

## Research Article

# Simpful: A User-Friendly Python Library for Fuzzy Logic

Simone Spolaor<sup>1, </sup>, Caro Fuchs<sup>2, </sup>, Paolo Cazzaniga<sup>3,4,5, </sup>, Uzey Kaymak<sup>2, </sup>, Daniela Besozzi<sup>1,4,5, </sup>, Marco S. Nobile<sup>2,4,5,\* </sup>

<sup>1</sup>Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milan, Italy

<sup>2</sup>School of Industrial Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands

<sup>3</sup>Department of Human and Social Sciences, University of Bergamo, Bergamo, Italy

<sup>4</sup>SYSBIO/ISBE.IT Centre of Systems Biology, Milan, Italy

<sup>5</sup>Bicocca Bioinformatics, Biostatistics and Bioimaging Centre (B4), Milan, Italy

## ARTICLE INFO

### Article History

Received 10 Jun 2020

Accepted 07 Oct 2020

### Keywords

Decision support

Fuzzy logic

Fuzzy networks

Modeling and control

Open- source software

Python library

## ABSTRACT

Many researchers have used fuzzy set theory and fuzzy logic in a variety of applications related to computer science and engineering, given the capability of fuzzy inference systems to deal with uncertainty, represent vague concepts, and connect human language to numerical data. In this work we propose *Simpful*, a general-purpose and user-friendly Python library designed to facilitate the definition, analysis, and interpretation of fuzzy inference systems. *Simpful* provides a lightweight Application Programming Interface that allows to intuitively define fuzzy sets and fuzzy rules, and to perform fuzzy inference. Worthy of note, in *Simpful* the fuzzy rules are specified by means of strings of text written in natural language. We provide here some practical examples to show that *Simpful* represents a valuable addition to the open-source software that supports fuzzy reasoning.

© 2020 The Authors. Published by Atlantis Press B.V.

This is an open access article distributed under the CC BY-NC 4.0 license (<http://creativecommons.org/licenses/by-nc/4.0/>).

## 1. INTRODUCTION

Fuzzy set theory and fuzzy logic [1,2] are extensions of classic set theory and logic, which have been largely used in computer science and engineering. The ability of fuzzy inference systems (FISs) [3] to deal with uncertainty, represent vague concepts, and connect human language to numerical data, allowed fuzzy logic to be successfully exploited in different contexts [3,4], and in knowledge- or data-driven applications [5], as in the case of decision-making [6], modeling and control [7,8], classification, and regression problems [9–11].

The success of fuzzy reasoning led to the development of several methods and software tools involving fuzzy sets or FISs, usually aimed at specific applications [12]. However, general-purpose software libraries and toolboxes capable of handling fuzzy sets and/or fuzzy logic are limited in number and scope, and they are often outdated or not open-source. Reasons for this shortcoming might be the difficulty in dealing with the complex objects required by fuzzy reasoning (i.e., fuzzy sets, fuzzy rules, and natural language), and the high number of existing types of FISs [13].

To overcome these limitations, here we propose *Simpful*, a user-friendly Python library designed to define FISs for any purpose. *Simpful* provides a lightweight Application Programming Interface (API) for fuzzy reasoning, including a set of classes and methods to intuitively define fuzzy sets and fuzzy rules, and to perform fuzzy

inference. A noticeable feature of *Simpful* is that fuzzy rules can be constructed by means of strings of text written in natural language, thus simplifying the definition of fuzzy rule bases.

To show the usefulness and the advantages of *Simpful*, we provide three practical applications, related to the definition of a Mamdani and Takagi-Sugeno FIS for the tipping problem, a FIS for a clinical decision support system for septic patients, and a dynamic fuzzy model (DFM) of a biochemical system. The first two examples illustrate how *Simpful* can be used to easily define the membership functions and a fuzzy rule base, and how it embeds the execution of fuzzy inference. The third example shows how *Simpful* can be exploited to model and simulate the dynamics of complex systems [25,26], by creating fuzzy networks (FNs) [27], i.e., networks where nodes represent linguistic variables, and the connections between them represent interactions in the form of fuzzy rule outputs fed as variable inputs to a downstream linguistic variable. These networks can be defined with arbitrary topologies, including cycles and feedback loops, to describe the interactions existing in complex systems [27–30].

The paper is structured as follows: In Section 2 we provide a survey of the available software for the design of FISs, while in Section 3 we describe the implementation details of *Simpful*. In Section 4 we describe the three examples of application of *Simpful*. Finally, in Section 5 we draw some final remarks and provide insights on future developments of the library.

\* Corresponding author. Email: [m.s.nobile@tue.nl](mailto:m.s.nobile@tue.nl)

## 2. RELATED WORK

A detailed overview of the software tools that deal with fuzzy logic and fuzzy reasoning can be found in Ref. [12]. In this section, we restrict our discussion to general-purpose software (see Table 1).

Matlab is one of the most popular environments used to implement fuzzy logic tools [19]. More recently, Mathworks has introduced a Fuzzy Logic Toolbox [20] that is still supported by the company, and offers a variety of functions to manage many systems involving fuzzy logic. Additional extensions and new software were also presented in the past (e.g., in Ref. [31]) to address the needs of different fuzzy logic communities. However, Matlab has the drawback of being commercially distributed only, thus open-source alternatives were developed by the scientific community. At present, most of these open-source software are designed for specific applications only, whereas others are often outdated or not continually maintained.

PyFuzzy [21] was the first general-purpose library to design FISs using the Python programming language (version 2.7). It is dependent on the ANTLR 3 runtime. PyFuzzy allows to manage all the entities needed to construct FISs and to create numerous types of fuzzy sets. It also supports the export and sharing of FISs by means of the Fuzzy Control Language (FCL) files. FCL files implement the old standard IEC 61131 (IEC61131-7) [32], which was designed for fuzzy control applications and remained for many years the only *de facto* standard to represent FISs. Despite its completeness, PyFuzzy is now outdated and not maintained anymore, and Python 2 is no longer officially supported.

To date, the software developed using Python 3 includes Scikit-Fuzzy [23] and Fuzzylab [22]. Scikit-Fuzzy is a fuzzy logic API meant to work in the scipy stack [33], which offers functions and classes to support the modeling of fuzzy systems. Fuzzylab is a recently published Python 3 library, based on the Octave Fuzzy Logic Toolkit, designed for the creation of logic controllers. Both these libraries do not support the definition of custom membership

functions (providing only pre-implemented shapes), nor Takagi-Sugeno inference systems of arbitrary order. At present, Scikit-Fuzzy supports only Mamdani inference, while Fuzzylab supports Mamdani and 0-order Takagi-Sugeno inference systems. Moreover, both libraries do not provide an interface close to natural language for the definition of FISs. For example, neither Fuzzylab nor Scikit-Fuzzy allow to define fuzzy rules as strings of text written in natural language. Fuzzylab employs a matrix that needs to be input by the user to define the full rule base. In its main API, Scikit-Fuzzy adopts the prefix notation (i.e., operators precede the operands) to define the antecedents of rules. A later developed API (designed to implement control systems) is also available, but it requires to manipulate linguistic variable objects and to index them by means of linguistic terms in order to define a fuzzy rule.

