A Comment on “A Direct Approach for Determining the Switch Points in the Karnik–Mendel Algorithm”

Chao Chen, Member, IEEE, Dongrui Wu, Senior Member, IEEE, Jonathan Mark Garibaldi, Member, IEEE, Robert John, Senior Member, IEEE, Jamie Twycross, and Jerry M. Mendel, Life Fellow, IEEE

Abstract—This letter is a supplement to the previous paper “A Direct Approach for Determining the Switch Points in the Karnik–Mendel Algorithm”. In the previous paper, the enhanced iterative algorithm with stop condition (EIASC) was shown to be the most inefficient in R. Such outcome is apparently different from the results in another paper in which EIASC was illustrated to be the most efficient in MATLAB. An investigation has been made into this apparent inconsistency and it can be confirmed that both the results in R and MATLAB are valid for the EIASC algorithm. The main reason for such phenomenon is the efficiency difference of loop operations in R and MATLAB. It should be noted that the efficiency of an algorithm is closely related to its implementation in practice. In this letter, we update the comparisons of the three algorithms in the previous paper, based on optimized implementations under five programming languages (MATLAB, R, Python, C, and Java). From this, we conclude that results in one programming language cannot be simply extended to all languages.

Index Terms—Centroid, direct approach (DA), enhanced iterative algorithm with stop condition (EIASC), enhanced KM (EKM) algorithm, interval type-2 (IT2) fuzzy set, Karnik–Mendel (KM) algorithm.

I. INTRODUCTION

In the previous paper [1], a direct approach (DA) based on derivatives was proposed for determining the switch points in the Karnik–Mendel (KM) algorithm, for determining the lower and upper bounds of the centroid in type-2 inference. It was shown by simulations in R that DA clearly outperformed other algorithms irrespective of the shapes of fuzzy sets. Based on such results, it was suggested that DA should always be used when \( N \), which is the number of discretizations of the universe of discourse, is greater than or equal to 100.

In [1], differences were also found in the way EIASC performed in comparison with other algorithms, with some of these results apparently being different from the findings previously published in [2]. Specifically, the results in [1], in which all the algorithms were coded in R, showed not only that DA was the most efficient, but also that EIASC was generally much less efficient than the enhanced KM (EKM) algorithm. This is a significantly different finding to the results in [2], in which experiments were performed in MATLAB.

In this letter, we explore the above issue in detail, and update the results of comparisons in [1], based on optimized implementations under five programming languages (MATLAB, R, Python, C, and Java). Section II provides a clarification of the optimization of DA in practice. Section III presents experimental comparisons and Section IV discusses the results. Section V concludes the key findings in this letter.

II. DA ALGORITHM

While carrying out the implementation of the DA algorithm in the aforementioned five languages, several detailed optimizations were identified. This section clarifies the optimization of the DA implementation used in this letter.

It is well known that, in both R and MATLAB, operations on vectors with indices are much slower than operations on the same vectors without indices. The original DA algorithm, as shown as Algorithm 1 in [1], includes many such operations with indices. In practice, most of these operations can be optimized by eliminating the use of indices. However, the detailed manner in which this is done is dependent on the programming language, with some operations being quicker in MATLAB and other operations being quicker in R.

For example, in Step 1 of the original DA the implementation in R could use a for loop, as:

\[
\text{for} (i \in 2:N) \\
X_{\text{diff}}[i - 1] = X[i] - X[i - 1] \\
\]

or vectorized as

\[
X_{\text{diff}} = X[2:N] - X[1:(N - 1)]
\]
where the vectorized form is perhaps twenty times as fast as the original for loop.

While such vectorizations were used in the DA algorithm implementation in [1], it was subsequently noticed that the original form of the algorithm requires several reversals of lengthy $R$ vectors, which is also an inefficient operation in R. It was observed that these reversals could be removed through some minor alterations to intermediate variables. It was also noticed that DA can be further optimized by eliminating some unnecessary computations in the calculation of $c_l$ as follows. The type-2 centroid $c$ is defined as

$$c = \frac{\sum_{i=1}^{N} x_i u_i}{\sum_{i=1}^{N} u_i}.$$  

By differentiating $c$ with respect to $u_j$ we obtain

$$\frac{\partial c}{\partial u_j} = \frac{x_j - c}{\sum_{i=1}^{N} u_i},$$

which can be rearranged to

$$c = x_j - \frac{\partial c}{\partial u_j} \sum_{i=1}^{N} u_i.$$

By applying this transformation to the calculation of $c_l$ using switch-point $L$ we obtain

$$c_l = x_L - \frac{\partial c}{\partial u_L} \left( \sum_{i=1}^{L} \bar{u}_i + \sum_{i=L+1}^{N} u_i \right).$$  \hspace{1cm} (1)

As a result of these various optimizations, a revised version of the original DA algorithm, termed as the DA* algorithm, is shown as Algorithm 1.

### Algorithm 1: Pseudo Code in R of the Optimised Implementation Denoted DA* for Obtaining $L$, the Switch Point; and $c_l$, the Lower Bound of the Centroid.

