IMPLEMENTING FUZZY ARITHMETIC

A. M. ANILE, S. DEODATO, AND G. PRIVITERA

June 1994

ABSTRACT. This paper presents the authors' efforts towards the development of an effective and friendly working tool for fuzzy arithmetic. It describes the two programming libraries currently produced. The first, implemented using the Fortran language, provides high precision computational capabilities, to demonstrate the feasibility and the correctness of a practical use of fuzzy arithmetic. The second, implemented in an object-oriented environmente using the $C++$ language, aims at being a powerful, friendly and highly portable tool to support generic users needing to manage imprecise data. Moreover, a simple application to environmental impact analysis is described to show the advantages deriving from the practical use of fuzzy arithmetic.

1. INTRODUCTION

The introduction of Fuzzy Sets Theory has changed the way ambiguity and imprecision are considered. In traditional theories we force our world representations to comply to extremely precise models, avoiding and rejecting imprecision as a perturbative factor. Nevertheless, imprecision is a very important way to represent information in real processes, where the increase in precision would otherwise become unmanageable. Fuzzy Set Theory allows the formalization of approximate reasoning, and preserves the original information contents of imprecision [11].

Any crisp theory can be fuzzified by generalizing sets within that theory to fuzzy sets. In particular, it is possible to introduce the concept of Fuzzy Number as the numerical representation of an imprecise knowledge about numerical quantities. Next, traditional arithmetic can be extended in order to deal with computations with fuzzy numbers. The resulting Fuzzy Arithmetic is now a well-formalized theory [5] that allows users to correctly manage uncertain values, such as those occurring in choosing among different alternatives, or those produced by the real-world measurement tools, or those resulting from subjective judgements of experts.

Key words and phrases. Fuzzy Arithmetic, Interval Analysis, Environmental Impact Analysis.
Submitted to Fuzzy Sets and Systems

Fuzzy Arithmetic applications are not widespread yet, mainly because of the lack of practical tools. However, several research fields may gain by using these computational methods. Some interesting areas where research is already growing are:

- Optimization related to engineering design [9].
- Image analysis and processing [8].
- Robotics [6].
- Environmental impact analysis [1].

In the following we will describe our implementation of some Fuzzy Arithmetic computational tools, together with an application to the field of environmental impact analysis. The fundamental questions arising in this context will be extensively investigated.

2. High Accuracy Fuzzy Arithmetic subroutines: a prototype

Our first implementation of a fuzzy arithmetic library was aimed at providing a prototype in order to support operations in high accuracy precision. This prototype can be used in order to check the correctness of fuzzy arithmetic codes in practical applications. The chosen working environment was an IBM ES9000 running a Fortran compiler with the IMSL/Math libraries. All the routines has been developed both in single and in double precision.

2.1. Fuzzy number definition. We define fuzzy numbers making use of interval analysis. In particular, we will relate two objects: confidence intervals and presumption levels [5]. A confidence interval is an interval of the reals that provides a representation for an imprecise numerical value by means of its sharpest enclosing range. A presumption level is an estimated truth value about some knowledge. Presumption levels belong to the $[0, 1]$ interval: we suppose the maximum of estimated truth to be at level 1, while we suppose the minimum at level 0. Once these definitions are provided, we represent a fuzzy number as an ordered set of confidence intervals, each of them providing the related numerical value at a given presumption level $\alpha \in [0, 1]$. These confidence intervals should comply with the relation

$$\alpha' > \alpha \Longrightarrow A_{\alpha'} \subset A_\alpha$$

where $\alpha, \alpha' \in [0, 1]$ and A_α, $A_{\alpha'}$ are the confidence intervals at presumption levels α and α'.

This definition follows the natural, often implicit, mechanism of human thinking in the subjective estimation of a numerical value when reasoning in one dimension.

2.2. Fuzzy arithmetic implementation. Once we have defined fuzzy numbers by means of confidence intervals, we can describe operations among fuzzy numbers as sequences of operations among confidence intervals.

