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 M. Garibaldi, *Member, IEEE,* Robert John, *Senior Member, IEEE,* Jamie Twycross, and Jerry 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 optimised 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, interval type-2 (IT2) fuzzy set, Karnik-Mendel (KM) algorithm, enhanced KM (EKM) algorithm, enhanced iterative algorithm with stop condition (EIASC), direct approach (DA).**

I. Introduction

In a 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 bound of the centroid in type-2 inference. It was shown by simulations in R that DA clearly outperformed other algorithms regardless of the shapes of fuzzy sets. Based on such results, it was suggested that DA should always be used when N, the number of discretisations of the universe of discourse, is greater than or equal to 100.

C. Chen, J. M. Garibaldi, R. John and J. Twycross are with the Laboratory for Uncertainty in Data and Decision Making (LUCID), the Intelligent Modelling and Analysis (IMA) and the Automated Scheduling Optimisation and Planning (ASAP) Research Groups, School of Computer Science, University of Nottingham, Nottingham, Jubilee Campus, NG8 1BB UK e-mail: {chao.chen, jon.garibaldi, robert.john, jamie.twycross}@nottingham.ac.uk.

D. Wu is Professor in the Key Laboratory of the Ministry of Education for Image Processing and Intelligent Control, School of Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074 China. E-mail: drwu@hust.edu.cn.

J. M. Mendel is Professor Emeritus at University of Southern California, Los Angeles, CA 90089-2564, USA (e-mail: mendel@sipi.usc.edu), and Tianjin 1000-Talents Foreign Experts Plan Endowment Professor and Honorary Dean of the College of Artificial Intelligence, Tianjin Normal University, Tianjin, China.

Manuscript received *** **, 2017; revised *** **, 2017; accepted *** **, 2017. Date of publication *** **, 2017; date of current version*** **, 2017.
Corresponding author: Dongrui Wu

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 optimised implementations under five programming languages (Matlab, R, Python, C and Java). Section II provides a clarification of the optimisation of DA in practice. Experimental comparisons are made in Section III and results are then discussed in Section IV. Section V concludes the key findings in this letter.

II. The DA algorithm

Whilst carrying out the implementation of the DA algorithm in the five languages mentioned above, several detailed optimisations were identified. This section clarifies the optimisation 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 optimised 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:

```
for(i in 2:N)
    Xdiff[i-1] = X[i] - X[i-1]
```

or vectorised as:

```
Xdiff = X[2:N] - X[1:(N-1)]
```

where the vectorised form is perhaps twenty times faster than the original `for` loop.

Whilst such vectorisations 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.

input : $X, \bar{U}, \underline{U}$, vectors of the primary variable, the upper membership grades, and the lower membership grades, respectively; x_i, $\bar{u}_i$, and $\underline{u}_i$ denote elements of the respective vectors;
output: L, the switch point; c_l, the lower bound of the centroid;

$X' \leftarrow \{x_i - x_{i-1}, 0 \mid i = 2, 3, ..., N\}$, a vector of consecutive differences of elements of X and an extra zero;
$S^1 \leftarrow \{\sum_{i=1}^{j} \bar{u}_i \mid j = 1, 2, ..., N\}$, a vector of the cumulative sum of the elements of $\bar{U}$;
$S^{2a} \leftarrow \{\sum_{i=1}^{j} \underline{u}_i \mid j = 1, 2, ..., N\}$, a vector of the cumulative sum of the elements of $\underline{U}$;
$S^2 \leftarrow \{s_N^{2a} - s_i^{2a} \mid i = 1, 2, ..., N\}$, where s_i^{2a} is the i^{th} element of vector S^{2a};
$T^P \leftarrow \{x'_i \cdot s_i^1 \mid i = 1, 2, ..., N\}$, where x'_i and s_i^1 are the i^{th} element of vectors X' and S^1 respectively;
$T^N \leftarrow \{x'_i \cdot s_i^2 \mid i = 1, 2, ..., N\}$, where x'_i and s_i^2 are the i^{th} element of vectors X' and S^2 respectively;

$d^N \leftarrow \sum_{i=1}^{N} t_i^N$, where t_i^N is the i^{th} element of vector T^N;
$D \leftarrow \{\sum_{i=1}^{j} \left(t_i^P + t_i^N\right) \mid j = 1, 2, ..., N\}$, where t_i^P is the i^{th} element of vector T^P;
Find the smallest $k \in 1, 2, ..., N-1$ such that $d_k \geqslant d^N$, 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;
if k *exists* **then** $L \leftarrow k$ **else** $L \leftarrow N - 1$;
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);

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.

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 optimised 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:

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

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

III. Experimental Comparison

Optimised implementations of DA*, EKM and EIASC under five programming languages (Matlab, C, Java, R and Python) were created, and comparisons were made based on the two generalised examples given in Section VI.B of [1].

