An Extension of the FuzzyR Toolbox for Non-Singleton Fuzzy Logic Systems

Chao Chen, Yu Zhao, Christian Wagner, Direnc Pekaslan, and Jonathan M. Garibaldi
Intelligent Modelling and Analysis Group (IMA),
Lab for Uncertainty in Data and Decision Making (LUCID),
School of Computer Science, University of Nottingham,
Nottingham, NG8 1BB, UK
{chao.chen, alyyz18, christian.wagner, direnc.pekaslan, jon.garibaldi}@nottingham.ac.uk

2021 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) | 978-1-6654-4407-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/FUZZ45933.2021.9494472

***Abstract*—Recent years have seen a surge in interest in non-singleton fuzzy systems. These systems enable the direct modelling of uncertainty affecting systems' inputs using the fuzzification stage. Moreover, recent work has shown how different composition approaches to modelling the interaction between the non-singleton input and the antecedent fuzzy sets enable the efficient handling of uncertainty without requiring changes in a system's rule base, with benefits both in terms of performance and interpretability. As thus far few current software toolkit support non-singleton fuzzy systems, this paper presents an extension of the FuzzyR toolbox, which is a freely available R package on CRAN, for non-singleton fuzzy logic systems. The updated toolbox enables a non-singleton model to be conveniently built from scratch, or for existing singleton fuzzy logic systems built using FuzzyR to be converted easily. Pre-defined operations include fuzzification of crisp inputs (e.g. into Gaussian membership functions), and a variety of composition approaches for computing rules' firing-strengths, based on the standard, centroid-based, and similarity-based methods. It is also possible to include user-defined options for these above-mentioned methods, without the need to modify (or update) the FuzzyR toolbox itself. In this paper, detailed introductions for the new non-singleton features of the toolkit are presented, complete with code samples in R to facilitate adoption both within and beyond the community. Further, the paper presents a series of validation experiments, replicating a recent empirical analysis of non-singleton fuzzy logic systems in the context of time-series prediction with different levels of noise.**

***Index Terms*—FuzzyR; non-singleton; fuzzy logic systems; fuzzy inference; toolkit; software; open-source.**

I. Introduction

Non-singleton fuzzy logic systems (NSFLS) are an extension of classical singleton fuzzy logic systems (FLS) [1, 2]. They are ideally suited for dealing directly with input uncertainty, which is one of the main types of uncertainty in real-world applications. The efficacy of NSFLSs has been demonstrated in different real-world applications [3, 4]. More recently, some new types (e.g. centroid-based, similarity-based) of NSFLSs have been introduced and shown to produce performance than standard NSFLS [5, 6]. In particular, a new method for NSFLS, ADONiS, has been shown to provide competitive performance while preserving strong interpretability, compared to FLSs optimized using machine learning – which commonly achieve good performance at the expense of interpretability [7, 8]. Some of these new NSFLSs have been successfully applied to real-world applications such as unmanned aerial vehicle (UAV) control applications [9, 10].

As discussed in [11], we believe that both for the reproducibility of research outputs as well as to enable the more widespread adoption of NSFLSs, the provision of free, widely available, open-source software is essential. A number of different toolkits for fuzzy logic systems, including type-1 and type-2, have been made available for research and application in recently years [11, 12, 13, 14, 15, 16, 17, 18]. However, to the best of our knowledge, no comprehensive toolkit for NSFLSs has been released.

When considering the programming lanaguage of such toolkits, R is particularly well suited. It has the advantages of being both free and open-source, and is used widely across disciplines, from computer science to the social sciences [19]. FuzzyR, which is aimed to be a comprehensive toolbox for fuzzy logic systems, has been published as a free and open-source R package on CRAN [20]. As described in [18], it already includes many features such as functions for building and evaluating type-1 and interval type-2 FLSs, as well as basic building blocks for the traditional non-singleton fuzzy logic systems (but no end-to-end implementation). Optimisation is also available based on an extended ANFIS architecture [21]. There is a function for performance evaluation based on eight different accuracy measures including a new accuracy measure UMBRAE proposed in [22]. Additionally, graphical user interfaces have been provided, which can be very useful for the purposes of teaching and learning.

