Enhanced Karnik–Mendel Algorithms

Dongrui Wu, *Student Member, IEEE*, and Jerry M. Mendel, *Life Fellow, IEEE*

***Abstract*—The Karnik–Mendel (KM) algorithms are iterative procedures widely used in fuzzy logic theory. They are known to converge monotonically and superexponentially fast; however, several (usually two to six) iterations are still needed before convergence occurs. Methods to reduce their computational cost are proposed in this paper. Extensive simulations show that, on average, the enhanced KM algorithms can save about two iterations, which corresponds to more than a 39% reduction in computation time. An additional (at least) 23% computational cost can be saved if no sorting of the inputs is needed.**

***Index Terms*—Enhanced Karnik–Mendel (EKM) algorithms, interval type-2fuzzy sets (IT2FSs), Karnik–Mendel (KM) algorithms, novel weighted averages, type-reduction, uncertainty measures.**

I. Introduction

THE FOLLOWING problem is frequently met in (but not limited to) fuzzy logic theory:

Given

$$x_i \in X_i \equiv [\underline{x}_i, \overline{x}_i], \qquad i = 1, 2, \ldots, N \tag{1}$$

$$w_i \in W_i \equiv [\underline{w}_i, \overline{w}_i], \qquad i = 1, 2, \ldots, N \tag{2}$$

where

$$\underline{x}_i \le \overline{x}_i, \qquad i = 1, 2, \ldots, N \tag{3}$$

$$\underline{w}_i \le \overline{w}_i, \qquad i = 1, 2, \ldots, N \tag{4}$$

compute

$$Y = \frac{\sum_{i=1}^{N} X_i W_i}{\sum_{i=1}^{N} W_i} \equiv [y_l, y_r] \tag{5}$$

where

$$y_l = \min_{\substack{\forall x_i \in [\underline{x}_i, \overline{x}_i] \\ \forall w_i \in [\underline{w}_i, \overline{w}_i]}} \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i} \tag{6}$$

$$y_r = \max_{\substack{\forall x_i \in [\underline{x}_i, \overline{x}_i] \\ \forall w_i \in [\underline{w}_i, \overline{w}_i]}} \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i}. \tag{7}$$

Places where this problem occurs are the following.

1) *Computing uncertainty measures for interval type-2 fuzzy sets (IT2 FSs):*
   a) In computing the centroid of an IT2 FS $\tilde{A}$ [2], [5], [8], $\underline{x}_i = \overline{x}_i = z_i$ (see Fig. 1) represent discretizations of the primary variable z, and interval $[\underline{w}_i, \overline{w}_i]$ is the membership grade[1] of z_i. Y is the centroid of $\tilde{A}$.
   b) In computing the variance of an IT2 FS $\tilde{A}$ [13], $\underline{x}_i = \overline{x}_i = [z_i - c(\tilde{A})]^2$, where $c(\tilde{A})$ is the center of the centroid of $\tilde{A}$, and interval $[\underline{w}_i, \overline{w}_i]$ is the membership grade of z_i. Y is the variance of $\tilde{A}$.
   c) In computing the skewness of an IT2 FS $\tilde{A}$ [13], $\underline{x}_i = \overline{x}_i = [z_i - c(\tilde{A})]^3$, and interval $[\underline{w}_i, \overline{w}_i]$ is the membership grade of z_i. Y is the skewness of $\tilde{A}$.
2) *Type-reduction*[2]*:*
   a) In centroid and center-of-sums type-reduction of IT2 fuzzy logic systems (FLSs) [8], an IT2 FS is first obtained by combining the output sets for fired rules, after which, computing the type-reduced set is equivalent to computing the centroid of that IT2 FS, as in item a) of 1). Y is the type-reduced set.
   b) In center-of-sets type-reduction of IT2 FLSs [8], X_i represents the centroid of the consequent IT2 FS of the ith rule, and W_i is the firing level of that rule. Y is the type-reduced set.
   c) In height type-reduction of IT2 FLSs [8], $\underline{x}_i = \overline{x}_i$ represents the point having maximum membership in the consequent type-1 FS of the ith rule, and W_i is the firing level of that rule. Y is the type-reduced set. The operations in modified height type-reduction [8] are quite similar, except that W_i is multiplied by a scale factor.
3) *Computing novel weighted averages*[3] *(NWAs):*
   a) In computing the interval weighted average (IWA) [6], X_i's are input signals, and W_i's are their associated weights, both of which are interval sets. Y, which is also an interval set, is the IWA.

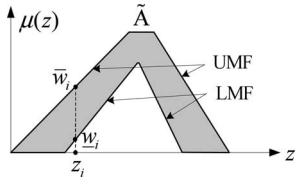


Fig. 1. IT2 FS. UMF: upper membership function; LMF: lower membership function. The shaded region is called a footprint of uncertainty (FOU).

Manuscript received May 25, 2007; revised October 12, 2007; accepted October 22, 2007. First published April 30, 2008; current version published July 29, 2009.

The authors are with the Signal and Image Processing Institute, Ming Hsieh Department of Electrical Engineering, University of Southern California, Los Angeles, CA 90089-2564 USA (e-mail: dongruiw@usc.edu; mendel@sipi.usc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TFUZZ.2008.924329

[1]The lower and upper memberships of z_i are usually denoted as $\underline{\mu}(z_i)$ and $\overline{\mu}(z_i)$, respectively [8]. To be consistent with (6) and (7), in this paper, we denote them as $\underline{w}_i$ and $\overline{w}_i$, respectively.

[2]Type-reduction for general type-2 fuzzy logic systems can be computed by using the α-plane concept [3]. The EKM algorithms can also be used for it; however, we will not go into the details of this because interval type-2 fuzzy sets and systems are so far much more popular.

[3]NWAs are weighted averages in which at least one of the weights are novel models [6], i.e., intervals, type-1 FSs, or IT2 FSs.

b) In computing the fuzzy weighted average (FWA) [4], [6], X_i's and W_i's are α-cuts on the input signals and the weights, both of which are type-1 FSs. Y is the corresponding α-cut on the FWA.
c) In computing the linguistic weighted average (LWA) [6], [12], X_i and W_i are α-cuts of the upper membership functions (UMFs) [lower membership functions (LMFs)] of the inputs signals and the weights, both of which are IT2 FSs. Y is the corresponding α-cut on the UMF (LMF) of the LWA, which is also an IT2 FS.

A slightly modified problem is encountered in the prey model from foraging theory [11], which [10] "*describes an agent searching for tasks of different types in a particular environment*." The agent makes a decision on which prey types to catch by maximizing its energy gain [10] J, where

$$J = \frac{\sum_{i=1}^{n} p_i \lambda_i v_i}{1 + \sum_{i=1}^{n} p_i \lambda_i e_i} \tag{8}$$

in which $p_i \in [0, 1]$ is the probability of catching prey type i if it is encountered, λ_i is the average rate of encounter with prey type i, v_i is the reward obtained from successfully catching prey type i, and e_i is the expected time required to catch prey type i. The agent needs to find the maximum of J, i.e.,

$$J_{\max} = \max_{\forall p_i \in [0,1]} \frac{\sum_{i=1}^{n} p_i \lambda_i v_i}{1 + \sum_{i=1}^{n} p_i \lambda_i e_i} \tag{9}$$

and then catches the prey types corresponding to $p_i = 1$. Equation (9) can be viewed as a special case of (7). To see that, rewrite $J_{\max}$ as

$$J_{\max} = \max_{\forall p_i \boldsymbol{\lambda}_i e_i \in [0, \boldsymbol{\lambda}_i e_i]} \frac{\sum_{i=1}^{n} (v_i/e_i) p_i \lambda_i e_i}{1 + \sum_{i=1}^{n} p_i \lambda_i e_i}. \tag{10}$$

Observe that v_i/e_i and $p_i \lambda_i e_i$ in (10) play the roles of $\overline{x}_i$ and w_i in (7), respectively. Though there is an extra constant in the denominator of (10), the basic procedure to compute $J_{\max}$ is essentially the same as that for y_r.

It is well known that y_l and y_r in (6) and (7) can be expressed as [8] (the derivation is given in Appendix A)

$$\begin{aligned} y_l &= \min_{\forall w_i \in [\underline{w}_i, \overline{w}_i]} \frac{\sum_{i=1}^{N} \underline{x}_i w_i}{\sum_{i=1}^{N} w_i} \\ &= \frac{\sum_{i=1}^{L} \underline{x}_i \overline{w}_i + \sum_{i=L+1}^{N} \underline{x}_i \underline{w}_i}{\sum_{i=1}^{L} \overline{w}_i + \sum_{i=L+1}^{N} \underline{w}_i} \end{aligned} \tag{11}$$

$$\begin{aligned} y_r &= \max_{\forall w_i \in [\underline{w}_i, \overline{w}_i]} \frac{\sum_{i=1}^{N} \overline{x}_i w_i}{\sum_{i=1}^{N} w_i} \\ &= \frac{\sum_{i=1}^{R} \overline{x}_i \underline{w}_i + \sum_{i=R+1}^{N} \overline{x}_i \overline{w}_i}{\sum_{i=1}^{R} \underline{w}_i + \sum_{i=R+1}^{N} \overline{w}_i} \end{aligned} \tag{12}$$

where L and R are the switch points satisfying

$$\underline{x}_L \le y_l \le \underline{x}_{L+1} \tag{13}$$

$$\overline{x}_R \le y_r \le \overline{x}_{R+1}. \tag{14}$$

There is no closed-form solution for L and R, and hence, for y_l and y_r. Karnik–Mendel (KM) algorithms [2], [8] are used to compute them iteratively. Though KM algorithms have been proven to converge monotonically and superexponentially fast [5], several (usually two to six) iterations are still needed before convergence occurs. Mendel and Liu also proposed an open problem in [5], i.e., how "*to find an optimal way to initialize the KM algorithm*," optimal in the sense that the superexponential convergence factor δ defined in (42) is minimized.