Most of the recent open-source software for fuzzy logic are aimed at machine learning, classification and regression analysis, or decision support systems. Among the most popular, one can find FuzzyR [18], a general-purpose toolkit for fuzzy reasoning, implemented in R and supporting type-1 and type-2 FISs; FuzzyLite [14], a collection of C++ libraries designed for fuzzy control; Fispro [15], a C++ software provided with a graphical user interface (GUI), designed for data-driven applications of FISs and their automatic learning from a dataset; Juzzy [16], a Java based toolkit for handling type-2 fuzzy sets, and JT2FIS [34], a Java class library for the definition of interval type-2 FISs. One of the most recent and interesting implementations in the fuzzy community is represented by the JFML library [17], the only open-source library (implemented in Java) incorporating the most recently developed standard for representing FISs, the IEEE 1855-2016 standard [35], which defines a new W3C eXtensible Markup Language named Fuzzy Markup Language (FML) [36]. Notably, a Python wrapper for JFML was also released [24], allowing the definition of FML-compliant FISs by means of Python scripts and of the JFML library. However, it should be noted that this solution requires the Py4J framework [37], which is needed for the Python interpreter to dynamically access Java objects in a Java Virtual Machine.

**Table 1** | Software for the design of FISs.

Name	Language	Latest Release	Description
FuzzyLite [14]	C++	2017	A collection of C++ libraries designed for fuzzy control, compatible with the FCL standard
FisPro [15]	C++	2019	A general-purpose software provided with a GUI, designed to facilitate the learning of fuzzy inference systems from data
Juzzy [16]	Java	2013	A Java based toolkit, implementing type-2 fuzzy reasoning
JFML [17]	Java	2018	A Java library implementing the FML standard
FuzzyR [18]	R	2019	An R toolkit, provided with a GUI, for the design of type-1 and type-2 fuzzy inference systems
Fuzzy Toolbox for Matlab [19]	Matlab	1994	General-purpose toolbox implemented in the Matlab environment
Fuzzy Logic Toolbox [20]	Matlab	2020	Commercially distributed toolbox, provided with a GUI and available inside the Matlab environment
PyFuzzy [21]	Python 2	2014	A Python 2 library, compatible with the FCL standard. The development of the library was discontinued and Python 2 is no longer officially supported. This library depends on the ANTLR 3 runtime
Fuzzylab [22]	Python 3	2019	Python library based on the Octave Fuzzy Logic Toolkit
Scikit-Fuzzy [23]	Python 3	2019	General-purpose API meant to work in the scipy stack, offering classes and methods to support the definition of fuzzy systems
Py4JFML [24]	Python 3	2019	A Python wrapper for the JFML java library

Note: FIS, fuzzy inference system; FCL, Fuzzy Control Language; GUI, graphical user interface; FML, Fuzzy Markup Language; API, Application Programming Interface.

**Table 2** | Features supported by the available software for the design of FISs.

Name	Mamdani	Zero-order Takagi-Sugeno	First-order Takagi-Sugeno	Higher Order Takagi-Sugeno	Open Source
FuzzyLite [14]	✓	✓	✓	✓	✓
FisPro [15]	✓	✓			✓
Juzzy [16]	✓				✓
JFML [17]	✓	✓	✓		✓
FuzzyR [18]	✓				✓
Fuzzy Toolbox for Matlab [19]	✓	✓	✓		
Fuzzy Logic Toolbox [20]	✓	✓	✓		
PyFuzzy [21]	✓	✓			✓
Fuzzylab [22]	✓	✓			✓
Scikit-Fuzzy [23]	✓				✓
Py4JFML [24]	✓	✓	✓		✓
Simpful	✓	✓	✓	✓	✓

Note: FIS, fuzzy inference system; FML, Fuzzy Markup Language.

An overview of the features supported by the abovementioned software for the design of FISs can be found in Table 2. Excluding the outdated and discontinued PyFuzzy, a general-purpose, open-source, and intuitive Python library is nowadays still missing, prompting the development of Simpful to overcome the limitations of the existing software.

### 3. SOFTWARE DESCRIPTION

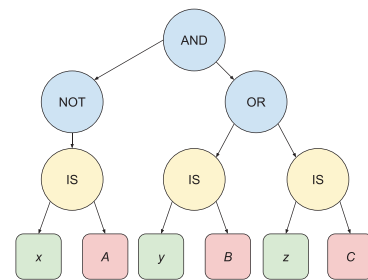
Simpful is implemented in the Python 3 programming language [38]. Its dependencies are numpy [39] and scipy [33]. The latest version of Simpful currently supports the following features:

- Definition of polygonal (e.g., vertex-based) and functional (e.g., sigmoidal, Gaussian, custom shaped) membership functions.
- Definition of fuzzy rules as strings of text written in natural language.
- Definition of arbitrarily complex fuzzy rules built with the logic operators AND, OR, NOT.
- Mamdani [40] and any order Takagi-Sugeno [41] inference methods.

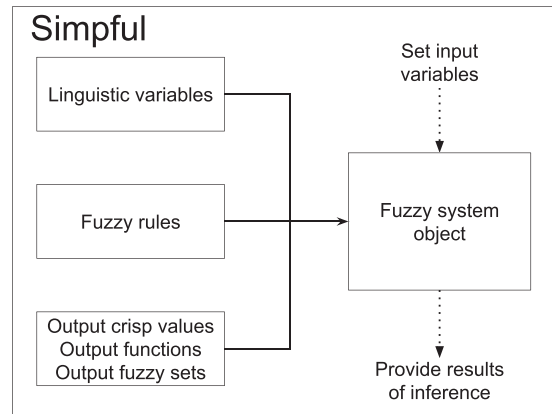
Simpful takes as input a human-readable representation of a FIS, consisting of a collection of fuzzy sets defined by membership functions, linguistic variables, fuzzy rules, and consequent outputs (specified as crisp values, arbitrary functions, or fuzzy sets). When this information is fed to Simpful’s FuzzySystem object, the system automatically performs a recursive tokening and parsing of the antecedents of each rule (exploiting the parentheses as delimiters), in order to identify atomic clauses and functional operators (i.e., the logical connectors AND, OR, NOT). By using these components, Simpful builds executable representations of the antecedents of rules in the form of derivation trees (Figure 1): the nodes represent the functional operators (blue nodes), while the leaves denote linguistic variables and terms (green and red nodes, respectively). By providing the input values for the antecedents, Simpful can perform a fuzzy inference and eventually provides the final output values (Figure 2).

A fuzzy set can be defined by using the FuzzySet object either as an ordered list of points in a plane, or as an arbitrary function:

“IF ((NOT (x IS A)) AND ((y IS B) OR (z IS C))) THEN ...”



