RAM running Windows 7.

In the first example, we test the execution time T_{exc} of EMD for a uniformly distributed random noise. Fig. 2(a) plots T_{exc} versus data length n ranging from 128 to 10^7 with parameters $n_m = NS = 10$. As Fig. 2(a) shows, the computation time of the proposed method for a fixed number of IMFs is linear w.r.t. n and confirms Eq. (18). Fig. 2(b) plots S_{EMD} versus $\log_{10} n$. The graph indicates that S_{EMD} is ≈0.7 M points/s for $n < 10^5$ and decreases to 0.6 M points/s for $n \geq 10^5$. Theoretically, Eq. (18) predicts that the computation speed is invariant for all n. However, because of the limited computer cache size, the data is moved between RAM and cache and thus S_{EMD} slightly deviates from being a constant.

Next, we compare the execution performance of the proposed EMD implementation and the NASA HHP-DPS [21], which was also implemented in the sequential C language. Fig. 2(a) shows that the proposed method is about 10 times faster than the NASA HHP-DPS. The HHP-DPS program fails to execute when $n > 8 \cdot 10^6$, whereas the proposed program can execute up to $n = 10^7$. We now look at the execution performance of EEMD for extracting all IMFs. The performances of the proposed program and the Matlab program provided in Ref. [2] are shown in Fig. 2(c). These results demonstrate that for $n < 1.5 \cdot 10^4$ the proposed EEMD implementation is 10^2–10^3 times faster than the Matlab program. Fig. 2(d) depicts the $T_{exc}/\log_2 n$ of the proposed method to be approximately linear, which confirms that the proposed method is an $O(n \log n)$ algorithm as predicted by Eq. (19), taking account of Eq. (15).

In the second example, we evaluate the performance of the proposed method using real-world data. Here we test the execution time of EEMD for a blood flow velocity signal (BFV) using transcranial color Doppler measurement [22] with $n = 15\,000$, $NE = 500$, $NS = 10$ and $\varepsilon = 0.2$. Fig. 3(a) plots T_{exc} versus n_m and indicates $S_{EEMD} \approx 1.7 \cdot 10^3$ point/s. The corresponding results for white noise are also presented. As demonstrated in Fig. 3(a), S_{EEMD} for the BFV case is faster than that for the white noise. This is mainly due to the greater number of extrema in the white noise than in the BFV signal for each IMF, and hence a longer computation time is needed in the former case. Fig. 3(b) shows the profiling of different steps in EEMD with $n_m = 10$. To detail, the extraction step in Eq. (1) involves a DO LOOP with only 2*ADD* and 1*MUL* and

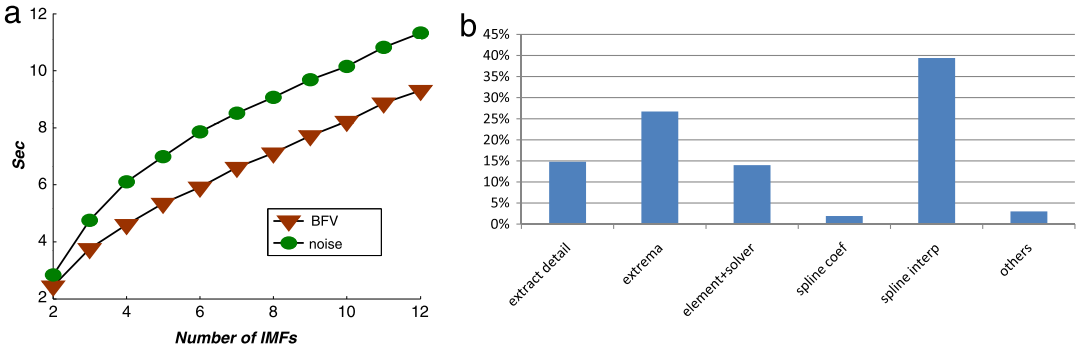

Fig. 3. The execution time of a BFV signal by EEMD with $n = 15\,000$, $NS = 10$, $NE = 500$, $\varepsilon = 0.2$ and $n_m = 10$. (a) T_{exc}. (b) The profiling of EEMD in different steps.

Table 5
Image decomposed by MEEMD with $NE = 100$, $n_m = 6$ and $NS = 10$.