Because the computational burden and iterative nature of KM algorithms may hinder them from some real-time applications,[4] a reduction in their computational cost is desired, and this is the focus of our paper, which not only tackles the aforementioned open problem, but also proposes other techniques to further reduce the computational cost.

The rest of the paper is organized as follows. Section II briefly introduces the original KM algorithms. Section III proposes the enhanced KM (EKM) algorithms. Section IV presents simulation results to verify the effectiveness of the proposed algorithms. Section V comments on the EKM algorithms, and finally, Section VI draws conclusions.

II. Original KM Algorithms

The original KM algorithms are presented in this section.

A. KM Algorithm for Computing y_l [2], [8]

1) Sort $\underline{x}_i$ $(i = 1, 2, \ldots, N)$ in increasing order and call the sorted $\underline{x}_i$ by the same name, but now, $\underline{x}_1 \le \underline{x}_2 \le \cdots \le \underline{x}_N$. Match the weights w_i with their respective $\underline{x}_i$ and renumber them so that their index corresponds to the renumbered $\underline{x}_i$.
2) Initialize w_i by setting

$$w_i = \frac{\underline{w}_i + \overline{w}_i}{2}, \qquad i = 1, 2, \ldots, N \tag{15}$$

and then compute

$$y = \frac{\sum_{i=1}^{N} \underline{x}_i w_i}{\sum_{i=1}^{N} w_i}. \tag{16}$$

3) Find switch point k $(1 \le k \le N - 1)$ such that

$$\underline{x}_k \le y \le \underline{x}_{k+1}. \tag{17}$$

4) Set

$$w_i = \begin{cases} \overline{w}_i, & i \le k \\ \underline{w}_i, & i > k \end{cases} \tag{18}$$

and compute

$$y' = \frac{\sum_{i=1}^{N} \underline{x}_i w_i}{\sum_{i=1}^{N} w_i}. \tag{19}$$

5) Check if $y' = y$. If yes, stop, set $y_l = y$, and call k L. If no, go to step 6).
6) Set $y = y'$ and go to step 3).

[4]Some fast algorithms [15], [16] have been proposed to approximate the KM algorithms. Because the outputs of these algorithms are different from those of the KM algorithms and they are not as widely used as the KM algorithms, they are not considered in this paper.

B. KM Algorithm for Computing y_r [2], [8]

1) Sort $\overline{x}_i$ $(i = 1, 2, \ldots, N)$ in increasing order, and call the sorted $\overline{x}_i$ by the same name, but now, $\overline{x}_1 \leq \overline{x}_2 \leq \cdots \leq \overline{x}_N$. Match the weights w_i with their respective $\overline{x}_i$ and renumber them so that their index corresponds to the renumbered $\overline{x}_i$.
2) Initialize w_i by setting

$$w_i = \frac{\underline{w}_i + \overline{w}_i}{2}, \qquad i = 1, 2, \ldots, N \tag{20}$$

and then compute

$$y = \frac{\sum_{i=1}^{N} \overline{x}_i w_i}{\sum_{i=1}^{N} w_i}. \tag{21}$$

3) Find switch point k $(1 \leq k \leq N-1)$ such that

$$\overline{x}_k \leq y \leq \overline{x}_{k+1}. \tag{22}$$

4) Set

$$w_i = \begin{cases} \underline{w}_i, & i \leq k \\ \overline{w}_i, & i > k \end{cases} \tag{23}$$

and compute

$$y' = \frac{\sum_{i=1}^{N} \overline{x}_i w_i}{\sum_{i=1}^{N} w_i}. \tag{24}$$

5) Check if $y' = y$. If yes, stop, set $y_r = y$, and call k R. If no, go to step 6).
6) Set $y = y'$ and go to step 3).

III. EKM Algorithms

This section presents EKM algorithms to reduce the computational cost of the original ones. First, a better initialization is used to reduce the number of iterations. Then, the termination condition of the iterations is changed to remove one unnecessary iteration. Finally, a subtle computing technique is used to reduce the computational cost of each iteration.

Similar to the original KM algorithms, the EKM algorithms also consist of two parts: one for computing y_l and the other for computing y_r. Because the two parts are quite similar, we focus on the EKM algorithm for computing y_l in this section.

A. Optimal Initial Switch Point

When we use (15) to initialize the KM algorithm, y_l in the first iteration can be expressed as

$$y_l = \frac{\sum_{i=1}^{N} \underline{x}_i \left[(\overline{w}_i + \underline{w}_i)/2\right]}{\sum_{i=1}^{N} (\overline{w}_i + \underline{w}_i)/2} \tag{25}$$

which looks quite different from (11), and suggests that better choices for the initialization of the KM algorithm in line with (11) should be possible.

Observe that (11) shows that when $i \leq L$, $\overline{w}_i$ is used to compute y_l, and when $i > L$, $\underline{w}_i$ is used to compute y_l. This implies that a better initialization of y_l is to find a good guess of L, L_0. Because y_l is the smallest value of Y, we conjecture that very probably it is smaller than $\underline{x}_{[N/2]}$,[5] i.e., the center element of $\underline{x}_i$; consequently, L_0 should also be smaller than $[N/2]$. We performed extensive simulations by initializing $L_0 = \{[N/2], [N/2.1], \ldots, [N/2.6]\}$ and comparing the number of iterations for the algorithms to converge for uniformly and independently distributed $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$, and found that $L_0 = [N/2.4]$ gave the fewest number of iterations (more details on the simulations and comparison are given in Section IV). We performed similar simulations for y_r and found that the optimal initial switch point is $R_0 = [N/1.7]$.

B. Termination Test

Observe from step 5) in Section II-A that the test $y' = y$ is performed to determine whether the iterations should stop or continue. When the iterations stop, $y' = y$, and because y' is obtained during the present iteration and y was obtained from the previous iteration, $y' = y$ means the present iteration makes no contribution to minimizing y_l; consequently, it can be deleted without changing y_l.

Denote the switch points for y' and y as k' and k, respectively. Then

$$y' = \frac{\sum_{i=1}^{k'} \underline{x}_i \overline{w}_i + \sum_{i=k'+1}^{N} \underline{x}_i \underline{w}_i}{\sum_{i=1}^{k'} \overline{w}_i + \sum_{i=k'+1}^{N} \underline{w}_i} \tag{26}$$

$$y = \frac{\sum_{i=1}^{k} \underline{x}_i \overline{w}_i + \sum_{i=k+1}^{N} \underline{x}_i \underline{w}_i}{\sum_{i=1}^{k} \overline{w}_i + \sum_{i=k+1}^{N} \underline{w}_i}. \tag{27}$$

Obviously, $y' = y$ is equivalent to $k' = k$. So, by changing the termination condition from $y' = y$ to $k' = k$, we have the same y_l but save one iteration. How to do this is shown in Section III-D.

C. Further Computational Cost Reduction

In the original KM algorithm for computing y_l, in each iteration, we compute $\sum_{i=1}^{N} w_i$ and $\sum_{i=1}^{N} \underline{x}_i w_i$ in entirety and then compute y' in (19). This is a waste of computational power because results from the previous iteration are not utilized.

After the jth iteration, let switch point k, $\sum_{i=1}^{N} w_i$, and $\sum_{i=1}^{N} \underline{x}_i w_i$ be denoted as k_j, $(\sum_{i=1}^{N} w_i)_j$, and $(\sum_{i=1}^{N} \underline{x}_i w_i)_j$, respectively. Usually, k_j and k_{j+1} are quite close to each other. Consequently, the w_i in the $(j+1)$th iteration shares lots of common terms with the w_i from the jth iteration. This means $(\sum_{i=1}^{N} w_i)_j$ and $(\sum_{i=1}^{N} \underline{x}_i w_i)_j$ can be used to compute $(\sum_{i=1}^{N} w_i)_{j+1}$ and $(\sum_{i=1}^{N} \underline{x}_i w_i)_{j+1}$, i.e., only the differences between $(\sum_{i=1}^{N} w_i)_{j+1}$ and $(\sum_{i=1}^{N} w_i)_j$, as well as $(\sum_{i=1}^{N} \underline{x}_i w_i)_{j+1}$ and $(\sum_{i=1}^{N} \underline{x}_i w_i)_j$, need to be computed, after which these differences are added to $(\sum_{i=1}^{N} w_i)_j$ and $(\sum_{i=1}^{N} \underline{x}_i w_i)_j$ [as shown in (32) and (33)]. A similar technique was used in [1]. The effectiveness of this technique is verified in Section IV-A3.