We believe that this paper, which presents an extension of the FuzzyR toolbox for NSFLSs, can help disseminate NSFLS research and advances, and encourage continuing contributions to the development and application of NSFLSs.

This paper is structured as follows. Section II provides the background of NSFLSs as well a review of the current availability of FLS toolkits and packages. Section III summarizes and describes the new features of the non-singleton extension. Demonstrations for different type of NSFLSs are provided in Section IV. Section V is the conclusion.

II. BACKGROUND

This section provides some essential background information for NSFLSs to establish terminology and notation..

A. Fuzzifier

A key difference between singleton FLS and NSFLS is the fuzzifier used for the fuzzification process, which converts a crisp input to a fuzzy set. As shown in Fig. 1, a singleton FLS uses a singleton fuzzifier which converts a crisp input x' into a fuzzy singleton - a fuzzy set which has only one non-zero membership grade ($\mu(x') = 1$) at the point of x'. However, as inputs in reality are commonly associated with uncertainty such as arising from measurement, non-singleton fuzzy systems enable the transformation of crisp inputs into non-singleton fuzzy sets. Hence, with non-singleton fuzzification, the crisp input x' is converted to a (commonly) normal fuzzy set, with non-zero membership grades at points other than x', modelling uncertainty associated with the input. Note that a variety of membership functions are commonly used for fuzzification, including Gaussian, triangular, trapezoidal, etc. Further, if inputs are available 'natively' as fuzzy sets, these can be used directly as inputs, skipping the fuzzification stage. Finally, in addition to the most commonly used type-1 fuzzification, it is possible to use other types of fuzzy sets, such as interval type-2 or general type-2 fuzzy sets as part of for NSFLSs. The choice of fuzzification methods should be relying on the features of the uncertainties in inputs for specific applications [5]. For example, the widths (range of support) of input fuzzy sets after fuzzification should be associated with the level of uncertainty (wider width may indicate higher uncertainty along the original input scale).

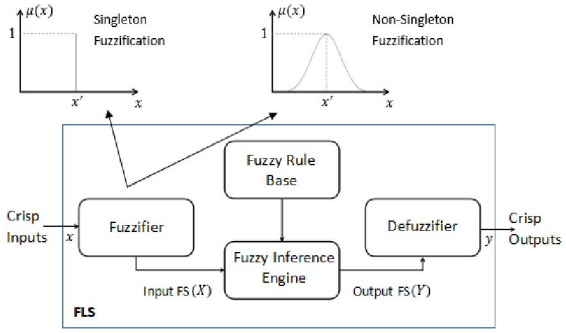

Fig. 1. An illustration of singleton and non-singleton fuzzification methods in a FLS. Figure is adapted from [6].

B. Inference Engine: Input Firing Method

A key step in the inference engine is to get the firing strength for each rule. To do this, it is necessary to calculate the composition of each fuzzified input's membership function with the corresponding antecedent's membership function. Then, the rule's firing strength is calculated by applying a t-norm to all firing degrees of all the rule's inputs.

With a singleton fuzzifier, the input firing strength is equal to the membership degree of the crisp input value to the antecedent membership function (see Fig. 2 for an illustration).

With a non-singleton fuzzifier, different methods may be used to generate the input firing strength. In FuzzyR and this paper, three types of input firing methods are considered and briefly introduced below.