Size ($R_x \times R_y$)	# of EMD applied	s
96×96	67 200	6.50
128×128	89 600	11.36
256×256	179 200	43.4
512×512	358 400	190.16

occupies ≈15% of the total execution time. Additionally, the tridiagonal solver plus the matrix element occupies only 13% of the total computation time, thus confirming the efficacy of the Thomas algorithm.

In the third example, we test the performance of the proposed method in handling real-world huge data sets. The EEMD is applied to a one channel overnight (8.8 h) sleep EEG (electroencephalography) signal with $n = 8.1 \cdot 10^6$, $n_m = 10$, $NE = 100$, $NS = 10$ and $\varepsilon = 0.2$. In this example, $S_{EEMD} \approx 7.5 \cdot 10^3$ and the execution time is $T_{exc} \approx 1080$ s, which demonstrates the efficacy of EEMD.

In the fourth example, we test the performance of the proposed method in imaging applications using MEEMD. An image with $n = R_x \times R_y$ of a white noise with parameters $NE = 100$, $n_m = 6$, $NS = 10$ and $\varepsilon = 0.2$ is chosen. The computation times T_{exc} and the number of applied one-dimensional EMDs of different image sizes are shown in Table 5. The computation speed is about $S_{EEMD} = 1.2 \cdot 10^4$ and the total computation time is proportional to data size n.

In the final example, we investigate the feasibility of using the proposed implementation in real-time EEMD applications. For real-time computation the execution time in the computation window must be less than that of the data window, i.e.

$$T_{exc} = sf \cdot W_{comp}/S_{EEMD} \leq W_{win}$$

where sf is the sampling frequency. Substituting Eq. (23) into this equation yields

$$S_{EEMD} \geq (1 + 2OP) \cdot sf. \tag{26}$$

The choices of the parameters depend on the specific application. For typical parameters such as $OP = 50\%$, $NS = 10$, $NE = 100$ and $n_m = 10$, the computation speed S_{EEMD} is about $7 \cdot 10^3$. Thus, real-time computation is achieved at a sample rate less than 3500 Hz.

5. Conclusion

In this study, we prove, for the first time, that the time complexity of EMD/EEMD is $T = 41 \cdot NE \cdot NS \cdot n(\log_2 n) = O(n \log n)$, where n is the data length and the parameters NS and NE are the sifting and ensemble numbers respectively. Therefore, the time complexity of the EMD/EEMD is equivalent to the traditional Fourier transform but with a larger factor. The space complexity is $M = (13 + \log_2 n) \cdot n = O(n \cdot \log n)$ floating points. We tested and performed the EMD/EEMD implementation on a laptop computer with 2.4 GHz CPU. The computer program executes EMD successfully with data length up to 10 million at 0.6–0.7 million points per second with 10 sifting numbers and 10 IMFs. Also, the EEMD algorithm can be executed in real-time for $NE = 100$ at a sampling rate within 3500 Hz. With this level of performance, it becomes clear that the EMD/EEMD is a computationally efficient method and can be applied to a much larger class of scientific and engineering problems.

Acknowledgments

M-T Lo was supported by NSC (Taiwan, ROC), Grant No. 100-2221-E-008-008-MY2, joint foundation of CGH and NCU, Grant Nos. CNJRF-101CGH-NCU-A4, VGHUST102-G1-2-3 and NSC support for the Center for Dynamical Biomarkers and Translational Medicine, National Central University, Taiwan (NSC 101-2911-I-008-001).