[5] $[N/2]$ denotes the nearest integer number to which $N/2$ can be rounded. This conversion is needed because L_0 (R_0) must be an integer number, whereas $N/2$ is not necessarily an integer.

D. EKM Algorithms

As a summary, the complete EKM algorithms are presented.

The *EKM algorithm for computing* y_l is as follows.

1) Sort $\underline{x}_i$ $(i = 1, 2, \ldots, N)$ in increasing order and call the sorted $\underline{x}_i$ by the same name, but now $\underline{x}_1 \le \underline{x}_2 \le \cdots \le \underline{x}_N$. Match the weights w_i with their respective $\underline{x}_i$ and renumber them so that their index corresponds to the renumbered $\underline{x}_i$.
2) Set $k = [N/2.4]$ (the nearest integer to $N/2.4$), and compute

$$a = \sum_{i=1}^{k} \underline{x}_i \overline{w}_i + \sum_{i=k+1}^{N} \underline{x}_i \underline{w}_i \tag{28}$$

$$b = \sum_{i=1}^{k} \overline{w}_i + \sum_{i=k+1}^{N} \underline{w}_i \tag{29}$$

and

$$y = \frac{a}{b}. \tag{30}$$

3) Find $k' \in [1, N-1]$ such that

$$\underline{x}_{k'} \le y \le \underline{x}_{k'+1}. \tag{31}$$

4) Check if $k' = k$. If yes, stop, set $y_l = y$, and call k L. If no, continue.
5) Compute $s = \text{sign}(k' - k)$, and[6]

$$a' = a + s \sum_{i=\min(k,k')+1}^{\max(k,k')} \underline{x}_i (\overline{w}_i - \underline{w}_i) \tag{32}$$

$$b' = b + s \sum_{i=\min(k,k')+1}^{\max(k,k')} (\overline{w}_i - \underline{w}_i) \tag{33}$$

$$y' = \frac{a'}{b'}. \tag{34}$$

6) Set $y = y'$, $a = a'$, $b = b'$, and $k = k'$. Go to step 3).

The *EKM algorithm for computing* y_r is as follows.

1) Sort $\overline{x}_i$ $(i = 1, 2, \ldots, N)$ in increasing order and call the sorted $\overline{x}_i$ by the same name, but now, $\overline{x}_1 \le \overline{x}_2 \le \cdots \le \overline{x}_N$. Match the weights w_i with their respective $\overline{x}_i$, and renumber them so that their index corresponds to the renumbered $\overline{x}_i$.
2) Set $k = [N/1.7]$ (the nearest integer to $N/1.7$), and compute

$$a = \sum_{i=1}^{k} \overline{x}_i \underline{w}_i + \sum_{i=k+1}^{N} \overline{x}_i \overline{w}_i \tag{35}$$

$$b = \sum_{i=1}^{k} \underline{w}_i + \sum_{i=k+1}^{N} \overline{w}_i \tag{36}$$

and

$$y = \frac{a}{b}. \tag{37}$$

3) Find $k' \in [1, N-1]$ such that

$$\overline{x}_{k'} \le y \le \overline{x}_{k'+1}. \tag{38}$$

4) Check if $k' = k$. If yes, stop, set $y_r = y$, and call k R. If no, continue.
5) Compute $s = \text{sign}(k' - k)$, and

$$a' = a - s \sum_{i=\min(k,k')+1}^{\max(k,k')} \overline{x}_i (\overline{w}_i - \underline{w}_i) \tag{39}$$

$$b' = b - s \sum_{i=\min(k,k')+1}^{\max(k,k')} (\overline{w}_i - \underline{w}_i) \tag{40}$$

$$y' = \frac{a'}{b'}. \tag{41}$$

6) Set $y = y'$, $a = a'$, $b = b'$, and $k = k'$. Go to step 3).

IV. Comparative Studies

Extensive simulations have been conducted to verify the performance of the EKM algorithms. The platform was a Dell Precision 690 Workstation running Windows XP x64 Edition and Matlab 7.3.0 with two Intel Xeon 2.66 GHz processors and 2 GB RAM. Because the results for computing y_r are quite similar to those for computing y_l, only the comparative studies for computing y_l are presented in this section.

A. Uniformly and Independently Distributed $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$

In the simulations, we increased N by 1 from 3 to 20 (i.e., $N = 3, 4, \ldots, 20$) and then increased it by 5 from 20 to 100 (i.e., $N = 25, 30, \ldots, 100$). For each N, 10 000 Monte Carlo simulations were used to compute y_l, i.e., for each N, 10 000 $\underline{x}_i$'s were generated using Matlab function rand(10000,1), and 10 000 pairs of $\{\underline{w}_i, \overline{w}_i\}$ were generated by using Matlab function rand(10000,2). Observe that all $\underline{x}_i$, $\underline{w}_i$, and $\overline{w}_i$ were constrained in $[0, 1]$, and $\underline{x}_i$'s were independent of $\underline{w}_i$ and $\overline{w}_i$. To make sure $\underline{w}_i \le \overline{w}_i$, we checked each pair of $\{\underline{w}_i, \overline{w}_i\}$ and assigned the smaller value to $\underline{w}_i$ and the larger one to $\overline{w}_i$.

The performance measures used in the comparative studies and related observations are presented next.

1) Verification of the Effectiveness of the Optimal Initial Switch Point: The performance measure used to verify the effectiveness of the optimal initial switch point (Section III-A) is the *mean and standard deviation of superexponential convergence factor* δ (see Fig. 2), which is defined in [5, eq. (30)]

[6]When $k' > k$, it is true that

$$a' = a + s \sum_{i=k+1}^{k'} \underline{x}_i (\overline{w}_i - \underline{w}_i), \qquad b' = b + s \sum_{i=k+1}^{k'} (\overline{w}_i - \underline{w}_i)$$

and when $k > k'$, it is true that

$$a' = a + s \sum_{i=k'+1}^{k} \underline{x}_i (\overline{w}_i - \underline{w}_i), \qquad b' = b + s \sum_{i=k'+1}^{k} (\overline{w}_i - \underline{w}_i).$$

Equations (32) and (33) express these two cases in a more concise form.

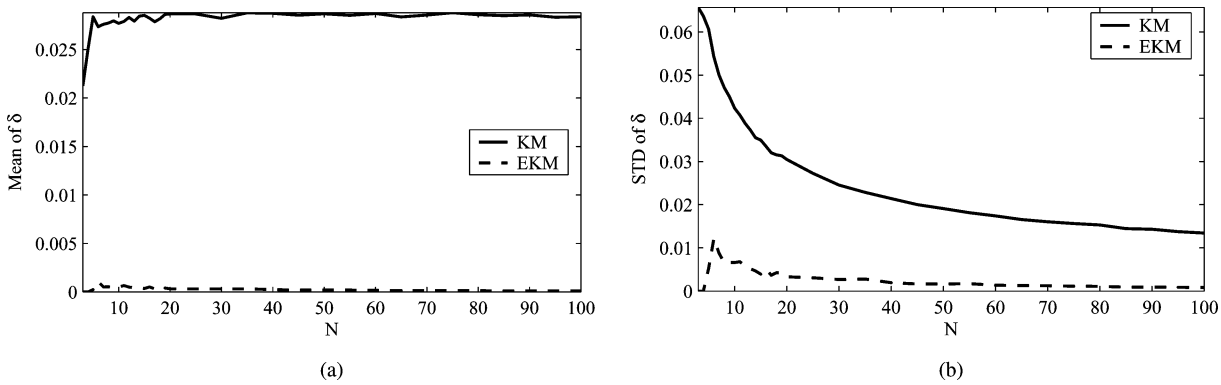


Fig. 2. (a) Mean. (b) Standard deviation of δ when computing y_l.

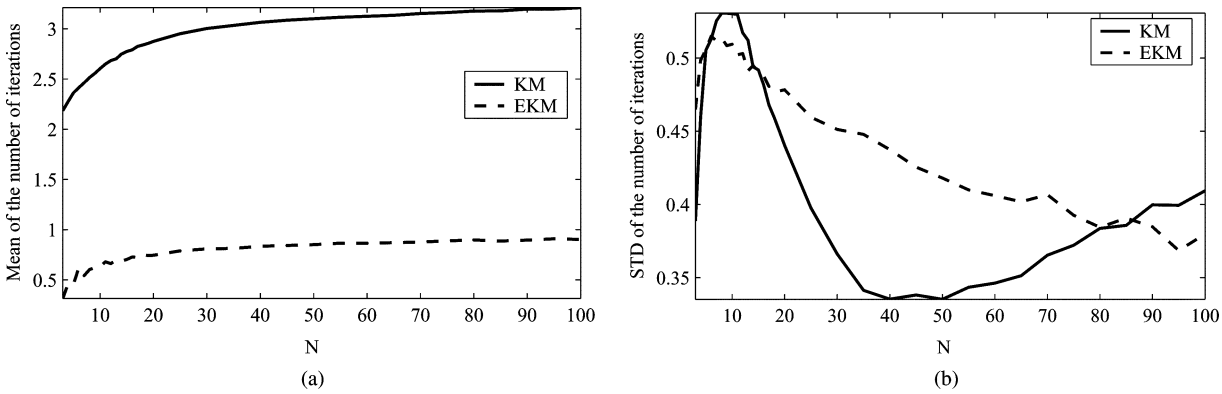


Fig. 3. (a) Mean. (b) Standard deviation of the number of iterations when computing y_l.

as an indicator of the superexponential convergence speed for computing y_l. The smaller the δ, the faster is the convergence. Because δ in [5] is defined for the continuous version of KM algorithms, and in this study, we used the discrete version of KM algorithms, we used the following discrete version of δ:

$$\delta \equiv \frac{y_{l1} - y_l}{y_{l0} - y_l} \tag{42}$$

where y_{l0} is the initial value of y_l [i.e., y computed by (16) for the original KM algorithm, or y computed by (30) for the EKM algorithm], and y_{l1} is y' computed from the first iteration of each algorithm.