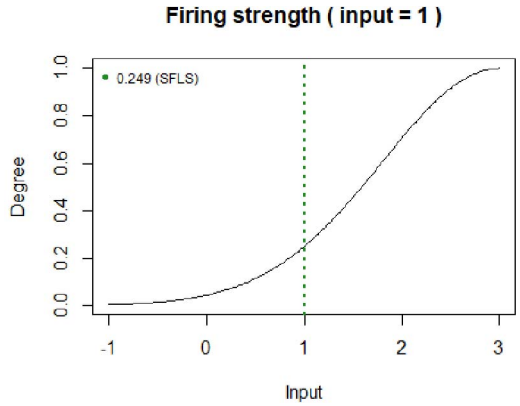

Fig. 2. An illustration of the input firing strength for SFLS.

1) Standard: The standard method is the most commonly used method for NSFLSs [5]. As described in [23], the intersection of the fuzzified input and its corresponding antecedent in a rule is calculated. Then, the input firing strength is given by the maximum membership grade of this intersection.

2) Centroid-based: Pourabdollah et al. introduced the centroid-based method for NSFLSs [5]. The first step of this method is the same as for the standard method, i.e. to capture the interaction between input and antecedent fuzzy set in a given rule, their intersection is computed – again a fuzzy set. Then the input firing strength is given by the degree of membership of the centroid (i.e. the centre of gravity) of the intersection, in the intersection fuzzy set.

3) Similarity-based: The similarity-based NSFLS was considered in [6]. It is a different type of method to the two types of methods above in the sense that it does not rely on intersection in order to determine the interaction between sets. Instead, the input firing strength is the degree of similarity between the input and the corresponding antecedent fuzzy sets. The similarity-based method is generic and any suitable similarity measure can be chosen depending on the application. One of the most commonly used similarity measures for fuzzy sets is the Jaccard similarity [24], but subsethood for example may also be appropriate [25].

Fig. 3 is an illustration of the firing strengths obtained based on three types of input firing methods mentioned above, based on a toy example of an input and antecedent fuzzy set.

III. The Non-Singleton Extension of FuzzyR

This section introduces the new non-singleton functionality in FuzzyR, as well as additional, complementary support functions and examples.

A. Features Overview

With the non-singleton extension, FuzzyR provides functions for creating and evaluating different types of NSFLSs.

To make it easier for both current and new users of FuzzyR to be familiar with the new non-singleton features, the changes of existing functions are summarised in Table I. Newly added functions are summarised in Table II.

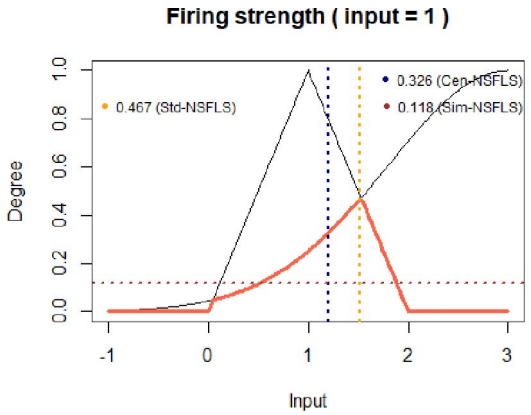

Fig. 3. An illustration of the firing strength obtained based on three different types of methods: standard (Std-NSFLS), centroid-based (Cen-NSFLS) and similarity-based (Sim-NSFLS). The Gaussian and Triangular membership functions are the antecedent and the fuzzified input respectively.

TABLE I
SUMMARY OF CHANGED FUNCTIONS

Function name	Description	Changes
addvar	Adds the linguistic variable for an input (or output) of a model of fuzzy inference system (fis).	Adding an additional parameter 'firing.method' as the choice of input firing method for non-singleton fuzzification.
evalmf	The function for getting the input firing strength.	Adding support for inputs with non-singleton fuzzification. Extra parameters (fuzzification.method, fuzzification.params, firing.method, input.range) are required.