References

[1] N.E. Huang, Z. Shen, S.R. Long, M.C. Wu, H.H. Shih, Q. Zheng, N.-C. Yen, C.C. Tung, H.H. Liu, The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 454 (1998) 903–995.
[2] Z. Wu, N.E. Huang, Ensemble empirical mode decomposition: a noise-assisted data analysis method, Adv. Adapt. Data Anal. 1 (2009) 1–41.
[3] D. Mandic, Filter bank property of multivariate empirical mode decomposition, IEEE Trans. Signal Process. 59 (2011) 2421–2426.
[4] Amir Bashan, et al., Comparison of detrending methods for fluctuation analysis, Physica A 387 (21) (2008) 5080–5090.
[5] Kun Hu, et al., Altered phase interactions between spontaneous blood pressure and flow fluctuations in type 2 diabetes mellitus: nonlinear assessment of cerebral autoregulation, Physica A 387 (10) (2008) 2279–2292.
[6] M.T. Lo, L.Y. Lin, W.H. Hsieh, P.C.I. Ko, Y.B. Liu, C. Lin, Y.C. Chang, C.Y. Wang, V.H.W. Young, W.C. Chiang, J.L. Lin, W.J. Chen, M.H.M. Ma, A new method to estimate the amplitude spectrum analysis of ventricular fibrillation during cardiopulmonary resuscitation, Resuscitation 84 (11) (2013) 1505–1511.
[7] M.T. Lo, K. Hu, Y.H. Liu, C.K. Peng, V. Novak, Multimodal pressure–flow analysis: application of Hilbert–Huang transform in cerebral blood flow regulation, EURASIP J. Adv. Signal Process. (2008) 785243.
[8] M.T. Lo, V. Novak, C.K. Peng, Y.H. Liu, K. Hu, Nonlinear phase interaction between nonstationary signals: a comparison study of methods based on Hilbert–Huang and Fourier transforms, Phys. Rev. E 79 (6) (2009) 1–11. Article 061924.
[9] Jian Zhang, et al., Performance enhancement of ensemble empirical mode decomposition, Mech. Syst. Signal Process. 24 (7) (2010) 2104–2123.
[10] Z. Wu, E.H. Norden, X. Chen, The multi-dimensional ensemble empirical mode decomposition method, Adv. Adapt. Data Anal. 1 (2009) 339–372.
[11] L.C. Wu, H.H. Chen, J.T. Horng, C. Lin, N.E. Huang, Y.C. Cheng, K.F. Cheng, A novel preprocessing method using Hilbert Huang transform for MALDI-TOF and SELDI-TOF mass spectrometry data, PLoS One 5 (2010) e12493.
[12] D. Chen, L. Wang, G. Ouyang, X. Li, Massively parallel neural signal processing on a many-core platform, Comput. Sci. Eng. 13 (6) (2011) 42–51.
[13] D. Chen, D. Li, M. Xiong, H. Bao, X. Li, GPGPU-aided ensemble empirical-mode decomposition for EEG analysis during anesthesia, IEEE Trans. Inf. Technol. Biomed. 14 (2010) 1417–1427.
[14] T.C. Chen, Y.Y. Chen, T.C. Ma, L.G. Chen, Design and implementation of cubic spline interpolation for spike sorting microsystems, in: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2011, pp. 1641–1644.
[15] M.H. Lee, K.K. Shyu, P.L. Lee, C.M. Huang, Y.J. Chiu, Hardware implementation of EMD using DSP and FPGA for online signal processing, IEEE Trans. Ind. Electron. 58 (2011) 2473–2481.
[16] P.O.W. Curtis, F. Gerald, Applied Numerical Analysis, Addison-Wesley Publishing Company, New York, 1989.
[17] L.H. Thomas, Elliptic problems in linear difference equations over a network, in: Watson Sci. Comput. Lab. Rept., Columbia University, New York, 1949.
[18] P. Flandrin, G. Rilling, P. Goncalves, Empirical mode decomposition as a filter bank, IEEE Signal Process. Lett. 11 (2004) 112–114.
[19] Z. Wu, N.E. Huang, On the filtering properties of the empirical mode decomposition, Adv. Adapt. Data Anal. 2 (4) (2010) 397–414.
[20] G. Rilling, P. Flandrin, P. Gonçalvés, On empirical mode decomposition and its algorithms, in: IEEE-EURASIP Workshop on Nonlinear Signal and Image Processing, NSIP-03, Grado (I), 2003.
[21] S. Kizhner, T.P. Flatley, N.E. Huang, K. Blank, E. Conwell, On the Hilbert–Huang transform data processing system development, in: Aerospace Conference, 2004. Proceedings. 2004 IEEE, 2004.
[22] K. Hu, M.T. Lo, C.K. Peng, Y. Liu, V. Novak, A nonlinear dynamic approach reveals a long-term stroke effect on cerebral blood flow regulation at multiple time scales, PLoS Comput. Biol. 8 (2012) e1002601.