Observe from Fig. 2 that both the mean and the standard deviation of δ for the EKM algorithm are smaller than those of the original KM algorithm. Most impressively, the mean of δ calculated from the EKM algorithm is about $1/100$ of that from the original KM algorithm, which suggests that the EKM algorithm converges much faster than the original one.

2) Verification of the Effectiveness of the Optimal Initial Switch Point and the Modified Termination Test: There are four performance measures used to verify that the EKM algorithm can reduce the number of iterations, i.e., the effectiveness of the techniques proposed in Sections III-A and B, which are as follows.

a) *Mean and standard deviation (STD) of the number of iterations obtained from different algorithms* (see Fig. 3): For both the original and EKM algorithms, the number of iterations is defined as the times the loop consisting of steps 3)–6) in Sections II-A and D are executed. These definitions are consistent with those used in [5]. For each N, the 10 000 numbers of iterations from the 10 000 Monte Carlo simulations were recorded for both the original and EKM algorithms, and their mean and standard deviation were computed accordingly, as shown in Fig. 3.

From Fig. 3(a), observe that the average number of iterations for the EKM algorithm is smaller than that for the original KM algorithm. More interestingly, the average number of iterations for the EKM algorithm is less than one. This is because uniformly and independently distributed $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$ were used in the simulation, and hence, $L_0 = [N/2.4]$ has a good chance to be the final switch point, especially when N is small, as confirmed by Fig. 4 and explained in Observation 2 later. When the distributions of $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$ are not uniform and independent, the average number of iterations for the EKM algorithm increases, as shown in Section IV-B. Also observe that as N increases, the average number of iterations in the original KM algorithm also increases; however, the increase in the average number of iterations becomes much slower as N gets larger. This coincides with the conclusion that the KM algorithms converge monotonically and superexponentially fast [5]. Interestingly, Fig. 3(b) shows that the standard deviation of the number of iterations

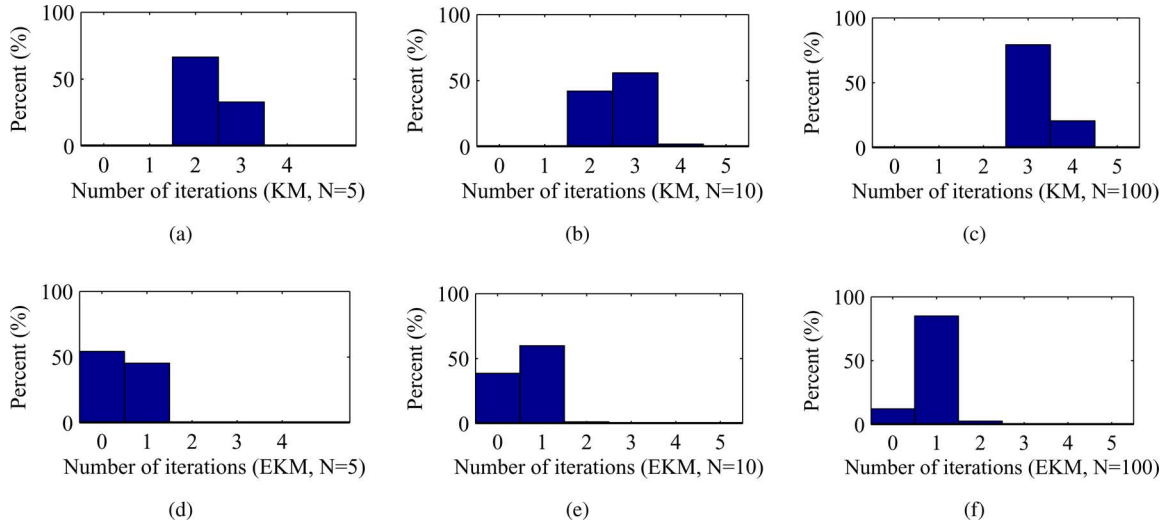


Fig. 4. Histograms of the number of iterations for different KM algorithms. (a)–(c) Original KM algorithm, $N = 5, 10, 100$. (d)–(f) EKM algorithm, $N = 5, 10, 100$.

for the original KM algorithm is not monotonic, and it achieves a minimum around $N = 45$.

b) *Histograms of the number of iterations obtained from different algorithms:* For both the original and EKM algorithms, histograms of the number of iterations for L are shown in Fig. 4, where $\{5, 10\}$ represent the typical N used in type-reduction and in computing the IWA, FWA, and LWA, $\{10, 100\}$ represent the typical N used in the prey model, and 100 represents the typical N used in computing the centroid, variance, and skewness of IT2 FSs. Observe that when $N \leq 100$, the original KM algorithm converges in two to four iterations whereas the EKM algorithm converges in zero to two iterations. Also observe that when $N = 5$, there is more than 50% probability that the EKM algorithm converges in zero iterations, i.e., the initial switch point L_0 equals the final switch point L. This result seems to be surprising, but it can be easily explained. When $N = 5$, there are only four possible switch points $L = \{1, 2, 3, 4\}$. The initial switch point $L_0 = [N/2.4] = 2$ has a good chance to be the final switch point, i.e., at least $1/4$ in probability. As N increases, the number of possible switch points also increases, and hence, the probability that the EKM converges in zero iterations decreases, as confirmed by Fig. 4(e) and (f).

c) *The average reduced number of iterations* (Fig. 5), which is computed by subtracting the average number of iterations of the EKM algorithm from that of the original KM algorithm. Observe from Fig. 5 that the EKM algorithm eliminates about two iterations and that the reduced number of iterations increases on average as N gets larger.

d) *Probability that the algorithm converges in no more than one iteration* (see Fig. 6), computed by recording the number of runs that converge in no more than one iteration and then dividing it by the total number of runs (10 000). As shown in Fig. 6, the EKM algorithm converges in at most one iteration with a probability of 0.97, and that probability is almost a constant for all N. On the other hand, the original KM algorithm almost surely needs more than one iteration to converge. These results are also verified by Fig. 4(d)–(f).

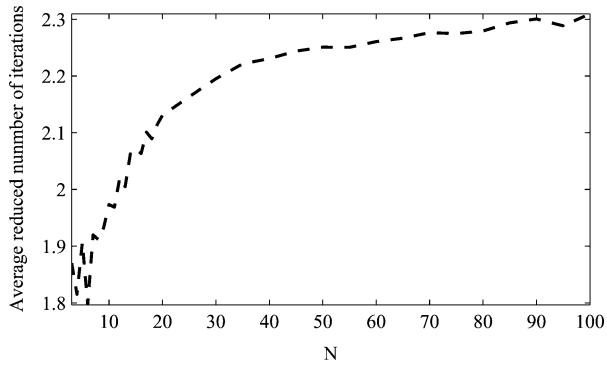


Fig. 5. Average reduced number of iterations of the EKM algorithm over the original KM algorithm.

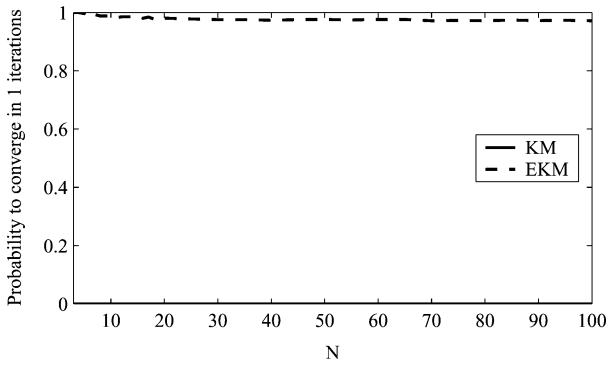


Fig. 6. Probability of each algorithm converging in no more than one iteration. Note that the probability corresponding to the original KM algorithm is 0.