TABLE II
SUMMARY OF NEW FUNCTIONS

Function name	Description
convertfis	This function can be used to convert an existing fis object from singleton fuzzification to non-singleton fuzzification.
x.fuzzification	To convert a crisp input x to a membership function with specified fuzzification method. It can be singleton or non-singleton fuzzification (e.g. gauss.fuzzification).
fuzzy.firing.defuzz	To get the input firing strength with a specified defuzziifcation, such as the centroid-based firing method.
fuzzy.firing.similarity.set	To get the input firing strength based on the set-theoretic similarity measure (the Jaccard similarity [24] is used by default).
cmp.firing	To compare different input firing methods for a given crisp input, fuzzification method, and corresponding antecedent membership function. See Fig. 3 for an example.
tipper.ns	An illustration function for creating a fis object for the waiter-tipping problem based on non-singleton fuzzification and firing methods. The function is most useful when referring to the source code as a demo for implementing a NSFLS with FuzzyR.

B. Fuzzification Methods

There are some pre-defined fuzzification methods (both for type-1 and interval type-2) available to use in FuzzyR. It is also easy to add user-defined fuzzification methods without making any changes to the FuzzyR toolbox itself as detailed below.

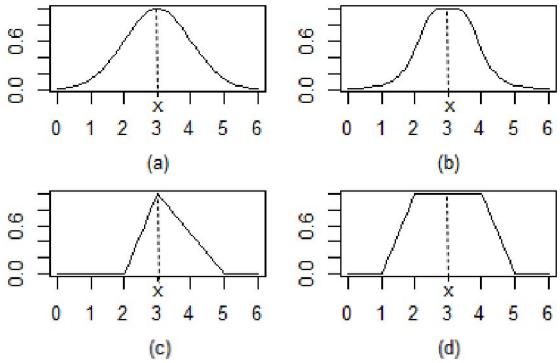

Fig. 4. Examples of Non-Singleton fuzzification produced using the R toolkit.

1) Pre-defined: Please refer to the manual of FuzzyR for a full list of pre-defined fuzzification methods. Fig. 4 shows some examples of fuzzified inputs based on different fuzzification methods. Below is a code example of using the Gaussian fuzzification method to generate the fuzzified membership function for a crisp input and corresponding fuzzification parameter (the standard deviation for the Gaussian membership function).

```
x <- 3
mf <- x.fuzzification('gauss.fuzzification', x, 1)
```

2) User-defined: As mentioned above, users can define their own fuzzification methods. The code snippet below is an example of how to define a trapezoidal fuzzification method which can produce the fuzzified input illustrated in Fig. 4(d).

```
mytrapezoidal.fuzzification <-
    function(x, params) {
        a <- params[1]
        b <- params[2]
        mf.params <- c(x - a, x - b, x + b, x + a)
        genmf('trapmf', mf.params)
    }

x <- 3
mf <- x.fuzzification('mytrapezoidal.fuzzification',
                      x, c(2, 1))
```

C. Input Firing Methods

As mentioned above, different input firing methods are available to use in FuzzyR. Please refer to the usage of function *addvar* for a detailed listing of built-in firing methods. At the time of writing, the built-in similarity-based firing method only supports the Jaccard similarity. However, it is possible for users to add other firing methods based on alternative similarity methods such as Sørensen–Dice [26, 27]. The name of user-defined similarity-based firing function should follow the form of *fuzzy.firing.similarity.xxx*, where *xxx* is the type of similarity measure used. Once the function has been defined, the type *xxx* can be used as a parameter of function *addvar*.

D. Creating a NSFLS

To create a NSFLS model, it is essential to use the following settings: the fuzzification method and its corresponding

parameters, and the input firing method. There are two ways of buiding a non-singleton FLS. It can be built from scratch, or it can be easily converted from an existing singleton model built with FuzzyR.