**Figure 1** | Example of derivation tree produced by Simpful while parsing a fuzzy rule.



**Figure 2** | Graphical representation of the FuzzySystem object in Simpful.

- In the first case, the points are passed to the constructor using the argument points. For each point, the first coordinate corresponds to its value in the universe of discourse  $U$ , while the second represents the degree of membership. This sequence of points ultimately identifies a polygon. Simpful joins each pair of consecutive points in the sequence to identify the membership function characterizing the fuzzy set. Simpful deals with input values that are outside the specified universe of discourse by extrapolating, using the closest point specified in the sequence. This also means Simpful supports the use of shouldered fuzzy sets.

- In the second case, the user creates a FuzzySet object by passing a function pointer, using the argument function. This custom function should be in the form  $f: \mathbb{U} \rightarrow [0, 1]$ ,  $\mathbb{U} \subseteq \mathbb{R}$ , mapping every element of the universe of discourse to a valid membership value. In case the codomain of the custom function  $f$  provided by the user is not equal to the expected interval, Simplful automatically clamps it to  $[0, 1]$ . Simplful also provides a set of objects with preimplemented general-purpose parametric functions, namely: Gaussian (Gaussian\_MF), inverted Gaussian (InvGaussian\_MF), double Gaussian (DoubleGaussian\_MF), sigmoid (Sigmoid\_MF), inverted sigmoid (InvSigmoid\_MF), triangular (Triangular\_MF) and trapezoidal (Trapezoidal\_MF). It is worth noting that all these objects are derived from the same abstract class, named MF\_object: by providing an implementation of the virtual method \_execute(), users can straightforwardly create arbitrarily complex fuzzy sets exploiting any custom function. In this case, the new method must accept an argument  $x \in \mathbb{U}$  and must return a valid membership value.

The user is also required to associate a meaningful linguistic term with each fuzzy set. The defined fuzzy sets are then employed in the creation of specific LinguisticVariable objects, provided with their own names, as given by the user. If the fuzzy sets are specified by means of a sequence of points only, Simplful automatically identifies the boundaries of the universe of discourse by exploiting the minimum and the maximum value among the first coordinates of all the points that define all the fuzzy sets. On the contrary, if a fuzzy set is defined by means of custom functions, the user can specify a valid interval of values for the LinguisticVariable object, by using the optional argument universe\_of\_discourse. The definition of this interval is required in order to use the draw() method of the LinguisticVariable class, which can be exploited to plot the fuzzy sets, thus allowing for a rapid inspection and debugging of their implementation. Worthy of note, both point- and function-based fuzzy sets can be exploited simultaneously in the definition of a single linguistic variable. In order to facilitate its usage, Simplful also provides a AutoTriangle() class, which returns a LinguisticVariable object whose universe of discourse is subdivided in a user-defined number of normalized triangular fuzzy sets.

The fuzzy rules used for the inference must be defined by means of well-formed strings, written in natural language (Figure 1). The current version of Simplful supports the most common fuzzy operators AND, OR, NOT, defined as

- NOT  $x = 1 - x$ .
- $x$  OR  $y = \max(x, y)$ .
- $x$  AND  $y = \min(x, y)$ .

Each rule must use the variables' names and linguistic terms that were defined in the LinguisticVariable objects.

In the case of Takagi–Sugeno systems, the consequent of rules must also use strings that are associated with the output crisp values or with the output functions defined by the user. In this case, the user has to define the functions exploited in the inference as follows:

- 0-order functions (i.e., constant functions) are defined as “output crisp values.”
- For higher order Takagi–Sugeno systems, the user can define “output functions” as strings of text involving the linguistic variables. These functions are evaluated at runtime by exploiting the eval() function of Python, which parses the expression given as a string argument and executes it as a code within the program.

Output crisp values, output functions, and output fuzzy sets must have an associated meaningful and unique string to identify them, which will be exploited in the definition of the fuzzy rules.

Linguistic variables, fuzzy rules, output crisp values, output functions, and output fuzzy sets are added to a fuzzy system object, which implements the whole FIS. Given the input values for the variables appearing in the antecedents of the fuzzy rules, the methods implementing the Mamdani or Takagi–Sugeno inference can be called for one or more of the variables appearing in the consequent of fuzzy rules, in order to obtain their final output. As an alternative, the user can invoke the inference() method of the FuzzySystem class to let Simplful choose and use the most appropriate inference method (i.e., Mamdani or Takagi–Sugeno). As a matter of fact, during the initialization phase, Simplful analyzes the outputs of the model and automatically determines the class of the defined FIS. The results of the inference are returned to the user as key-value pairs inside a dictionary, where keys represent the names of the variables. Figure 2 shows a schematic overview of how to construct a fuzzy system object and how to perform inference in Simplful.

The source code of Simplful is available, under GPL license, on GitHub at the following URL: <https://github.com/aresio/simpful>. Simplful can be installed by using the PyPI facility: `pip install simpful`. The example code described in this work can be found on Code Ocean at the following URL: <https://codeocean.com/capsule/2230971/tree>.

## 4. ILLUSTRATIVE EXAMPLES

In this section we provide three examples, together with their corresponding Python code, to show the potential and the usage of Simplful.

### 4.1. Tipping Problem

The tipping problem consists in computing a fair tip (in terms of percentage of the overall bill), taking into account a restaurant's services. Listings 1 and 2 show two examples of Simplful code to define a FIS that calculates the tipping amount on the basis of two input variables, describing food and serving staff quality.

In Listing 1 the tipping problem is modeled as a Mamdani FIS. In line 5 a fuzzy system object is created. The fuzzy sets and the linguistic variable “Service” are defined in lines 8 to 11; this variable contains three fuzzy sets, “poor,” “good,” and “excellent,” ranging from 0 to 10. From line 13 to 15 the linguistic variable for food quality is defined, exploiting two fuzzy sets, “rancid” and “delicious.” The output variable “Tip” and its fuzzy sets are defined from line 18 to 21. All fuzzy sets used in this example are triangular (hence the use of

**Listing 1** | A Mamdani FIS for the tipping problem, defined in Simpful.

```

1  from simpful import *
2
3  # A simple fuzzy inference system for the tipping problem
4  # Create a fuzzy system object
5  FS = FuzzySystem()
6
7  # Define fuzzy sets and linguistic variables
8  S_1 = FuzzySet(function=Triangular_MF(a=0, b=0, c=5), term="poor")
9  S_2 = FuzzySet(function=Triangular_MF(a=0, b=5, c=10), term="good")
10 S_3 = FuzzySet(function=Triangular_MF(a=5, b=10, c=10), term="excellent")
11 FS.add_linguistic_variable("Service", LinguisticVariable([S_1, S_2, S_3], concept="Service quality", universe_of_discourse=[0,10]))
12
13 F_1 = FuzzySet(function=Triangular_MF(a=0, b=0, c=10), term="rancid")
14 F_2 = FuzzySet(function=Triangular_MF(a=0, b=10, c=10), term="delicious")
15 FS.add_linguistic_variable("Food", LinguisticVariable([F_1, F_2], concept="Food quality", universe_of_discourse=[0,10]))
16
17 # Define output fuzzy sets and linguistic variable
18 T_1 = FuzzySet(function=Triangular_MF(a=0, b=0, c=10), term="small")
19 T_2 = FuzzySet(function=Triangular_MF(a=0, b=10, c=20), term="average")
20 T_3 = FuzzySet(function=Trapezoidal_MF(a=10, b=20, c=25, d=25), term="generous")
21 FS.add_linguistic_variable("Tip", LinguisticVariable([T_1, T_2, T_3], universe_of_discourse=[0,25]))
22
23 # Define fuzzy rules
24 R1 = "IF (Service IS poor) OR (Food IS rancid) THEN (Tip IS small)"
25 R2 = "IF (Service IS good) THEN (Tip IS average)"
26 R3 = "IF (Service IS excellent) OR (Food IS delicious) THEN (Tip IS generous)"
27 FS.add_rules([R1, R2, R3])
28
29 # Set antecedents values
30 FS.set_variable("Service", 4)
31 FS.set_variable("Food", 8)
32
33 # Perform Mamdani inference and print output
34 print(FS.Mamdani_inference(["Tip"]))