3) Verification of the Effectiveness of (32)–(34): The performance measure used to verify the effectiveness of (32)–(34) is the *computation time saving achieved by replacing (18)–(19) with (32)–(34)* (see Fig. 7), where the computation times of

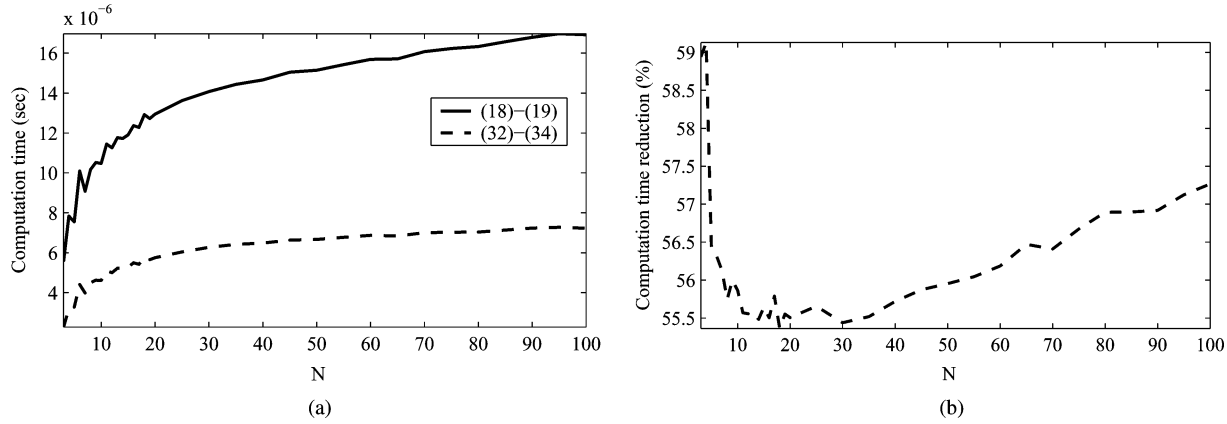


Fig. 7. (a) Computation time of (18) and (19) and (32)–(34). (b) Computation time reduction of (32)–(34) over (18) and (19).

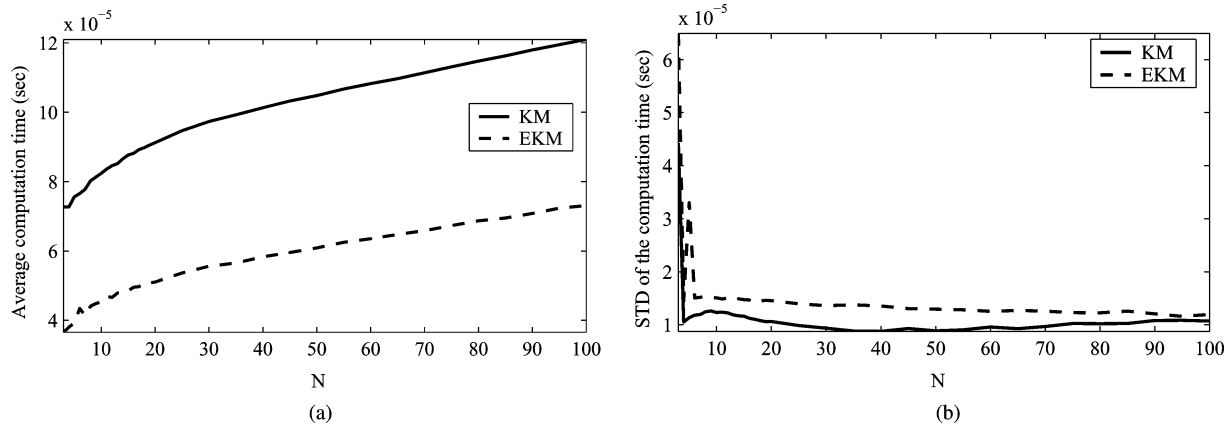


Fig. 8. (a) Mean. (b) Standard deviation of the computation time for different KM algorithms.

(18)–(19) and (32)–(34) were both recorded by using Matlab functions *tic* and *toc*. Observe from Fig. 7 that the computation of (32)–(34) is more than two times faster than that of (18) and (19). This justifies the use of (32)–(34) in the EKM algorithm.

4) Overall Performance of the EKM Algorithm: The following measures were used to study the overall performance of the EKM algorithm.

a) *The mean and standard deviation of the computation time for different algorithms* (see Fig. 8): The computation time was obtained by using Matlab command *tic* at the beginning of the algorithm and *toc* at the end. For each N, the 10 000 computation times from the 10 000 Monte Carlo simulations were recorded for both the original and EKM algorithms, and their mean and standard deviation were computed accordingly, as shown in Fig. 8.

Observe from Fig. 8(a) that the average computation time for both algorithms increases as N increases; however, the EKM algorithm is much faster than the original KM algorithm. Also observe that the standard deviation of the computation time for the EKM algorithm is slightly larger than that for the original KM algorithm.

b) *The percentage of computation time reduction over the original KM algorithm* (see Fig. 9), which equals $[(t_o - t_e)/t_o] \times 100\%$, where t_o and t_e are the average computation times of the original and EKM algorithm, respectively. Observe from Fig. 9 that the percentage of computation time reduction of the EKM algorithm over the original KM algorithm decreases as N increases. For $N \in [3, 100]$, there is more than a 39% computation time reduction.

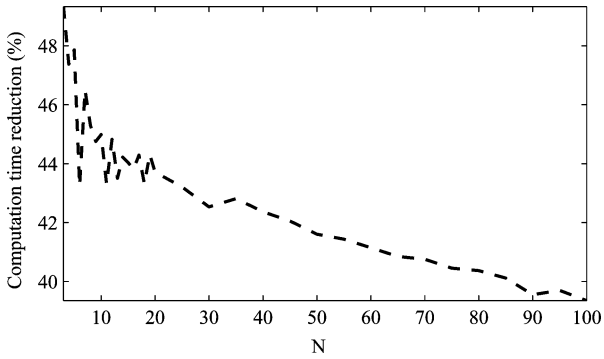
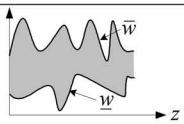
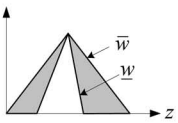
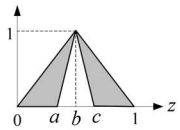
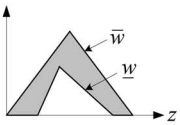
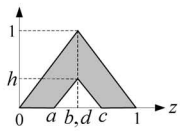
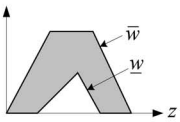
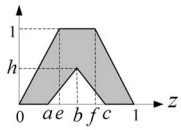
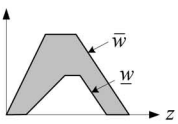
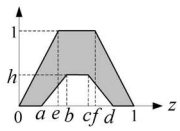
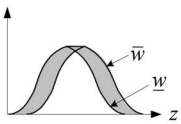
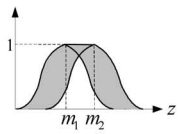
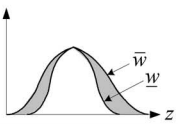
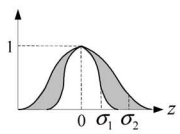


Fig. 9. Percentage of computation time reduction of the EKM algorithm over the original KM algorithm.

Finally, we wish to emphasize that the earlier results were obtained only for uniformly and independently distributed $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$, and the performance of the EKM algorithms may be

TABLE I
OPTIMAL INITIAL SWITCH POINTS FOR CENTROID COMPUTATIONS OF DIFFERENT IT2 FSS ($N = 100$)

Shape Description	Example	L_0/N	R_0/N	$\frac{L_0/N}{+R_0/N}$	*"Expected"* FOU	L/N	R/N	$\frac{L/N}{+R/N}$
Uniformly distributed $\underline{w}$ and $\overline{w}$ [a]		0.4167	0.5882	1.0049		0.4142	0.5858	1
Triangular UMF and triangular LMF with the same height [b]		0.4195	0.5814	1.0009		0.4161	0.5839	1
Triangular UMF and triangular LMF with different heights [c]		0.3401	0.6593	0.9994		0.3519	0.6481	1
Trapezoidal UMF and triangular LMF with different heights [d]		0.3263	0.6726	0.9989		0.3325	0.6675	1
Trapezoidal UMF and trapezoidal LMF with different heights [e]		0.3887	0.6134	1.0021		0.3616	0.6384	1
Gaussian MF with fixed STD and uncertain means [f]		0.4094	0.5906	1		0.4257	0.5743	1
Gaussian MF with fixed mean and uncertain STDs [g]		0.4303	0.5697	1		0.4356	0.5644	1

[a] $h_1 = 1/3$, $h_2 = 2/3$.
[b] $a = 1/4$, $b = 1/2$, $c = 3/4$.
[c] $a = 1/4$, $b = d = 1/2$, $c = 3/4$, $h = 0.3848$.
[d] $a = 1/4$, $b = 1/2$, $c = 3/4$, $e = 1/3$, $f = 2/3$, $h = 0.4349$.
[e] $a = 1/5$, $b = 2/5$, $c = 3/5$, $d = 4/5$, $e = 1/3$, $f = 2/3$, $h = 0.4012$.
[f] $m_2 - m_1 = 2.55$, $\sigma = 2.55$.
[g] $\sigma_1 = 1.7$, $\sigma_2 = 3.4$.

different for other distributions. This is discussed further in the next section.