Below is an example to convert an existing singleton model to its non-singleton counterpart. Note that the *tipper* function returns a singleton model. More details about how to build a singleton model can be found in the source code of the *tipper* function, which is available with the FuzzyR package on CRAN.

```
fis.singleton <- tipper()
fuzzification.method <- 'gauss'
fuzzification.params <- 1
fis.nonsingleton <- convertfis(fis.singleton, 's2n',
     fuzzification.method, fuzzification.params)
```

To build a non-singleton model from scratch, the key difference from building a singleton model is that the above mentioned parameters for non-singleton have to be used with the function *addvar*. See an example below for using *addvar*. Note that the 'tnorm.min.max' method is the standard method discussed in Section II-B.

```
fis <- newfis('hello_non-singleton_fis')
domain <- c(0, 10)
fuzzification.method <- 'gauss'
fuzzification.params <- 1
firing.method <- 'tnorm.min.max'
fis <- addvar(fis, 'input', 'service', domain,
    fuzzification.method, fuzzification.params,
    firing.method)
```

A complete example for creating a non-singleton model can be found in the source code for function *tipper.ns*. Note that there is no difference in using function *evalfis* for a non-singleton model (compared to a singleton model) to get outputs for given specific input values. For example,

```
fis <- tipper.ns()
input <- matrix(sample(0:10, 10, rep=T), ncol = 2)
evalfis(input, fis)
```

IV. Experimental Demonstration

In this section, we demonstrate the use of the non-singleton extension of FuzzyR to compare different NSFLS architectures (i.e. standard, centroid-based and similarity-based). The same methodology from the studies in [5, 6] is followed and chaotic time-series forecasting experiments are implemented. Note that the aim of the experiments in this paper is *not* to achieve the best or even high prediction performance, but to validate that the results of the FuzzyR extension are in line with the literature and to demonstrate a practical application of the toolkit.

A. Experiment Data

1) Mackey-Glass time series: The well known chaotic Mackey-Glass (M-G) time series is used in the forecasting experiments. As in the studies [5, 6], initially 2000 samples (from $t = -999$ to $t = 1000$) are generated and, in order to avoid fluctuations in the initial part of the time series, only the last 1000 (from $t = 1$ to $t = 1000$) points are preserved for use in the experiments. The system is trained from $t = 1$ to $t = 700$ and tested from $t = 701$ to $t = 1000$. Fig.5 shows a pre-computed example of the noise-free time series in training and testing intervals.

With the ground truth pre-computed using the M-G equations, the performance of different systems can be easily quantified by MSE, standard deviation or other error measurement, and can then be compared each other to come to a conclusion.

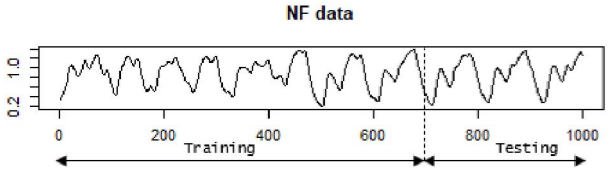

Fig. 5. Illustration of the pre-computed (noise-free) time series [28].

2) Additive noise: Two different levels of white Gaussian noise (10 db and 5 db) are added to the generated M-G time series dataset in different variant of experiments. The training and testing data of the M-G time series used in our experiments are illustrated in Fig. 6.

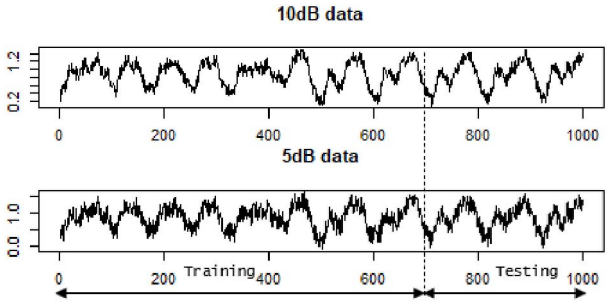

Fig. 6. Examples of the training and the testing data of the the MG time series at NF data and three different SNR level. Where value of delta $\delta = 0.030$ at SNR=10dB and $\delta = 0.096$ at SNR=5dB.

Training data was used to generate the rules for all FLSs based on the Wang-Mendel (W-M) method [29] , as well as to find the noise level for determining the fuzzification parameter.

More details on M-G time-series generation, noise adding and rule-generation can be found in [5, 6].