```

the preimplemented function `Triangular_MF`), except for the fuzzy set that denotes a “Generous” tip, which is an example of a trapezoidal set (`Trapezoidal_MF`). The resulting membership functions are visualized in Figure 3. The fuzzy rules are defined in lines 24 to 27. Once the input values are set (lines 30 and 31, where “Service” and “Food” quality scored 4 and 8 points, respectively), Mamdani fuzzy inference is performed (line 34) to obtain the final tipping percentage, which is equal to 14.17% in this example.

Listing 2 shows the definition of another FIS to solve the tipping problem, this time using a Takagi–Sugeno model. Again, the fuzzy system object is created in line 5. From line 8 to 15 the fuzzy sets for the input variables are defined. In this example, the fuzzy sets are defined as polygons, using an ordered list of points, instead of exploiting a pre-implemented membership function as in the previous example. However, the resulting fuzzy sets are the same as the sets used in the Mamdani FIS. The output crisp values for a “small” and “average” tip are set to 5% and 15% respectively in line 18 and 19. The output value for a “generous” tip is a function (defined in line 22) depending on the scores for service and food quality. In this example this is a linear function, but any arbitrary function can be handled by Simpful. The fuzzy rules are then defined in lines 25 to 28. The input values are set in line 31 and 32, and Takagi–Sugeno inference is performed in line 35. Given that “Service” and

“Food” quality scored 4 and 8 points, the tipping percentage should be 14.77% according to this example.

Finally, Figure 4 shows a comparison of the output surfaces produced by `simpful` and `Scikit-Fuzzy`. The response of the two libraries is identical, confirming the correctness of `Simpful`’s fuzzy inference.

## 4.2. Clinical Decision Support for Sepsis

When a patient enters the intensive care unit (ICU) with symptoms of sepsis, clinicians must diagnose quickly and start treatment within an hour of admission. Clinical decision support systems aim at helping clinicians with these decisions by processing patient data, such as blood levels and symptoms, and giving suggestions for diagnosis or treatment plans.

In Listing 3, we provide a simplified clinical decision support model that calculates how likely it is the patient suffers from sepsis. First, the fuzzy system object is created in line 5. The fuzzy sets and their linguistic terms for the input variables are specified in lines 8 to 31. In this example, the fuzzy sets are defined by using sigmoidal and Gaussian functions. Since these functions have  $\mathbb{U} = \mathbb{R}$  as domain, it is not possible to automatically estimate the limits of the universe of discourse for plotting the membership functions with the `draw()`

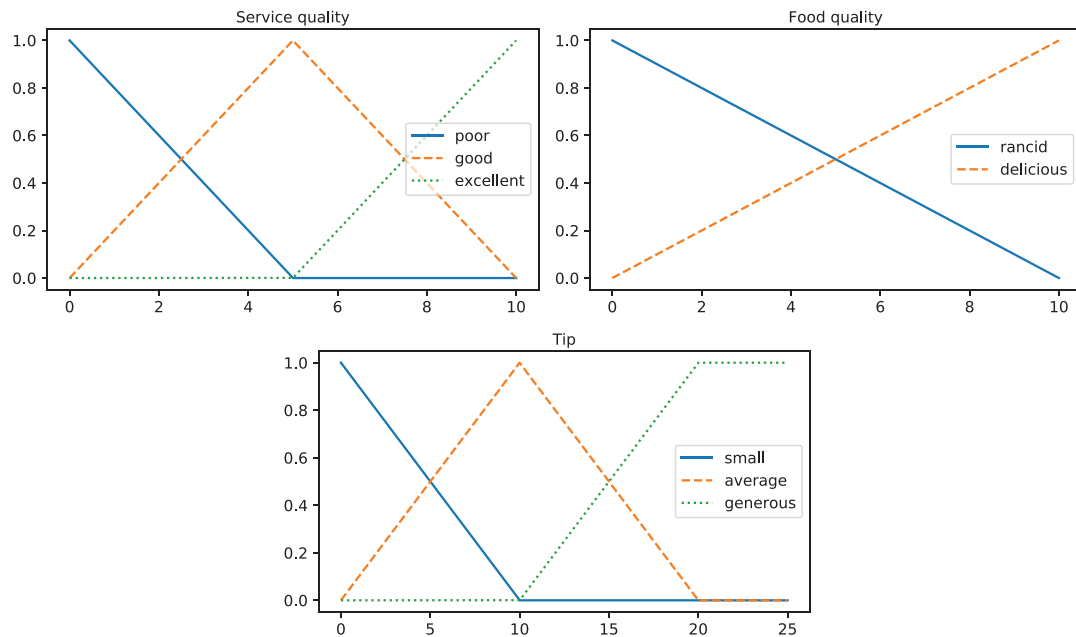


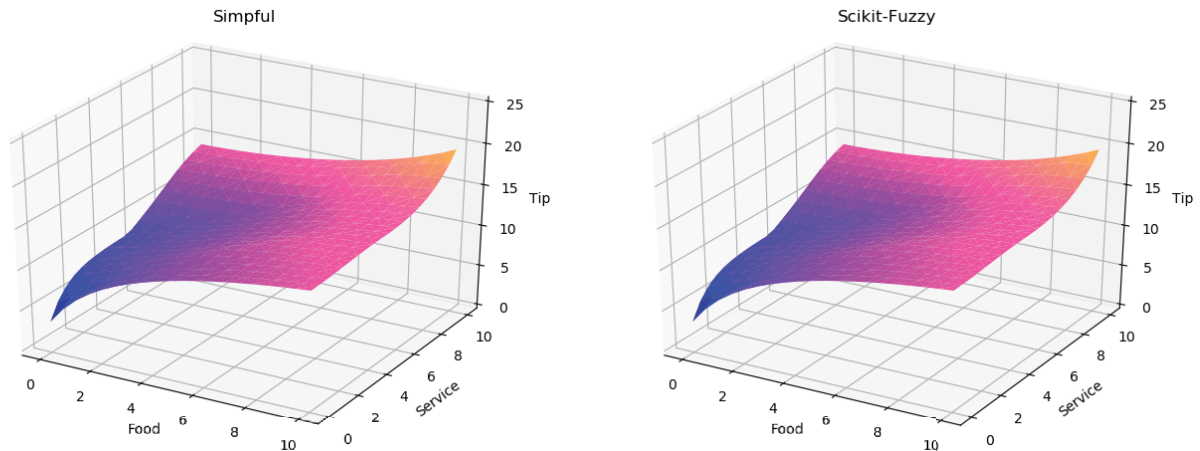
Figure 3 | The membership functions for the tipping problem.