B. Centroid Computation of IT2 FSs

Note that the optimal initial switch points $L_0 = [N/2.4]$ and $R_0 = [N/1.7]$ were obtained experimentally, where $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$ were assumed to be uniformly and independently distributed in $[0, 1]$ (corresponding to the case shown in row 2 of Table I). This may be a reasonable assumption for type-reduction, prey modeling, and computing the IWA, FWA, and LWA because these problems have many independent parameters, and hence, lots of randomness; however, for centroid, variance, and skewness computations of IT2 FSs, usually, $\underline{w}_i$ and $\overline{w}_i$, which are obtained from the UMF and the LMF of a footprint of uncertainty [(FOU); see Fig. 1], are not uniformly distributed, and they also depend on $\underline{x}_i$. In this case, the optimal switch points obtained in Section III may no longer be optimal. The centroid computation is used next as an example to demonstrate this; the results for variance and skewness computations are similar.

In all, 10 000 Monte Carlo simulations were used to find the optimal initial switch points to compute the centroids of different triangular or trapezoidal IT2 FSs shown in rows 3–6 of Table I. As an example, we explain the simulation procedure by considering the IT2 FS shown in Fig. 10. Without loss of generality, the universe of discourse was fixed to be $[0, 1]$. Four

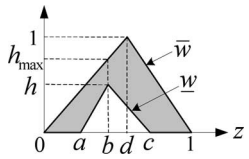


Fig. 10. Triangular IT2 FS, whose UMF and LMF have different heights.

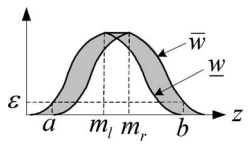


Fig. 11. Gaussian IT2 FS.

points (a, b, c, and h) are needed to determine the LMF, and one (d) is needed to determine the UMF. The following procedure was used to generate them in each of the 10 000 Monte Carlo simulations.

1) Three uniformly distributed random numbers were generated and sorted. The smallest one was assigned to a, the largest one was assigned to c, and the middle one was assigned to b.
2) One uniformly distributed random number was generated and assigned to d.
3) Another uniformly distributed random number t was generated for h. Observe from Fig. 10 that once b and d are determined, h cannot be larger than $h_{\max}$ because otherwise, the LMF would intersect the UMF. Consequently, $h = \min(t, h_{\max})$ was used.

The resulting IT2 FS was then discretized ($N = 100$) to compute y_l, i.e., the left bound of its centroid. The corresponding final switch point was recorded. When all 10 000 Monte Carlo simulations were completed, the optimal initial switch point L_0 was computed as the average of all 10 000 final switch points.

Monte Carlo simulations were also used to find the optimal initial switch points for computing the centroids of the Gaussian IT2 FSs shown in rows 7 and 8 of Table I. Unlike a triangular or a trapezoidal IT2 FS, which has a finite domain, a Gaussian IT2 FS has an infinite domain; hence, the domain over which x should be discretized had to be specified. In this paper, the domain of a Gaussian IT2 FS was chosen as the interval $[a, b]$ for which its upper memberships are equal to or larger than a small positive constant ε (see Fig. 11), where $\varepsilon = 0.01$ was used in the simulations.

For the row 7 Gaussian IT2 FSs with fixed standard deviation (STD) σ and uncertain means $[m_1, m_2]$, we used $\sigma = \{0.1, 0.2, \ldots, 5\}$, and for each σ, we chose $m_2 - m_1 = \{0.1, 0.2, \ldots, 5\}$. The optimal initial switch point L_0 was computed as the average of the final switch points from all these $50 \times 50 = 2500$ simulations. For the row 8 Gaussian IT2 FSs with fixed mean m and uncertain STDs $[\sigma_1, \sigma_2]$, we used $\sigma_1 = \{0.1, 0.2, \ldots, 5\}$, and for each σ_1, we chose $\sigma_2 = \{\sigma_1 + 0.1, \sigma_1 + 0.2, \ldots, 5\}$. Note that the value of m is not important here since we are only interested in the final switch points, which are situated symmetrically about m [9]. The optimal initial switch point L_0 was computed as the average of the final switch points for these $49 + 48 + \cdots + 1 = 1225$ simulations.

In Table I, the "expected" FOU for each category of shapes is also shown, e.g., for the uniformly distributed $\underline{w}$ and $\overline{w}$, the values[7] $E[\underline{w}_i] = 1/3$ and $E[\overline{w}_i] = 2/3$ were computed $\forall i$, and hence, the "expected" FOU for this category of shapes is a fuzzy granule [7] shown in column 6 and row 2, where $h_1 = E[\underline{w}_i] = 1/3$, $h_2 = E[\overline{w}_i] = 2/3$. The switch points for this "expected" FOU are presented in the next two columns. They were computed by using the closed-form formulas presented in [7]. For "expected" FOUs of the triangular and trapezoidal IT2 FSs shown in rows 3–6 of Table I, points a–f can be computed from probability theory; however, it was very difficult to compute the value of h mathematically (see the procedure to generate h in the second paragraph of Section IV-B). Hence, the mean value of the 10 000 h generated from the 10 000 Monte Carlo simulations was used. The variables $m_2 - m_1$, σ, σ_1, and σ_2 for the "expected" FOUs of the Gaussian IT2 FSs were also computed from the Monte Carlo simulations. Generally, for the values shown in the footnotes of Table I, the fractions were computed from probability theory, and the decimal fractions were computed from Monte Carlo simulations.

We observe the following from Table I.

1) Regardless of the shape of the FOU, the optimal initial switch point for y_l is always smaller than $N/2$; for y_r, it is always larger than $N/2$, and $L_0 + R_0 \approx N$.[8]
2) The optimal initial switch points are close to the switch points of the corresponding "expected" FOU.
3) For some shapes, L_0/N are quite close to $1/2.4 \approx 0.4167$, and R_0/N are quite close to $1/1.7 \approx 0.5882$, whereas for some other shapes, there are noticeable differences.
4) Though the probability of generating a symmetrical IT2 FS in the simulations is zero, the "expected" FOUs are all symmetrical.

Table I suggests that, in practice, the switch points of the "expected" FOU may be used as estimates of the optimal initial switch points, given the distribution of $\underline{w}_i$ and $\overline{w}_i$.

Next, we will show that for IT2 FSs of a particular shape, there is always a performance improvement over the original KM algorithms whether we initialize $L_0 = [N/2.4]$ and $R_0 = [N/1.7]$ or use the values shown in Table I.

Consider IT2 FSs with trapezoidal UMFs and triangular LMFs (shown in the fifth row of Table I), for which the optimal

[7] $E[\underline{w}_i]$ is computed as follows. Let (t_1, t_2) be one of the 10 000 pairs generated by Matlab function *rand(10000,2)* (see the first paragraph in Section IV-A) and $p(t_1)$ and $p(t_2)$ be the probability density function of t_1 and t_2, respectively. Note that t_1 and t_2 are independent, and each of them is uniformly distributed in $[0, 1]$. Let $\underline{w}_i = \min(t_1, t_2)$, $\overline{w}_i = \max(t_1, t_2)$, and δ be an infinitesimal positive number. Then, $p(\underline{w}_i = w) = p(t_1 = w)[p(t_2 = w) + p(t_2 = w + \delta) + \cdots + p(t_2 = 1)] + p(t_2 = w)[p(t_1 = w) + p(t_1 = w + \delta) + \cdots + p(t_1 = 1)] = 2p(t_1 = w)[p(t_2 = w) + p(t_2 = w + \delta) + \cdots + p(t_2 = 1)] = 2p(w_1 = w)\int_w^1 p(w_2 = z)dz = 2 \times 1 \times (1 - w) = 2(1 - w)$, and hence, $E[\underline{w}_i] = \int_0^1 2w(1 - w)dw = 1/3$. $E[\overline{w}_i]$ is computed in a similar way.

[8] For any symmetrical IT2 FS, such as the Gaussian IT2 FS shown in the last two rows of Table I, it is always true that $L + R = N$. This is a theoretical result proved in [9], which is why $L/N + R/N = 1$ for the Gaussian MFs and all of the "expected" FOUs.