B. The comparison of different NSFLSs

In the prediction experiments, 9 sequential (over time) inputs are used to predict the following 1 output (one-step ahead prediction), as carried out in the studies [5, 6]. In total, five different prediction experiments are implemented and in each of these experiments, three different NSFLSs (std-NSFLS, Cen-NSFLSs and Sim-NSFLSs) are compared.

In the first experiment, rule-generation is implemented with the noise-free training dataset $t = 1$ to $t = 700$ (See Fig. 5) and noise-free testing is carried out. Since the testing dataset does not contain any noise, the singleton fuzzy logic system (SFLS) is built and the inputs are processed as singleton FSs.

In the second experiment, the same rule-set is adopted from the noise-free rule generation and 10 dB noisy dataset ($t = 700$ to $t = 1000$) is used in the testing. Since the

testing dataset contain 10 dB noise, NSFLSs are built and utilised -with Gaussian input FSs- in the fuzzification step. The experiment is repeated three times by using std-NSFLS, cen-NSFLSs and sim-NSFLSs in each variant.

In the third experiment, the same noise-free rule-set and 5 dB noisy dataset is used in the testing. As it is practised in the previous experiment, NSFLSs are built and utilised to cope with noisy conditions of testing dataset. Again std-NSFLS, cen-NSFLSs and sim-NSFLSs results are obtained.

In the fourth experiment, rule-generation is implemented with 10 dB noisy training dataset (See Fig. 6), and testing is implemented with the 10 dB noisy dataset.

In the last experiment, 5 dB noisy dataset (See Fig. 6) is used in the rule-generation and 5 dB testing is done along with std-NSFLS, cen-NSFLSs and Sim-NSFLSs architectures.

C. Results

In order to mitigate the effect of randomness, each experiment was repeated 30 times and the averaged MSEs have been summarised in Fig. 7.

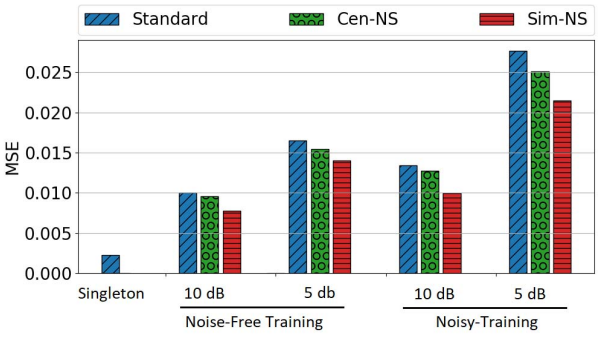

Fig. 7. The M-G time-series prediction performance produced by the different (N)FLSs based on average MSEs.

It is clear that in all experiments here, Sim-NSFLS shows the best performance, followed by Cen-NSFLS, then Std-NSFLS. For comparison, the result produced can be found in [5][6]. Note again that we are not suggesting that this behaviour is generalizable, i.e. in different applications, different NSFLS designs may be advantageous. In this paper, the experiments largely take the role of a demonstrator of the toolkit. All the source code for the experiments is available from the authors[1].

On average over all the experiments, both sim-NSFLS and cen-NSFLS show a reduced deviation (up to 17.31% and 31.57% respectively) from the pre-computed time series and thus better performance in comparison to the sta-NSFLS.

When the previous studies [5, 6] results are scrutinised, it can be seen that the FuzzyR package results are in line with the previous results and in all of sub-experiments, sim-NSFLS perform the best, followed by cen-NSFLS, then std-NSFLS. A visualisation of the fifth experiment (5 db noisy training and 5 dB testing) prediction for all sta-NSFLS, cen-NSFLS and sim-NSFLS can be seen in Fig. 8.