In particular, let $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{B}}$ be fuzzy numbers and let $\diamond$ be a generic arithmetic operator. The fuzzy number $\tilde{\mathcal{A}} \diamond \tilde{\mathcal{B}}$ will be built by computing the operation $A_\alpha \diamond B_\alpha$

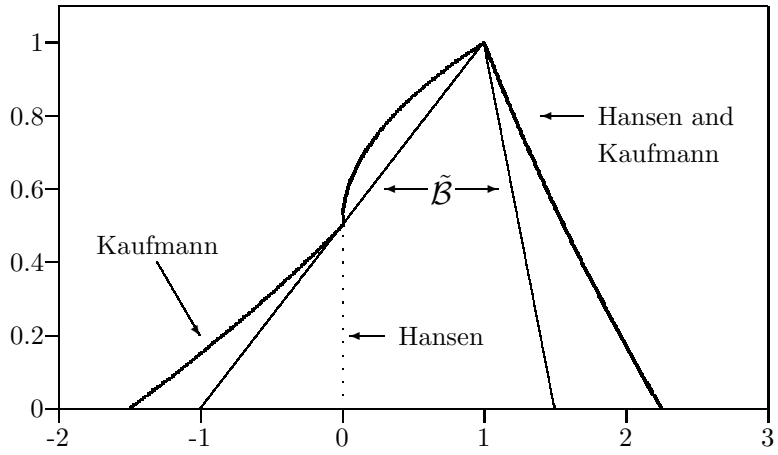

FIGURE 1. Different values of $\tilde{\mathcal{B}}^2$, with $\tilde{\mathcal{B}} = [-1, 1, 1.5]$ TFN, computed by using the Hansen and Kaufmann versions of the power operator.

for each $\alpha \in [0, 1]$, where A_α and B_α are the confidence interval of $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{B}}$ at presumption level α. It is proved that such a method complies with the extension principle of Zadeh [5].

2.2.1. *The interval arithmetic library.* First at all we have implemented a library for interval arithmetic. We based our work on Hansen's definitions [4], even if the literature about this subject is rather extensive and different definitions are often given for the same functions [5] [7]. When needed, these different definitions have been critically examined. In particular their related implementation have been widely tested and their relative performances have been assessed (e.g., the power function was implemented following either the Hansen and the Kaufmann definitions, see Figure 1).

In general, assuming X and Y to be intervals of reals and $\diamond$ to be one of the binary arithmetical operators, the correspondent interval operation is defined as

$$X \diamond Y = \{x \diamond y : x \in X, y \in Y\}.$$

That is, the resulting interval $X \diamond Y$ should contain all the possible values computable by $\diamond$ on items belonging to X and items belonging to Y.

Besides the traditional arithmetic operations, we introduced the cancellation operation [4] as

$$X \setminus Y = [a - c, b - d]$$

with $X = [a, b]$ and $Y = [c, d]$ confidence intervals. This operation evaluates 0 when $X = Y$, while the interval subtraction does not (see Figure 2), but otherwise it does not always produce a confidence interval as output.

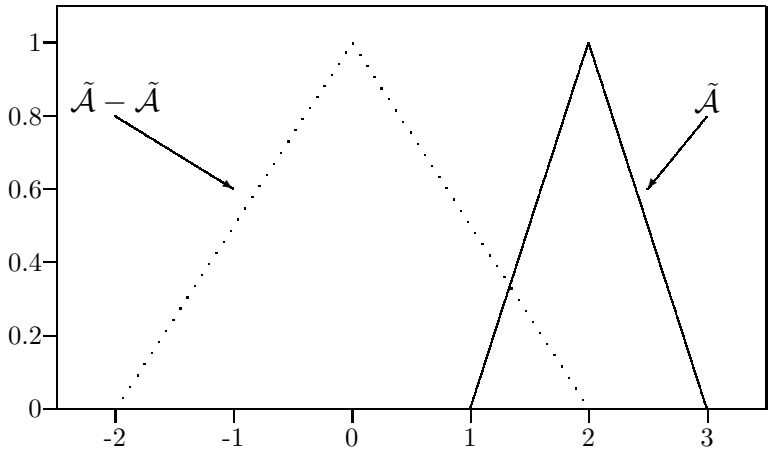

FIGURE 2. The computed value of $\tilde{\mathcal{A}} - \tilde{\mathcal{A}}$, with $\tilde{\mathcal{A}} = [1, 2, 3]$ TFN.

2.2.2. *The fuzzy arithmetic library.* We can represent a fuzzy number by means of a sequence of number NLIV of confidence intervals, or by explicitly providing its membership function. In the latter case we have implemented a function FZLIV that computes the required NLIV confidence intervals, as we use them to internally represent a fuzzy number. To achieve the maximum available precision, this function uses the IMSL routines UVMIF (to compute the maximum of the given membership function) and ZBREN (to compute the zeroes of the given membership funtion). The lower and upper bounds of the used confidence intervals are stored into two arrays, called A1 and A2. Each arithmetic operation among fuzzy numbers is computed by applying the given operation, once at a time, to all the confidence intervals used to represent the fuzzy operands.