A. Generalised bell-shaped 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 $\underline{u}_i$ (membership grades) are defined by generalised bell-shaped function:

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

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

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

B. Generalised 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. $\underline{u}_i$ is the multiplication of $\bar{u}_i$ with another random number between 0 and 1, so that $\underline{u}_i$ 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 discretisations across the universe of discourse), was set to be 10, 200, 400, ..., and 2000 (11 different values of N). For each value of N, 5000 Monte Carlo simulations were made and the time costs for computing the centroids were aggregated to be compared for each algorithm.

The platform was a Macbook Pro (13-inch, 2017) with 3.10GHz Intel Core i5 processor and 16GB 2133 MHz LPDDR3 memory, running macOS High Sierra version 10.13.1. The programming languages and software environment are R x64 version 3.4.2, Matlab R2017b, Python 3.6.3, Apple LLVM version 9.0.0 (clang-900.0.38) for C (compiled with options -O3 and -std=c99), and Java™ SE Development Kit 9.0.1.

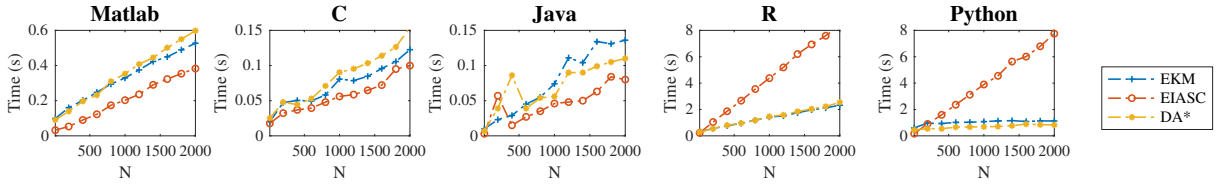

Fig. 1: Practical computational cost comparisons based on generalised bell-shaped fuzzy sets.

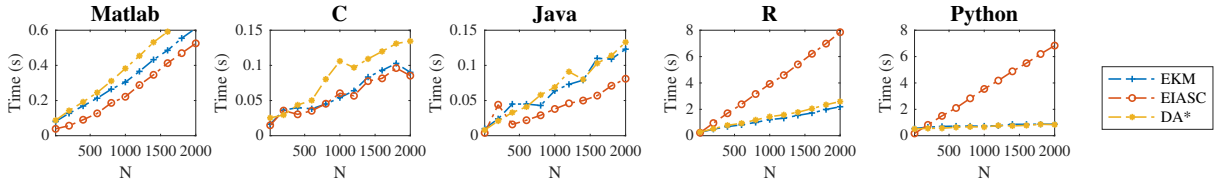

Fig. 2: Practical computational cost comparisons based on generalised random-shaped fuzzy sets.

Note that sorting of the vector X is not included in time comparisons, since all the three algorithms require X to be sorted prior to starting. Results of the comparisons can be seen in Figs. 1 and 2. As can be observed in these figures, EIASC performs best in Matlab, C and Java, but much worse in R and Python. On the other hand, based on these optimised implementations, the DA* and EKM algorithm perform similarly, both being more efficient that EIASC in R and Python.

IV. Discussion

As a result of the new optimised implementations for each of the algorithms and detailed experiments performed on a single computer, it is now clear that the statement made in the original paper ("that DA should always be used when N, the number of discretisations of the universe of discourse, is greater than or equal to 100") is not correct. Rather, we have found a far more complex overall picture.

It is interesting that EIASC performs the best in Matlab, C and Java, but it is the worst in R and Python. This is mainly due to the efficiency difference of functions or operations in different programming languages. For example, loop operations (iterations), on which the EIASC algorithm heavily relies, are much less efficient in R than they are in Matlab. It should be noted that it is common for loops to be inefficient in interpreted programming languages such as R and Python. Thus, in such programming languages, the use of EIASC should be carefully considered especially when time efficiency is sensitive.

Regardless of the algorithm used, the computational time differences between programming languages is very large. Thus, for example, using an efficient compiled language such as C over Matlab makes more of a difference than the choice of algorithm.

Whilst the DA* algorithm is not clearly better than EKM, nevertheless, the centroid is found without the need for multiple iterations, as mentioned in the Introduction of [1]. This may make the algorithm more desirable for real-time control problems when the calculation time of the algorithm needs to be known in advance.

It should be noted that, though it was not mentioned in [3], extra operations are required for the enhanced Karnik-Mendel (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 order 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 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 on-line (https://gitlab.com/chao.chen/DA_Letter_2017.git).

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

References

[1] C. Chen, R. John, J. Twycross, and J. M. Garibaldi, "A direct approach for determining the switch points in the Karnik-Mendel algorithm," *IEEE Transactions on Fuzzy Systems*, vol. PP, no. 99, p. 1, 2017.

[2] D. Wu, "Approaches for Reducing the Computational Cost of Interval Type-2 Fuzzy Logic Systems: Overview and Comparisons," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 1, pp. 80–99, 2013.

[3] D. Wu and J. M. Mendel, "Enhanced Karnik–Mendel algorithms," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 4, pp. 923–934, 2009.