Listing 2 | A Takagi-Sugeno FIS for the tipping problem, defined in Simpsful.

```

1  from simpful import *
2
3  # A simple fuzzy inference system for the tipping problem
4  # Create a fuzzy system object
5  FS = FuzzySystem()
6
7  # Define fuzzy sets and linguistic variables
8  S_1 = FuzzySet(points=[[0., 1.], [5., 0.]], term="poor")
9  S_2 = FuzzySet(points=[[0., 0.], [5., 1.], [10., 0.]], term="good")
10 S_3 = FuzzySet(points=[[5., 0.], [10., 1.]], term="excellent")
11 FS.add_linguistic_variable("Service", LinguisticVariable([S_1, S_2, S_3], concept="Service quality"))
12
13 F_1 = FuzzySet(points=[[0., 1.], [10., 0.]], term="rancid")
14 F_2 = FuzzySet(points=[[0., 0.], [10., 1.]], term="delicious")
15 FS.add_linguistic_variable("Food", LinguisticVariable([F_1, F_2], concept="Food quality"))
16
17 # Define output crisp values
18 FS.set_crisp_output_value("small", 5)
19 FS.set_crisp_output_value("average", 15)
20
21 # Define function for generous tip (food score + service score + 5%)
22 FS.set_output_function("generous", "Food+Service+5")
23
24 # Define fuzzy rules
25 R1 = "IF (Service IS poor) OR (Food IS rancid) THEN (Tip IS small)"
26 R2 = "IF (Service IS good) THEN (Tip IS average)"
27 R3 = "IF (Service IS excellent) OR (Food IS delicious) THEN (Tip IS generous)"
28 FS.add_rules([R1, R2, R3])
29
30 # Set antecedents values
31 FS.set_variable("Service", 4)
32 FS.set_variable("Food", 8)
33
34 # Perform Sugeno inference and print output
35 print(FS.Sugeno_inference(["Tip"]))

```



**Figure 4** | Comparison of output surfaces obtained on the Mamdani fuzzy inference system (FIS) for the tipping problem, implemented in Simpful (left) and Scikit-Fuzzy (right): the output of the two libraries is the same.

**Listing 3** | A clinical decision support system to diagnose sepsis, implemented in Simpful.

```

1  from simpful import *
2
3  # A simple decision support model to diagnose sepsis in the ICU
4  # Create a fuzzy system object
5  FS = FuzzySystem()
6
7  # Define fuzzy sets for the variable PaO2
8  P1 = FuzzySet(function=Sigmoid_MF(c=40, a=0.1), term="low")
9  P2 = FuzzySet(function=InvSigmoid_MF(c=40, a=0.1), term="high")
10 LV1 = LinguisticVariable([P1,P2], concept="PaO2 level in blood", universe_of_discourse=[0,80])
11 FS.add_linguistic_variable("PaO2", LV1)
12 LV1.plot()
13
14 # Define fuzzy sets for the variable base excess
15 B1 = FuzzySet(function=Gaussian_MF(mu=0, sigma=1.25), term="normal")
16 LV2 = LinguisticVariable([B1], concept="Base excess of the blood", universe_of_discourse=[-10,10])
17 FS.add_linguistic_variable("BaseExcess", LV2)
18 LV2.plot()
19
20 # Define fuzzy sets for the variable trombocytes
21 T1 = FuzzySet(function=Sigmoid_MF(c=50, a=0.75), term="low")
22 T2 = FuzzySet(function=InvSigmoid_MF(c=50, a=0.75), term="high")
23 LV3 = LinguisticVariable([T1,T2], concept="Trombocytes in blood", universe_of_discourse=[0,100])
24 FS.add_linguistic_variable("Trombocytes", LV3)
25 LV3.plot()
26
27 # Define fuzzy sets for the variable creatinine
28 C1 = FuzzySet(function=Sigmoid_MF(c=300, a=0.2), term="low")
29 C2 = FuzzySet(function=InvSigmoid_MF(c=300, a=0.1), term="high")
30 LV4 = LinguisticVariable([C1,C2], concept="Creatinine in blood", universe_of_discourse=[0,600])
31 FS.add_linguistic_variable("Creatinine", LV4)
32 LV4.plot()
33
34 # Define the consequents
35 FS.set_crisp_output_value("low_probability", 1)
36 FS.set_crisp_output_value("high_probability", 99)
37
38 # Define the fuzzy rules
39 RULE1 = "IF (PaO2 IS low) AND (Trombocytes IS high) AND (Creatinine IS high) AND (BaseExcess IS normal) THEN (Sepsis IS low_probability)"

```

**Listing 3** A clinical decision support system to diagnose sepsis, implemented in Simpful.

```

40 RULE2 = "IF (PaO2 IS high) AND (Trombocytes IS low) AND (Creatinine IS low) AND (NOT(BaseExcess IS normal)) THEN (Sepsis IS
41 high_probability)"
42 # Add fuzzy rules to the fuzzy reasoner object
43 FS.add_rules([RULE1, RULE2])
44
45 # Set antecedent values
46 FS.set_variable("PaO2", 50)
47 FS.set_variable("BaseExcess", -1.5)
48 FS.set_variable("Trombocytes", 50)
49 FS.set_variable("Creatinine", 320)
50
51 # Perform Sugeno inference and print output
52 print(FS.Sugeno_inference(["Sepsis"]))

```

method. Therefore, the user has to explicitly set the universe of discourse by specifying the argument `universe_of_discourse` (lines 10, 16, 23 and 30). Please note that this (explicitly limited) universe of discourse is only used to plot the membership functions: the system still supports input values outside this range when inferring the output. In line 12, the plots for the membership functions of the first variable are generated (Figure 5).

In lines 15 to 17, the variable “base excess” is created by first specifying the fuzzy set describing physiological values (using the preimplemented `Gaussian_MF` function). This is the only fuzzy set for this linguistic variable, meaning that the fuzzy set for the abnormal values for the base excess is not explicitly modeled, but corresponds to the complement of the fuzzy set for normal values (note the `NOT` operator in second fuzzy rule, line 40). The same results could be achieved by modeling the set of non-normal values explicitly, either using the `InvGaussian_MF` function, or splitting the universe of discourse in a low, medium, and high fuzzy set, and later connecting the low and high fuzzy set in the fuzzy rule through an `OR` operator. However, using the `NOT` operator simplifies the FIS and preserves its high levels of interpretability. The output crisp values “low\_probability” and “high\_probability,” referring to the probability that the patient is suffering from sepsis, are defined in lines 35 and 36. The fuzzy rules are then defined and added to the fuzzy system object in lines 39 to 43. Lines 46 to 49 provide example input values to the model, and fuzzy inference is performed in line 52 to obtain the probability that the patient is suffering from sepsis which, in this example, is equal to 69.3%.