Figures 3 and 4 shows some applications of the main arithmetic operators to fuzzy numbers.

2.3. Computing fuzzy functions. Arithmetic operations are simple functions defined on the real domain. As we want to provide users with the capability of computing generic real functions applied to fuzzy arguments, we need to classify these functions as [5]:

regular functions: the general monotonic increasing functions.
Let $\tilde{\mathcal{X}}$ be a fuzzy number and let $f : R \longrightarrow R$ be a regular function, then

$$\forall \alpha \in [0, 1] \qquad f(X_\alpha) = [f(a^{(\alpha)}), f(b^{(\alpha)})]$$

where $X_\alpha = [a^{(\alpha)}, b^{(\alpha)}]$ is the generic confidence interval used to represent $\tilde{\mathcal{X}}$ at the α presumption level.

non-regular functions: the general monotonic decreasing functions.

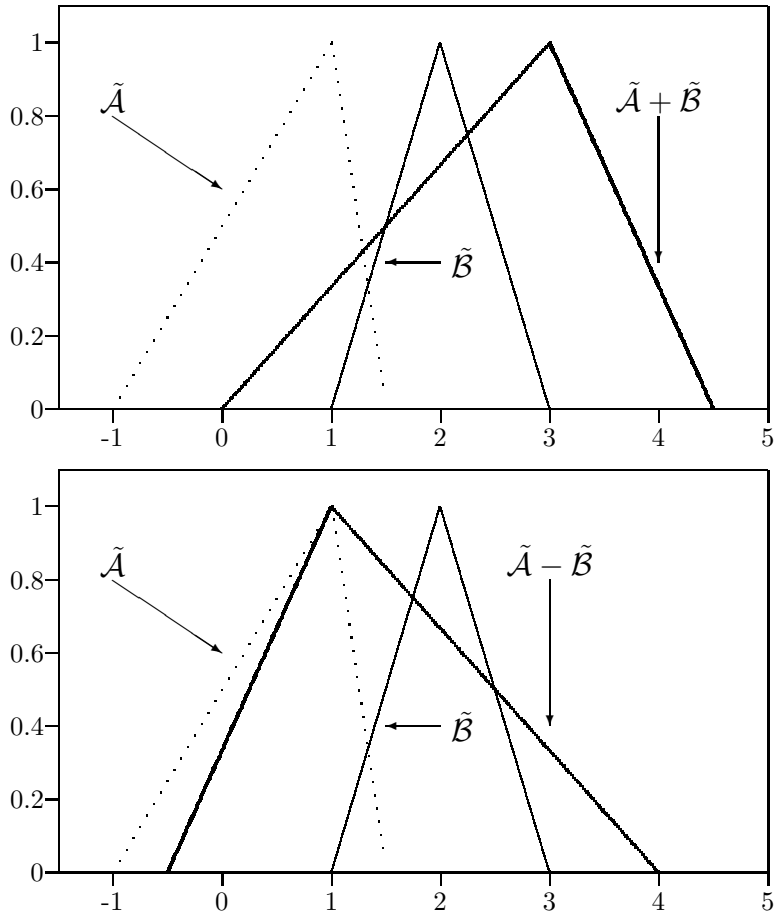

FIGURE 3. Some simple applications of fuzzy arithmetic operators.