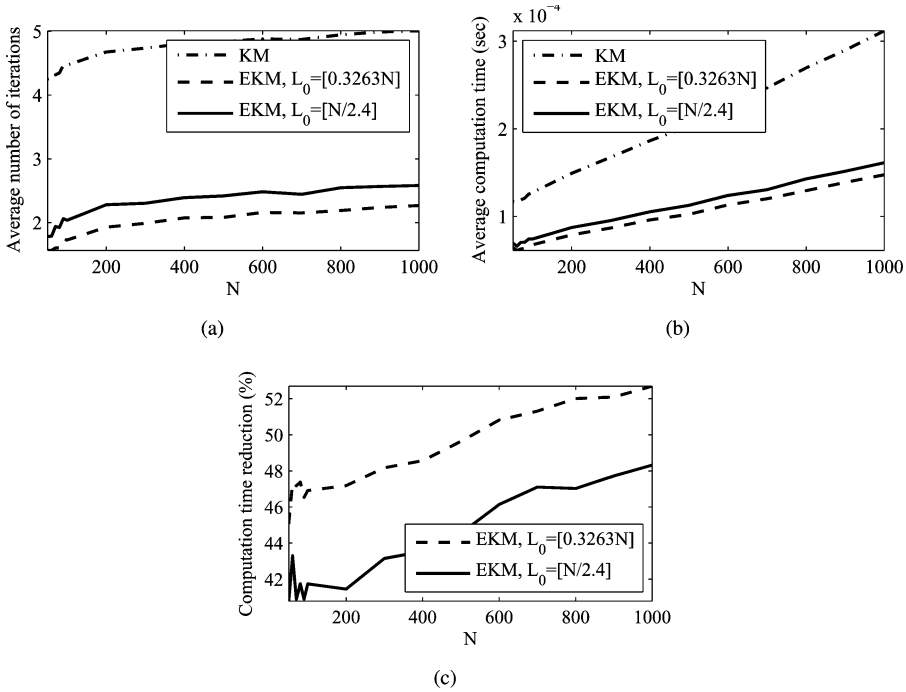


Fig. 12. Comparison of EKM algorithms using $L_0 = \{[0.3263N], [N/2.4]\}$ against the original KM algorithm for computing the centroid of the IT2 FS shown in the fifth row of Table I. (a) Average number of iterations. (b) Average computation time. (c) Computation time-reduction over the original KM algorithm.

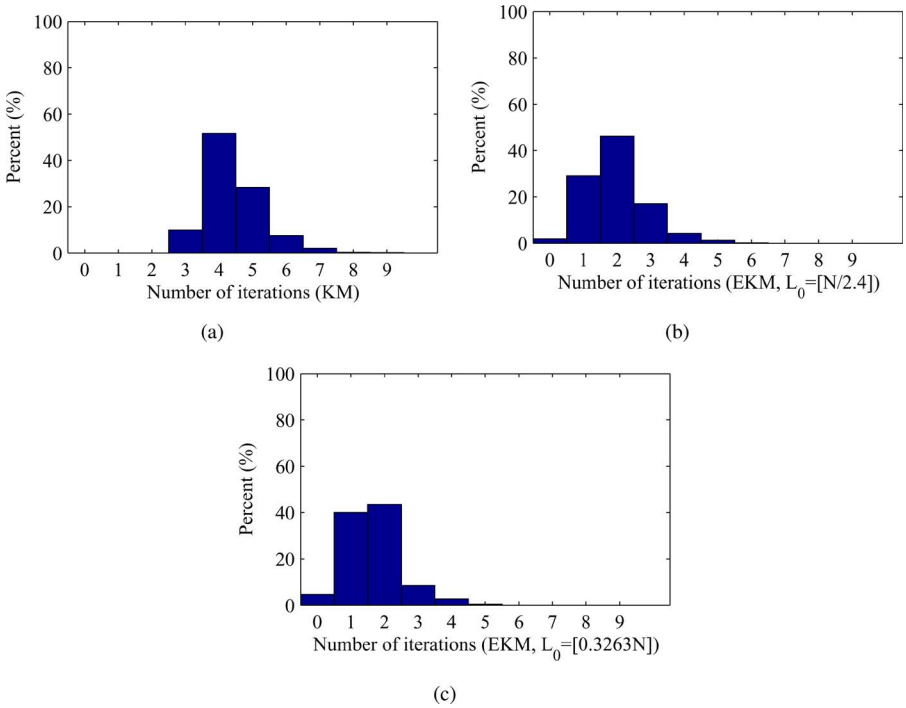


Fig. 13. Histograms of the number of iterations for different KM algorithms ($N = 100$).

L_0 is $[0.3263N]$. This shape is of interest because 0.3263 is the number furthest away from 1/2.4 in Table I; hence, the performance deterioration should be the largest among all the shapes if $L_0 = [N/2.4]$ is used in the EKM. A comparison of EKM algorithms with $L_0 = \{[0.3263N], [N/2.4]\}$ and the original KM algorithm for computing y_l is shown in Fig. 12. The corresponding histograms are shown in Fig. 13. Observe from Fig. 12 that both of the EKM algorithms significantly outperform the original KM algorithm, and the EKM with $L_o = [N/2.4]$ has more than a 41% computation time reduction over the original KM algorithm. Observe also from Fig. 13 that the original KM algorithm needs about three to seven iterations to converge, the

EKM algorithm with $L = [N/2.4]$ needs about zero to five iterations to converge, and the EKM algorithm with $L = [0.3263N]$ needs about zero to four iterations to converge. Although the EKM algorithm with $L_0 = [N/2.4]$ performs a little worse than the one with $L_0 = [0.3263N]$, the deterioration is small.

Additionally, in practice, two cases frequently occur when computing the centroids of IT2 FSs.

1) The shapes of the FOUs are known ahead of time, e.g., in the center-of-sets type-reduction [8], it is necessary to compute the centroids of the consequent IT2 FSs for all rules. When the design of an FLS is completed, those consequent IT2 FSs are fixed; consequently, their centroids can be computed off-line and then recorded for on-line usage. For the off-line computations, speed is not very important, and hence, the small computation time increase when using $L_0 = [N/2.4]$ instead of the other optimal L_0 can be ignored.
2) The shapes of the FOUs are not known ahead of time, e.g., in the vector similarity measure [14], to map the output IT2 FS of the linguistic weighted average [12] $\tilde{Y}_{\mathrm{LWA}}$ to a word in a codebook,[9] it is necessary to compute the centroid of $\tilde{Y}_{\mathrm{LWA}}$; however, the shape of $\tilde{Y}_{\mathrm{LWA}}$ is not known until it has actually been computed. In this case, it is not known what the optimal L_0 should be, even though Table I provides results for many FOUs. A safe choice would be $L_0 = [N/2.4]$.

The earlier analysis suggests the use of $L_0 = [N/2.4]$ and $R_0 = [N/1.7]$, regardless of the problem.

V. Discussion

One of the most time-consuming steps in the EKM algorithms is the first one, i.e., sorting $\underline{x}_i$ ($\overline{x}_i$) in ascending order; however, frequently, this step can be skipped, e.g., note the following.

1) In computing the centroid of an IT2 FS, $\underline{x}_i = \overline{x}_i = z_i$, where z_i's are the samples of the primary variable (see Fig. 1), and they are already in ascending order.
2) In computing the skewness of an IT2 FS, $\underline{x}_i = \overline{x}_i = [z_i - c(\tilde{A})]^3$, because z_i's are already in ascending order, $[z_i - c(\tilde{A})]^3$ are also in ascending order.
3) For center-of-sets type-reduction, once the design of an IT2 FLS is completed, the centroids of the consequent IT2 FSs can be computed and sorted off-line, and hence, they can be used directly in on-line computations without further sorting.

Monte Carlo simulations were also used to study the further computation time reduction for the EKM algorithms when sorting is not needed. The same $\underline{w}_i$, $\overline{w}_i$, and $\underline{x}_i$ that were used in Section IV-A were used in the present simulations. Denote the time to finish step 1) of the EKM algorithms as t_1 and the time to finish steps 2)–6) as t_2. The ratio $t_1/(t_1 + t_2)$ was used as an indicator of how much computation time can be saved if no sorting is needed. The results are shown in Fig. 14. Observe that as N increases from 3 to 100, the percentage of computation time reduction first decreases and then increases. The minimum is around $N = 11$. Fig. 14 also shows that there is at least 23% computation time reduction. This computation time saving makes the EKM algorithms more suitable for real-time applications.

[9]A codebook consists of a list of words and their associated FOUs.

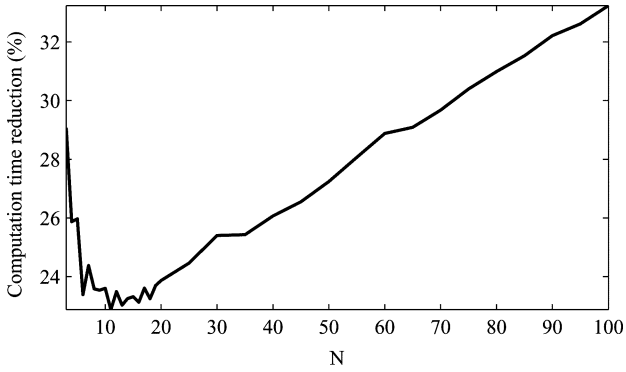


Fig. 14. Percentage of computation time reduction for the EKM algorithm if sorting of $\underline{x}_i$ ($\overline{x}_i$) is not needed.

VI. Conclusion

KM algorithms are iterative procedures widely used in centroid, variance, and skewness computations of IT2 FSs, type-reduction of IT2 FLSs, and for computing the IWA, FWA, and LWA. They converge monotonically and superexponentially fast; however, several (usually two to six) iterations are still needed before convergence occurs. In this paper, EKM algorithms have been proposed to reduce the computational cost. Extensive simulations show that, on average, the EKM algorithms can save about two iterations, which corresponds to a more than 39% reduction in computation time. An additional (at least) 23% computational cost can be saved if no sorting of $\underline{x}_i$ ($\overline{x}_i$) is needed.