### 4.3. Repressilator

The repressilator is a synthetic regulatory network consisting of three genes placed in a feedback loop, where the genetic product of each gene inhibits the expression of the next gene in the network (Figure 6). This simple system was designed to exhibit a stable oscillatory regime, studied by means of mechanistic modeling, and then implemented *in vivo* in the bacterium *E. coli* [42]. Here, we provide a simple redefinition of the repressilator in terms of a DFM, to show how Simpful can also be applied for the fuzzy modeling of complex systems.

DFM is a formalism useful to analyze the emergent behavior of complex systems characterized by uncertainty [25]. A DFM consists

of a set of linguistic variables describing the components of the system, and a set of fuzzy rules providing a qualitative description of their interactions. A DFM can be considered as a FN [27], i.e., a network of interacting FISs. Thus, a FN can be depicted as a directed graph (as in Figure 6), where nodes represent linguistic variables, and arcs the presence of some fuzzy rules governing them.

The example code of the repressilator is given in Listing 4. In line 6 a fuzzy system object is created. From line 9 to 12, the three linguistic variables related to the three species constituting the repressilator are defined. All three species are characterized by a universe of discourse ranging from 0 to 1 (the default universe of discourse), and by the presence of two fuzzy sets, “low” and “high,” representing the quantity of each protein. Note that here the `AutoTriangle()` class is used (line 9) in order to define a general linguistic variable characterized by 2 symmetrical fuzzy sets covering the whole universe of discourse. In this example, a value of 1 in the universe of discourse corresponds to the maximum quantity, while 0 to the absence of the protein. Analogously, the output crisp values are defined in lines 15 and 16, by setting “low” to 0 and “high” to 1. Lines 19 to 26 contain the definition of the fuzzy rules, representing the negative feedbacks existing between the three genes.

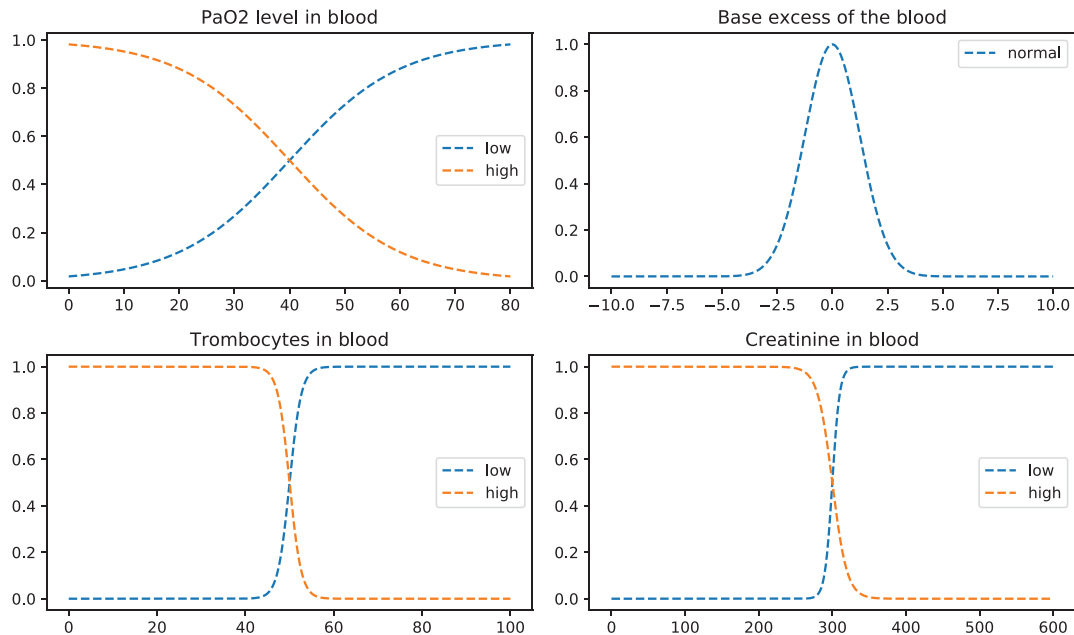
The initial state of the DFM is set in lines 29 to 31, the number of simulation steps is defined in line 34, while the data structure containing the results of the simulation is initialized in lines 35 and 36. Lines 39 to 42 contain the for loop in which the simulation is performed. In particular, the new state of the system is inferred in line 40, updated in line 41, and then stored in the previously defined data structure. Note that, in this example, we exploit the `inference()` method provided by the `FuzzySystem` object (see line 40).

The final output, i.e., the simulation of the system’s dynamics, can be plotted as shown in Figure 7. Despite its simplicity and the lack of a precise kinetic parameterization, this model can reproduce the typical oscillatory dynamics of the three species, as shown in the original model [42].

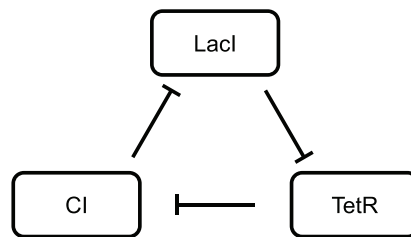
## 5. CONCLUSIONS

Simpful is a novel library that addresses the need of having a lightweight, open-source, Python API to support the creation of readable FISs, based on either Mamdani or Takagi–Sugeno fuzzy





**Figure 5** | The membership functions for the clinical decision support system to diagnose sepsis at the intensive care unit (ICU).



**Figure 6** | Graphical representation of the repressilator.

reasoning. Users can define fuzzy sets as polygons (formalized as sequences of vertices), parametric functions (e.g., Gaussian and sigmoid), or even arbitrary custom functions in the universe of discourse/degree of membership space. Fuzzy rules are encoded as strings of text written in natural language, making FISs created in *Simpful* easier to read and to inspect compared to competitor libraries. To show its usage, we provided three examples: the definition of a FIS for the tipping problem, the definition of a clinical decision support system to diagnose sepsis, and the modeling and simulation of a complex biochemical system by means of a DFM. Thanks to its features, *Simpful* is a valuable addition to the open-source software that support fuzzy reasoning, and it is expected to highly facilitate the definition, analysis and interpretation of FISs in a wide variety of data- and knowledge-driven applications.

*Simpful* was employed to implement the *FuzzX* framework for the modeling and simulation of hybrid (qualitative and quantitative) systems [26]. The porting of *FUMOSO* [25] to *Simpful*, currently in progress, will also promote the use of DFMs for the investigation of complex systems. *Simpful* is also employed within *pyFUME* [43], a novel Python package developed to estimate FISs

automatically from data [44–47]. Moreover, *Simpful* can be readily integrated in computational intelligence methods that use Mamdani or Takagi–Sugeno inference, such as the class of global optimization meta-heuristics exploiting fuzzy reasoning for dynamic parameter adaptation [48,49].