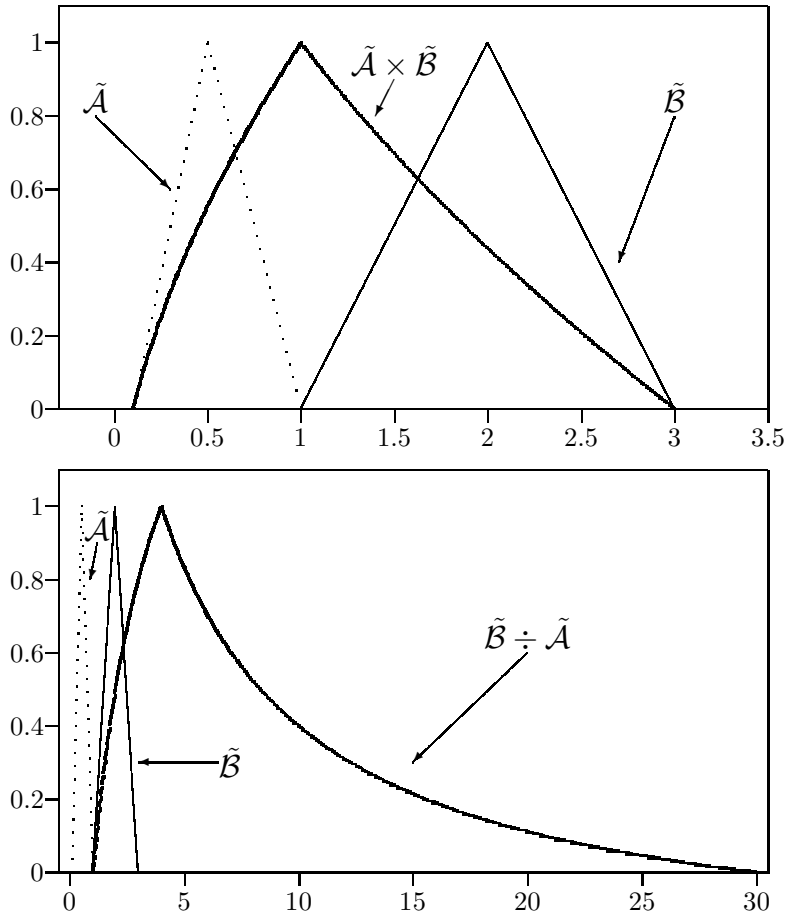

FIGURE 4. Some simple applications of fuzzy arithmetic operators.

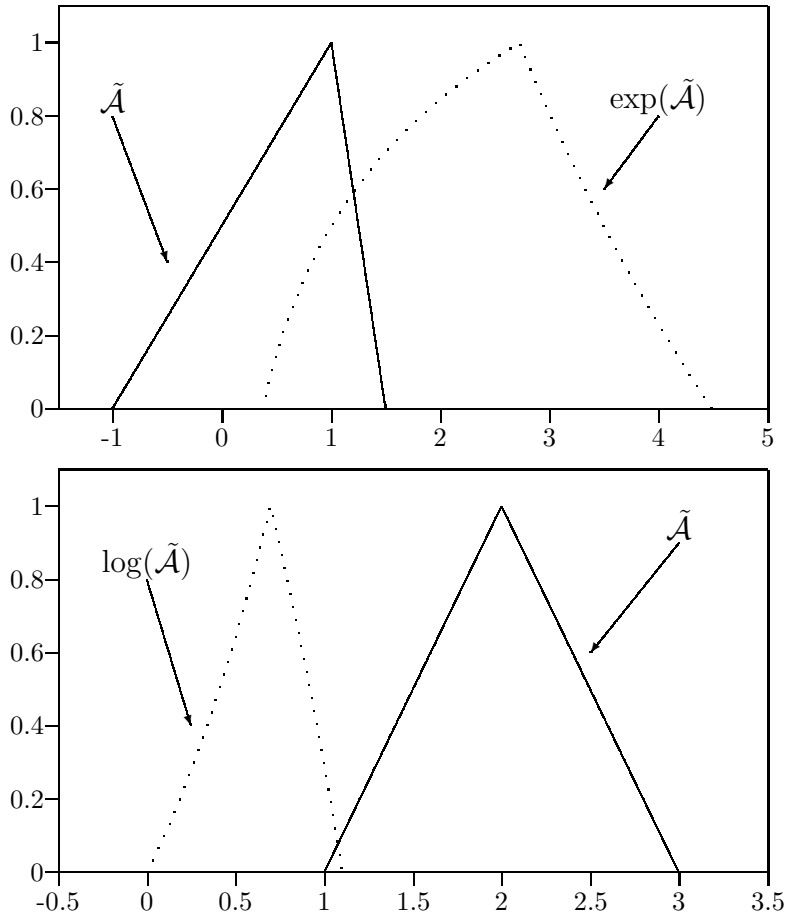

FIGURE 5. Some simple applications of fuzzy functions.

Let $\tilde{\mathcal{X}}$ be a fuzzy number and let $f : R \longrightarrow R$ be a non-regular function, then

$$\forall \alpha \in [0,1] \qquad f(X_\alpha) = [f(b^{(\alpha)}), f(a^{(\alpha)})]$$

where $X_\alpha = [a^{(\alpha)}, b^{(\alpha)}]$ is the generic confidence interval used to represent $\tilde{\mathcal{X}}$ at the α presumption level.

irregular functions: otherwise.

Let $\tilde{\mathcal{X}}$ be a fuzzy number and let $f : R \longrightarrow R$ be an irregular function. In this case the $f(X_\alpha)$ confidence intervals have to be computed one by one, by searching the absolute minimum and maximum of the function for the given interval.

By this way we have implemented the basic trigonometric functions, the logarithm function and the exponential function (see Figure 5).