[1]https://www.lucidresearch.org/software.html

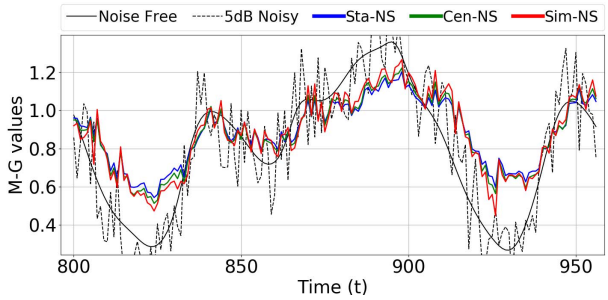

Fig. 8. The result produced by new toolkit in the 5 dB noisy training and 5 dB noisy testing.

Overall, the experiments in the literature discussed above could be replicated with the non-singleton extension of FuzzyR.

V. CONCLUSIONS

This paper presented an extension of the FuzzyR toolbox for non-singleton fuzzy logic systems. FuzzyR is freely available on CRAN and we only ask authors/developers to cite this paper and [18] when using the toolkit. Essential background and new features for NSFLS were briefly introduced, with further detail on the usage of the functions available in the manual or source code of FuzzyR. The toolbox was evaluated and its use demonstrated as part of building and comparing SFLS and NSFLS models, based on Mackey-Glass time series prediciton experiments replicated from the literature. It should be mentioned that the purpose of experiments in this paper is not to achieve best performance of SFLS and NSFLS models, but to validate the toolbox and provide the reader with a tangible demonstration of its use. Given that the software resources available for non-singleton fuzzy logic systems are very limited, we are hopeful that the toolbox and this paper can support the accessibility of non-singleton fuzzy logic systems to researchers from within and beyond the computer science community. NSLFSs have considerable potential, not least in offering an elegant approach to handling uncertainty and noise affecting system inputs without impacting a system's rules and overall interpretability. On the other hand, the integration of the toolkit with standards such as FML [30] is an interesting avenue for future work.

REFERENCES

[1] G. C. Mouzouris and J. M. Mendel, "Non-singleton fuzzy logic systems," in *Proceedings IEEE International Conference on Fuzzy Systems*, 1994, pp. 456–461.

[2] ——, "Nonsingleton fuzzy logic systems: theory and application," *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 1, pp. 56–71, 1997.

[3] A. B. Cara, C. Wagner, H. Hagras, H. Pomares, and I. Rojas, "Multiobjective Optimization and Comparison of Nonsingleton Type-1 and Singleton Interval Type-2 Fuzzy Logic Systems," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 3, pp. 459–476, 2013.

[4] A. Pourabdollah, C. Wagner, M. Smith, and K. Wallace, "Real-world utility of non-singleton fuzzy logic systems:

A case of environmental management," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2015, pp. 1–8.

[5] A. Pourabdollah, C. Wagner, J. H. Aladi, and J. M. Garibaldi, "Improved Uncertainty Capture for Nonsingleton Fuzzy Systems," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 6, pp. 1513–1524, 2016.

[6] C. Wagner, A. Pourabdollah, J. McCulloch, R. John, and J. M. Garibaldi, "A similarity-based inference engine for non-singleton fuzzy logic systems," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2016, pp. 316–323.

[7] D. Pekaslan, C. Wagner, and J. M. Garibaldi, "ADONiS - Adaptive Online Non-Singleton Fuzzy Logic Systems," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 10, pp. 2302 – 2312, 2020.

[8] D. Pekaslan, C. Chen, C. Wagner, and J. M. Garibaldi, "Performance and Interpretability in Fuzzy Logic Systems – Can We Have Both?" in *Proceedings Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2020, pp. 571–584.

[9] C. Fu, A. Sarabakha, E. Kayacan, C. Wagner, R. John, and J. M. Garibaldi, "A comparative study on the control of quadcopter UAVs by using singleton and non-singleton fuzzy logic controllers," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2016, pp. 1023–1030.

[10] ——, "Similarity-based non-singleton fuzzy logic control for improved performance in UAVs," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2017, pp. 1–6.