In future releases, we plan to extend *Simpful* with support for additional fuzzy logic operators and other fuzzy inference methods (e.g., Tsukamoto [50], and AnYa [51] methods). In particular, we will add support for weighted fuzzy rules [52], type-2 FISs [53], and probabilistic fuzzy reasoning [54], the latter providing a means to combine the interpretability of FIS with the statistical properties of probabilistic systems. Finally, *Simpful* will support the FML format defined in the IEEE Std 1855-2016 [35], possibly by leveraging existing software (i.e., *JFML* and *Py4JFML* [17,24]), to facilitate the import, export, and sharing of the FISs defined within this library.

## CONFLICTS OF INTEREST

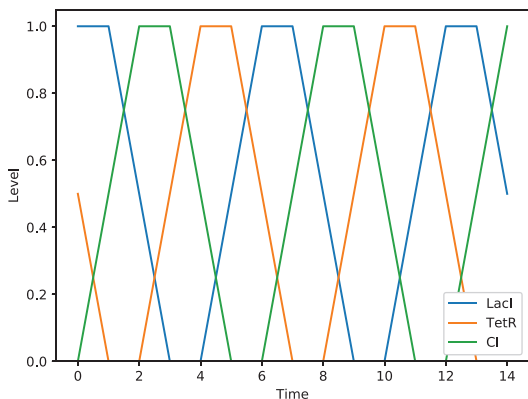
The authors declare no conflicts of interest.

**Listing 4** | A DFM of the repressilator model, implemented using Simpful.

```

1  from simpful import *
2  from copy import deepcopy
3
4  # A simple dynamic fuzzy model of the repressilator
5  # Create a fuzzy reasoner object
6  FS = FuzzySystem()
7
8  # Define fuzzy sets and linguistic variables
9  LV = AutoTriangle(2, terms=['low', 'high'])
10 FS.add_linguistic_variable("LacI", LV)
11 FS.add_linguistic_variable("TetR", LV)
12 FS.add_linguistic_variable("CI", LV)
13
14 # Define output crisp values
15 FS.set_crisp_output_value("low", 0.0)
16 FS.set_crisp_output_value("high", 1.0)
17
18 # Define fuzzy rules
19 RULES = []
20 RULES.append("IF (LacI IS low) THEN (TetR IS high)")
21 RULES.append("IF (LacI IS high) THEN (TetR IS low)")
22 RULES.append("IF (TetR IS low) THEN (CI IS high)")
23 RULES.append("IF (TetR IS high) THEN (CI IS low)")
24 RULES.append("IF (CI IS low) THEN (LacI IS high)")
25 RULES.append("IF (CI IS high) THEN (LacI IS low)")
26 FS.add_rules(RULES)
27
28 # Set antecedents values
29 FS.set_variable("LacI", 1.0)
30 FS.set_variable("TetR", 0.5)
31 FS.set_variable("CI", 0.0)
32
33 # Set simulation steps and save initial state
34 steps = 14
35 dynamics = []
36 dynamics.append(deepcopy(FS._variables))
37
38 # At each simulation step, perform Sugeno inference, update state and save the results
39 for i in range(steps):
40     new_values = FS.inference()
41     FS._variables.update(new_values)
42     dynamics.append(new_values)

```

**Figure 7** | Dynamics of the variables of the dynamic fuzzy model (DFM) representing the repressilator.

## AUTHORS' CONTRIBUTIONS

MSN conceived the idea of the library; SS and MSN designed and implemented the library; SS and CF conceived the usage examples, analyzed the results and performed comparisons with the other methods; SS, CF and MSN prepared and created the figures and wrote the first draft of the manuscript; PC, UK and DB critically reviewed and edited the manuscript; All authors read and approved its final version.

## Funding Statement

This work was partially funded by the SYSBIO/ISBE.IT Research Centre of Systems Biology.