Appendix A
Derivation of the KM Algorithms

Regardless of whether y_l in (6) or y_r in (7) are computed, it is necessary to minimize or maximize the function (the material in this Appendix is taken from [8])

$$\frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i} \equiv f(w_1, \ldots, w_N). \tag{A.1}$$

Differentiating $f(w_1, \ldots, w_N)$ with respect to w_k, observe that

$$\frac{\partial f(w_1, \ldots, w_N)}{\partial w_k} = \frac{x_k - f(w_1, \ldots, w_N)}{\sum_{i=1}^{N} w_i}. \tag{A.2}$$

As noted by Karnik and Mendel [2], equating $\partial f/\partial w_k$ to zero does not give us any information about the value of w_k that optimizes $f(w_1, \ldots, w_N)$, i.e.,

$$f(w_1, \ldots, w_N) = x_k \Rightarrow \frac{\sum_{i=1}^{N} x_i w_i}{\sum_{i=1}^{N} w_i} = x_k$$
$$\Rightarrow \frac{\sum_{i \neq k}^{N} x_i w_i}{\sum_{i \neq k}^{N} w_i} = x_k. \tag{A.3}$$

Observe that w_k no longer appears in the final expression in (A.3), and therefore, the direct calculus approach does not work. Returning to (A.2), because $\sum_{i=1}^{N} w_i > 0$, it is true that

$$\frac{\partial f(w_1,\ldots,w_N)}{\partial w_k} \begin{cases} \geq 0, & \text{if } x_k \geq f(w_1,\ldots,w_N) \\ < 0, & \text{if } x_k < f(w_1,\ldots,w_N). \end{cases} \tag{A.4}$$

This equation gives us the direction in which w_k should be changed in order to increase or decrease $f(w_1,\ldots,w_N)$, i.e.,

$$\text{If } x_k > f(w_1,\ldots,w_N)$$
$$\begin{cases} f(w_1,\ldots,w_k) \text{ increases as } w_k \text{ increases} \\ f(w_1,\ldots,w_k) \text{ decreases as } w_k \text{ decreases.} \end{cases} \tag{A.5}$$
$$\text{If } x_k < f(w_1,\ldots,w_N)$$
$$\begin{cases} f(w_1,\ldots,w_k) \text{ increases as } w_k \text{ decreases} \\ f(w_1,\ldots,w_k) \text{ decreases as } w_k \text{ increases.} \end{cases} \tag{A.6}$$

Because $w_k \in [\underline{w}_k, \overline{w}_k]$, the maximum value w_k can attain is $\overline{w}_k$ and the minimum value it can attain is $\underline{w}_k$. Therefore, (A.6) implies that $f(w_1,\ldots,w_N)$ attains its minimum value y_l if: 1) for those values of k for which[10] $x_k < f(w_1,\ldots,w_N)$, set $w_k = \overline{w}_k$, and 2) for those values of k for which $x_k > f(w_1,\ldots,w_N)$, set $w_k = \underline{w}_k$. Similarly, deduce from (A.6) that $f(w_1,\ldots,w_N)$ attains its maximum value y_r if 1) for those values of k for which $x_k < f(w_1,\ldots,w_N)$, set $w_k = \underline{w}_k$, and 2) for those values of k for which $x_k > f(w_1,\ldots,w_N)$, set $w_k = \overline{w}_k$. Consequently, to compute y_l or y_r, w_k switches only one time between $\overline{w}_k$ and $\underline{w}_k$, or between $\underline{w}_k$ and $\overline{w}_k$, respectively.

Putting all of these facts together, y_l in (6) and y_r in (7) can be expressed as (11) and (12), respectively. The KM algorithms summarized in Section II locate the switch points, and in general, the switch point for y_l, L, is smaller than the switch point for y_r, R.

ACKNOWLEDGMENT

The authors would like to thank Prof. H. Hagras for suggesting the study of Gaussian FOUs in Section IV-B.

REFERENCES

[1] S.-M. Guu, "Fuzzy weighted averages revisited," *Fuzzy Sets Syst.*, vol. 126, pp. 411–414, 2002.
[2] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Inf. Sci.*, vol. 132, pp. 195–220, 2001.
[3] F. Liu, "An efficient centroid type-reduction strategy for general type-2 fuzzy logic system," *Inf. Sci.*, vol. 178, no. 9, pp. 2224–2236, 2008.
[4] F. Liu and J. M. Mendel, "Aggregation using the fuzzy weighted average, as computed using the Karnik–Mendel algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 1, pp. 1–12, 2008.
[5] J. M. Mendel and F. Liu, "Super-exponential convergence of the Karnik–Mendel algorithms for computing the centroid of an interval type-2 fuzzy set," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 2, pp. 309–320, Apr. 2007.
[6] J. M. Mendel and D. Wu, *Perceptual Computing: Aiding People in Making Subjective Judgments.* Wiley-IEEE Press, to be published.
[7] J. M. Mendel and H. Wu, "New results about the centroid of an interval type-2 fuzzy set, including the centroid of a fuzzy granule," *Inf. Sci.*, vol. 177, pp. 360–377, 2007.
[8] J. M. Mendel, *Rule-Based Fuzzy Logic Systems: Introduction and New Directions.* Upper Saddle River, NJ: Prentice–Hall, 2001.
[9] J. M. Mendel, "On a 50% savings in the computation of the centroid of a symmetrical interval type-2 fuzzy," *Inf. Sci.*, vol. 172, no. 3, pp. 417–430, 2005.
[10] N. Quijano, K. M. Passino, and B. W. Andrews, "Foraging theory for multizone temperature control," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 18–27, Nov. 2006.
[11] D. Stephens and J. Krebs, *Foraging Theory.* Princeton, NJ: Princeton Univ. Press, 1986.
[12] D. Wu and J. M. Mendel, "Aggregation using the linguistic weighted average and interval type-2 fuzzy sets," *IEEE Trans. Fuzzy Syst.*, vol. 15, no. 6, pp. 1145–1161, 2007.
[13] D. Wu and J. M. Mendel, "Uncertainty measures for interval type-2 fuzzy sets," *Inf. Sci.*, vol. 177, no. 23, pp. 5378–5393, 2007.
[14] D. Wu and J. M. Mendel, "A vector similarity measure for linguistic approximation: Interval type-2 and type-1 fuzzy sets," *Inf. Sci.*, vol. 178, no. 2, pp. 381–402, 2008.
[15] D. Wu and W. W. Tan, "Computationally efficient type-reduction strategies for a type-2 fuzzy logic controller," in *Proc. FUZZ-IEEE*, Reno, NV, May 2005, pp. 353–358.
[16] H. Wu and J. M. Mendel, "Uncertainty bounds and their use in the design of interval type-2 fuzzy logic systems," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 5, pp. 622–639, Oct. 2002.

[10] These results are stated in the order $x_k < f(w_1,\ldots,w_N)$, and then, $x_k > f(w_1,\ldots,w_N)$ because x_k's in (A.1) are naturally ordered, i.e., $x_1 \leq x_2 \leq \cdots \leq x_N$. If x_k's are not so ordered, then they must first be so reordered.

Dongrui Wu (S'05) received the B.E. degree in automatic control from the University of Science and Technology of China, Hefei, China, in 2003 and the M.E. degree in electrical engineering from the National University of Singapore, Singapore, in 2005. He is currently working toward the Ph.D. degree in electrical engineering with the University of Southern California, Los Angeles.

His current research interests include control theory and applications, robotics, optimization, pattern classification, information fusion, computing with words, computational intelligence, and their applications to smart oilfield technologies.

Mr. Wu received the 2005 Best Student Paper Award from the IEEE International Conference on Fuzzy Systems, Reno, NV.

Jerry M. Mendel (S'59–M'61–SM'72–F'78–LF'04) received the Ph.D. degree in electrical engineering from the Polytechnic Institute of Brooklyn, Brooklyn, NY.

Since 1974, he has been with the Department of Electrical Engineering, University of Southern California, Los Angeles, where he is currently a Professor. He has authored or coauthored over 470 technical papers and is the author and/or editor of eight books, including *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions* (Prentice–Hall, 2001). His current research interests include type-2 fuzzy logic systems and their applications to a wide range of problems, including smart oilfield technology and computing with words.

Prof. Mendel is a Distinguished Member of the IEEE Control Systems Society. He was the President of the IEEE Control Systems Society in 1986 and is currently the Chairman of the Fuzzy Systems Technical Committee and an Elected Member of the Administrative Committee of the IEEE Computational Intelligence Society. Among his awards are the 1983 Best Transactions Paper Award from the IEEE Geoscience and Remote Sensing Society, the 1992 Signal Processing Society Paper Award, the 2002 Transactions on Fuzzy Systems Outstanding Paper Award, a 1984 IEEE Centennial Medal, an IEEE Third Millennium Medal, a Pioneer Award from the IEEE Granular Computing Conference, May 2006, for outstanding contributions in type-2 fuzzy systems, and the 2008 Fuzzy Systems Pioneer Award from the IEEE Computational Intelligence Society.