[11] C. Wagner, "Juzzy - A Java based toolkit for Type-2 Fuzzy Logic," in *Proceedings IEEE Symposium on Advances in Type-2 Fuzzy Logic Systems*, 2013, pp. 45–52.

[12] R. Babuska, "Fuzzy Modelling and Identification," *Control Engineering Laboratory, Faculty of Information Technology and Systems, Delft University of Technology, Delft, The Netherlands, version*, vol. 3, 2000.

[13] J. R. Castro, O. Castillo, and P. Melin, "An Interval Type-2 Fuzzy Logic Toolbox for Control Applications," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2007, pp. 1–6.

[14] The Mathworks, "Fuzzy Logic Toolbox - Design and simulate fuzzy logic systems," https://www.mathworks.com/products/fuzzy-logic.html, 2019.

[15] J. M. Alonso and L. Magdalena, "Generating Understandable and Accurate Fuzzy Rule-Based Systems in a Java Environment," in *Fuzzy Logic and Applications*, 2011, pp. 212–219.

[16] S. Guillaume and B. Charnomordic, "Fuzzy Inference Systems: an integrated modelling environment for collaboration between expert knowledge and data using FisPro," *Expert Systems with Applications*, vol. 39, no. 10, pp. 8744–8755, aug 2012.

[17] C. Wagner, M. Pierfitt, and J. McCulloch, "Juzzy online: An online toolkit for the design, implementation, execution and sharing of Type-1 and Type-2 fuzzy logic systems," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2014, pp. 2321–2328.

[18] C. Chen, T. R. Razak, and J. M. Garibaldi, "FuzzyR: An Extended Fuzzy Logic Toolbox for the R Programming Language," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2020, pp. 1–8.

[19] C. Wagner, S. Miller, and J. M. Garibaldi, "A fuzzy toolbox for the R programming language," *Proceedings IEEE International Conference on Fuzzy Systems*, pp. 1185–1192, 2011.

[20] C. Chen, J. M. Garibaldi, and T. Razak, "FuzzyR: Fuzzy Logic Toolkit for R," https://CRAN.R-project.org/package=FuzzyR, 2019.

[21] C. Chen, R. John, J. Twycross, and J. M. Garibaldi, "An extended ANFIS architecture and its learning properties for type-1 and interval type-2 models," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2016, pp. 602–609.

[22] C. Chen, J. Twycross, and J. M. Garibaldi, "A new accuracy measure based on bounded relative error for time series forecasting," *PLOS ONE*, vol. 12, no. 3, pp. 1–23, 2017.

[23] J. M. Mendel, *Uncertain Rule-Based Fuzzy Systems: Introduction and New Directions, 2nd Edition*. Springer International Publishing, 2017.

[24] P. Jaccard, "Nouvelles Recherches Sur la Distribution Florale," *Bulletin de la Societe Vaudoise des Sciences Naturelles*, vol. 44, pp. 223–270, 1908.

[25] D. Pekaslan, J. M. Garibaldi, and C. Wagner, "Exploring Subsethood to Determine Firing Strength in Non-Singleton Fuzzy Logic Systems," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2018, pp. 1–8.

[26] L. R. Dice, "Measures of the Amount of Ecologic Association Between Species," *Ecology*, vol. 26, no. 3, pp. 297–302, 1945.

[27] T. J. Sørensen, "A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons." *Kongelige Danske Videnskabernes Selskab*, vol. 5, no. 4, pp. 1–34, 1948.

[28] A. Pourabdollah, C. Wagner, and J. Aladi, "Changes under the hood - a new type of non-singleton fuzzy logic system," in *Proceedings IEEE International Conference on Fuzzy Systems*, 2015, pp. 1–8.

[29] L. X. Wang and J. M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1414–1427, 1992.

[30] G. Acampora, *Fuzzy Markup Language: A XML Based Language for Enabling Full Interoperability in Fuzzy Systems Design*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 17–31.