## REFERENCES

- [1] L.A. Zadeh, Fuzzy sets, *Inf. Control.* 8 (1965), 338–353.
- [2] L.A. Zadeh, Computing with words, *IEEE Trans. Fuzzy Syst.* 4 (1996), 103–111.
- [3] J. Yen, R. Langari, *Fuzzy Logic: Intelligence, Control, and Information*, vol. 1, Prentice Hall, Upper Saddle River, 1999.
- [4] G.J. Klir, B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice Hall, Upper Saddle River, 1995.
- [5] E. Hüllermeier, From knowledge-based to data-driven fuzzy modeling, *Informatik-Spektrum.* 38 (2015), 500–509.
- [6] A. Mardani, A. Jusoh, E.K. Zavadskas, Fuzzy multiple criteria decision-making techniques and applications—two decades review from 1994 to 2014, *Expert Syst. Appl.* 42 (2015), 4126–4148.
- [7] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes, *IEEE Trans. Syst. Man Cybern.* 3 (1973), 28–44.
- [8] R. Babuška, H.B. Verbruggen, An overview of fuzzy modeling for control, *Control Eng. Pract.* 4 (1996), 1593–1606.
- [9] L.Y. Cai, H.K. Kwan, Fuzzy classifications using fuzzy inference networks, *IEEE Trans. Syst. Man Cybern. Part B.* 28 (1998), 334–347.
- [10] Y.-H.O. Chang, B.M. Ayyub, Fuzzy regression methods—a comparative assessment, *Fuzzy Sets Syst.* 119 (2001), 187–203.
- [11] E.H. Ruspini, Numerical methods for fuzzy clustering, *Inf. Sci.* 2 (1970), 319–350.
- [12] J. Alcalá-Fdez, J.M. Alonso, A survey of fuzzy systems software: taxonomy, current research trends, and prospects, *IEEE Trans. Fuzzy Syst.* 24 (2015), 40–56.
- [13] R.R. Yager, L.A. Zadeh, *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, vol. 165, Springer Science & Business Media, New York, NY, USA, 2012.
- [14] J. Rada-Vilela, *The FuzzyLite Libraries for Fuzzy Logic Control*, 2018. <https://www.fuzzylite.com/>
- [15] S. Guillaume, B. Charnomordic, Learning interpretable fuzzy inference systems with FisPro, *Inf. Sci.* 181 (2011), 4409–4427.
- [16] C. Wagner, Juzzy—a java based toolkit for type-2 fuzzy logic, in 2013 IEEE Symposium on Advances in Type-2 Fuzzy Logic Systems (T2FUZZ), IEEE, Singapore, 2013, pp. 45–52.
- [17] J.M. Soto-Hidalgo, J.M. Alonso, G. Acampora, J. Alcalá-Fdez, JFML: a java library to design fuzzy logic systems according to the IEEE std 1855-2016, *IEEE Access.* 6 (2018), 54952–54964.
- [18] C. Wagner, S. Miller, J.M. Garibaldi, A fuzzy toolbox for the R programming language, in 2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011), IEEE, Taipei, Taiwan, 2011, pp. 1185–1192.
- [19] R. Babuška, *Fuzzy Toolbox for MATLAB: Reference Guide*, Version 3.0, Technical Report, Delft University of Technology, Department of Electrical Engineering, Control Laboratory, Delft, Netherlands, 1994.
- [20] MathWorks, *Fuzzy Logic Toolbox - r2020a*, 2020. <https://www.mathworks.com/products/fuzzy-logic.html>
- [21] Pyfuzzy-python Fuzzy Package, 2014. <http://pyfuzzy.sourceforge.net/>
- [22] E. Avelar, O. Castillo, J. Soria, Fuzzy logic controller with fuzzy-lab python library and the robot operating system for autonomous robot navigation: a practical approach, in: O. Castillo, P. Melin, J. Kacprzyk (Eds.), *Intuitionistic and Type-2 Fuzzy Logic Enhancements in Neural and Optimization Algorithms: Theory and Applications*, Springer, Cham, Switzerland, 2020, pp. 355–369.
- [23] SciKit-Fuzzy, 2019. <https://pythonhosted.org/scikit-fuzzy/>
- [24] J. Alcalá-Fdez, J.M. Alonso, C. Castiello, C. Mencar, J.M. Soto-Hidalgo, Py4JFML: a Python wrapper for using the IEEE Std 1855-2016 through JFML, in 2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, New Orleans, LA, USA, 2019, pp. 1–6.
- [25] M.S. Nobile, G. Votta, R. Palorini, S. Spolaor, H. De Vitto, P. Cazzaniga, et al., Fuzzy modeling and global optimization to predict novel therapeutic targets in cancer cells, *Bioinformatics.* 36 (2019), 2181–2188.
- [26] S. Spolaor, M.S. Nobile, G. Mauri, P. Cazzaniga, D. Besozzi, Coupling mechanistic approaches and fuzzy logic to model and simulate complex systems, *IEEE Trans. Fuzzy Syst.* 28 (2020), 1748–1759.
- [27] A. Gegov, *Fuzzy Networks for Complex Systems*, Springer, Berlin, Heidelberg, Germany, 2010.
- [28] H. Kawamura, Fuzzy network for decision support systems, *Fuzzy Sets Syst.* 58 (1993), 59–72.
- [29] P.-T. Chang, E.S. Lee, Fuzzy decision networks and deconvolution, *Comput. Math. Appl.* 37 (1999), 53–63.
- [30] A.M. Yaakob, A. Serguieva, A. Gegov, FN-TOPSIS: fuzzy networks for ranking traded equities, *IEEE Trans. Fuzzy Syst.* 25 (2017), 315–332.
- [31] O. Castillo, P. Melin, J.R. Castro, Computational intelligence software for interval type-2 fuzzy logic, *Comput. Appl. Eng. Educ.* 21 (2013), 737–747.
- [32] International Electrotechnical Commission (IEC), Technical report, publisher IEC, IEC 61131-7, Programmable Controllers Part 7 - Fuzzy Control Programming, 2000.
- [33] E. Jones, T. Oliphant, P. Peterson, et al., *Scipy: Open Source Scientific Tools for Python*, 2001. <https://www.scipy.org/>
- [34] M. Castañón-Puga, J.R. Castro, M. Flores-Parra, Jt2fis: Java type-2 fuzzy inference system—an object-oriented class library for building java intelligent applications, in *International Conference on Enterprise Information Systems*, Angers, France, SCITEPRESS, 2013, vol. 2, pp. 524–529.
- [35] IEEE-SA Standards Board, *IEEE Standard for Fuzzy Markup Language*, IEEE Std 1855-2016, 2016.
- [36] G. Acampora, Fuzzy markup language: a XML based language for enabling full interoperability in fuzzy systems design, in: G. Acampora, V. Loia, C.S. Lee, M.H. Wang (Eds.), *On the Power of Fuzzy Markup Language*, Springer, Berlin, Heidelberg, Germany, 2013, pp. 17–31.
- [37] Py4J - a Bridge between Python and Java, 2018. <https://www.py4j.org/>
- [38] T.E. Oliphant, *Python for scientific computing*, *Comput. Sci. Eng.* 9 (2007), 10–20.
- [39] T.E. Oliphant, *A Guide to NumPy*, vol. 1, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.
- [40] E.H. Mamdani, S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, *Int. J. Man-Mach. Stud.* 7 (1975), 1–13.
- [41] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. Syst. Man Cybern.* 15 (1985), 116–132.

- [42] M.B. Elowitz, S. Leibler, A synthetic oscillatory network of transcriptional regulators, *Nature*. 403 (2000), 335.
- [43] C. Fuchs, S. Spolaor, M.S. Nobile, U. Kaymak, pyFUME: a Python package for fuzzy model estimation, in 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, Glasgow, UK, 2020, pp. 1–8.
- [44] C. Fuchs, A. Wilbik, U. Kaymak, Towards more specific estimation of membership functions for data-driven fuzzy inference systems, in 2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE, Rio de Janeiro, Brazil, 2018, pp. 1–8.
- [45] M. Setnes, R. Babuska, U. Kaymak, H.R. van Nauta Lemke, Similarity measures in fuzzy rule base simplification, *IEEE Trans. Syst. Man Cybern. Part B*. 28 (1998), 376–386.
- [46] U. Kaymak, R. Babuska, Compatible cluster merging for fuzzy modelling, in Proceedings of 1995 IEEE International Conference on Fuzzy Systems, IEEE, Yokohama, Japan, 1995, vol. 2, pp. 897–904.
- [47] C. Fuchs, S. Spolaor, M.S. Nobile, U. Kaymak, A graph theory approach to fuzzy rule base simplification, in: M.J. Lesot *et al.* (Eds.), International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, Springer, Cham, Switzerland, 2020, pp. 387–401.
- [48] M.S. Nobile, P. Cazzaniga, D. Besozzi, R. Colombo, G. Mauri, G. Pasi, Fuzzy self-tuning PSO: a settings-free algorithm for global optimization, *Swarm Evol. Comput.* 39 (2018), 70–85.
- [49] F. Valdez, P. Melin, O. Castillo, A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation, *Expert Syst. Appl.* 41 (2014), 6459–6466.
- [50] Y. Tsukamoto, An approach to fuzzy reasoning method, in: M.M. Gupta, R.K. Ragade, R.R. Yager (Eds.), *Advances in Fuzzy Set Theory and Applications*, North-Holland Publishing Company, Amsterdam, Netherlands, 1979.
- [51] P. Angelov, R. Yager, Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density, in 2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS), IEEE, Paris, France, 2011, pp. 62–69.
- [52] X. He, Weighted fuzzy logic and its applications, in Proceedings COMPSAC 88: the Twelfth Annual International Computer Software & Applications Conference, IEEE Computer Society, Chicago, IL, USA, 1988, pp. 485–486.
- [53] N.N. Karnik, J.M. Mendel, Q. Liang, Type-2 fuzzy logic systems, *IEEE Trans. Fuzzy Syst.* 7 (1999), 643–658.
- [54] J. van den Berg, U. Kaymak, W.-M. van den Bergh, Probabilistic reasoning in fuzzy rule-based systems, in: P. Grzegorzewski, O. Hryniewicz, M.Á. Gil (Eds.), *Soft Methods in Probability, Statistics and Data Analysis*, Springer, Heidelberg, Germany, 2002, pp. 189–196.