**input**: $X, \bar{U}, U$, vectors of the primary variable, the upper membership grades, and the lower membership grades, respectively; $x_i, \bar{u}_i$, and $u_i$ denote elements of the respective vectors;  
**output**: $L$, the switch point; $c_l$, the lower bound of the centroid;  
1. $X' \leftarrow \{x_i - x_{i-1} | 0 = i = 2, 3, \ldots, N\}$, a vector of consecutive differences of elements of $X$ and an extra zero;  
2. $S^l \leftarrow \{\sum_{i=1}^{N} \bar{u}_i | j = 1, 2, \ldots, N\}$, a vector of the cumulative sum of the elements of $\bar{U}$;  
3. $S^u \leftarrow \{\sum_{i=1}^{N} u_i | j = 1, 2, \ldots, N\}$, a vector of the cumulative sum of the elements of $U$;  
4. $S^u \leftarrow \{s^u_i - s^u_i | i = 1, 2, \ldots, N\}$, where $s^u_i$ is the $i$th element of vector $S^u$;  
5. $T^l \leftarrow \{x'_i, s'_i | i = 1, 2, \ldots, N\}$, where $x'_i$ and $s'_i$ are the $i$th element of vectors $X'$ and $S^l$ respectively;  
6. $T^U \leftarrow \{x'_i, s'_i | i = 1, 2, \ldots, N\}$, where $x'_i$ and $s'_i$ are the $i$th element of vectors $X'$ and $S^u$ respectively;  
7. $d^N \leftarrow \sum_{i=1}^{N} t^N_i$, where $t^N_i$ is the $i$th element of vector $T^N$;  
8. $D \leftarrow \{\sum_{i=1}^{N} (t^F_i + t^N_i) | j = 1, 2, \ldots, N\}$, where $t^F_i$ is the $i$th element of vector $T^F$;  
9. Find the smallest $k \in 1, 2, \ldots, N - 1$ such that $d_k \geq d^N$, where $d_k$, which represents $\left(\frac{\partial c}{\partial u_{k+1}} + d^N\right)$, is the $k$th element of the vector $D$;  
10. if $k$ exists then $L \leftarrow k$ else $L \leftarrow N - 1$;  
11. if $L \neq 1$ then $\frac{\partial c}{\partial u_L} \leftarrow d_{L-1} - d^N$ else $\frac{\partial c}{\partial u_L} \leftarrow -d^N$;  

Compute $c_l$ by Equation (1);

A. Generalized Bell-Shaped INTERVAL TYPE-2 (IT2) Fuzzy Sets

It was assumed that the vector $X$ (the discrete universe of discourse), containing $x_i$ (primary variable), is uniformly distributed from 0 to 10. $\bar{u}_i$ and $u_i$ (membership grades) are defined by generalized bell-shaped function

$$\bar{u}_i = \frac{1}{1 + \left(\frac{x_i - \bar{a}}{b}\right)^2},$$

$$u_i = \frac{1}{1 + \left(\frac{x_i - \bar{a}}{b}\right)^2},$$

where $\bar{a}$ and $b$ are randomly selected between 1 and 2; $\bar{a}$ is the multiplication of $\bar{a}$ with a random number between 1 and 2; $c$ is a random number between 0 and 10.

B. Generalized Randomly-Shaped IT2 Fuzzy Sets

It was assumed that vectors $X$ and $\bar{U}$, containing $x_i$ and $\bar{u}_i$, respectively, are randomly generated values from 0 to 1, based on the uniform distribution. $\bar{u}_i$ is the multiplication of $\bar{u}_i$ with another random number between 0 and 1, so that $\bar{u}_i$ is also within the range of 0 to 1.

C. Comparisons

Comparisons were made for the above two types of IT2 fuzzy sets separately. In each comparison, $N$, which is the length of $X$ (i.e., the number of discretizations across the universe of discourse), was set to be 10, 200, 400, . . . , and 2000 (11 different values of $N$). A total of 5000 Monte-Carlo simulations were made for each value of $N$, and the time costs for computing the centroids were aggregated to be compared for each algorithm.

The platform was a Macbook Pro (13-in, 2017) with 3.10-GHz Intel Core i5 processor and 16-GB 2133-MHz LPDDR3 memory, running macOS High Sierra version 10.13.1. The programming languages and software environment were R x64 version 3.4.2, MATLAB R2017b, Python 3.6.3, Apple
It should be noted that, though it was not mentioned in [3], extra operations are required for the EKM algorithm to avoid infinite loops caused by numerical issues. In [1], some inefficient operations were applied within the EKM implementation to avoid the infinite-loop issue, which made the EKM algorithm less efficient than it should have been. For the comparisons in this letter, the inefficient operations used in [1] have been replaced by more efficient versions.

In summary, it is clear that the computational efficiency of an algorithm is closely related to the platform, and how it is implemented. Note that, in computer science, the dependence on languages is usually avoided by focusing more on the orders of algorithms (using big O notation). In this letter, it can be easily derived that the asymptotic time complexity of all algorithms is \( O(N) \), which is linear.

V. CONCLUSION

In this letter, we updated the comparisons in [1] under five commonly used programming languages. Results showed that EIASC performed the best in MATLAB, C, and Java, but it is the worst in R and Python. Both DA* (the slightly revised version of DA) and EKM showed the best performance in R and Python, with broadly similar results. Since the performance of algorithms is closely related to the implementations, we have made our implementations accessible online (https://gitlab.com/chao.chen/DA_Letter_2017.git).

We suggest that future proposals of related algorithms should focus more on the orders of algorithms and take care as to the conclusions drawn, which may be dependent on the platform used (language and implementation environment).

REFERENCES