2.4. The multiple occurrence problem. The main problem with computing fuzzy functions arises when calculating algebraic expressions containing multiple occurrences of the same fuzzy variables. In this case, different occurrences of the same variable will be considered as different variables, and the resulting fuzzy number will be less sharp than how it should be (See Figure 2). The problem arises from the interval analysis definitions which, assuming X to be a confidence interval and $\diamond$ to be a generic arithmetic operator, define

$$X \diamond X = \{x \diamond y : x \in X, y \in X\}$$

instead of

$$X \diamond X = \{x \diamond x : x \in X\}$$

The problem is well-known and several solutions have been discussed to fully solve it under particular conditions [2] [9] [10] or to limit its drawbacks [3].

2.4.1. *The Fuzzy Weighted Averages algorithm.* The FWA algorithm [2] provides us with a consistent way to consider only once the repeated values, using combinatorial arithmetic analysis to exactly compute fuzzy functions.

Let $f : R^n \longrightarrow R$ be a given function and let $\tilde{\mathcal{X}}_i, i = 1, ..., n$, be n fuzzy variables, each of them occurring as many times as we want into the definition of the f function. The FWA algorithm computes the $\tilde{\mathcal{Y}} = f(\tilde{\mathcal{X}}_1, ..., \tilde{\mathcal{X}}_n)$ fuzzy value by means of the following steps:

(1) First at all discretize the $[0, 1]$ interval into m levels $\alpha_j, j = 1, ..., m$. The refinement used in discretization will affect approximation accuracy of the resulting fuzzy value.
(2) Represent each variable $\tilde{\mathcal{X}}_i, i = 1, ..., n$ as a sequence of confidence intervals computed at the chosen α_j presumption levels, $j = 1, ..., m$.
(3) For each chosen $\alpha_j, j = 1, ..., m$, compute the 2^n vectors $(x_1, ..., x_n)$ obtained by permutation from the lower and upper bounds of all the confidence intervals resulting from the discretization of the $\tilde{\mathcal{X}}_i, i = 1, ..., n$, at the given α_j level. Each fuzzy variable $\tilde{\mathcal{X}}_i$ will appear only once in this process, whatever is the number of times it appears into the definition of the f function.
(4) For each chosen $\alpha_j, j = 1, ..., m$, evaluate the function $f(x_1, ..., x_n)$ for each of the obtained 2^n vectors, computing 2^n values $y_k, k = 1, ..., 2^n$.
(5) Compute the Y_j confidence interval representing the $\tilde{\mathcal{Y}}$ fuzzy value at the α_j presumption level by getting as its lower (upper) bound the minimum (maximum) of the y_k values computed for the given α_j.

The FWA algorithm can be used only for regular and non-regular functions, because it considers only the bounding values of the used confidence intervals. Nevertheless, this algorithm can be improved in order both to reduce its complexity and to apply to some irregular functions [9].

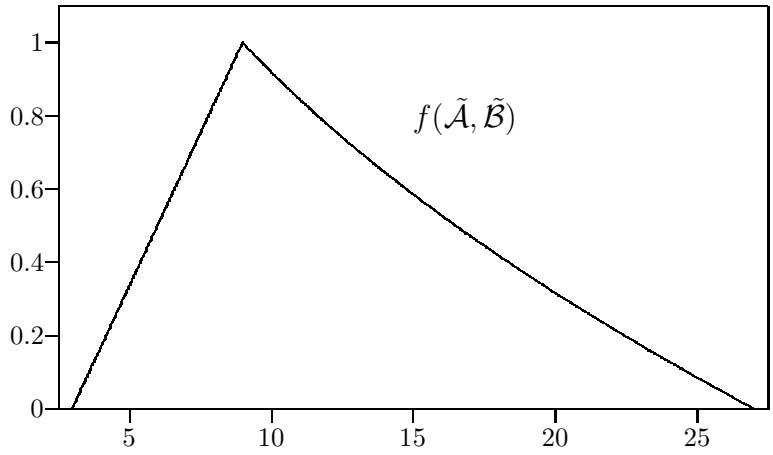

FIGURE 6. Evaluation of the function $f(\tilde{\mathcal{X}}_1, \tilde{\mathcal{X}}_2) = \tilde{\mathcal{X}}_1^2 - \tilde{\mathcal{X}}_1\tilde{\mathcal{X}}_2 + \tilde{\mathcal{X}}_2 + 8$ on the fuzzy numbers $\tilde{\mathcal{A}} = [-1, 1, 2]$ and $\tilde{\mathcal{B}} = [4, 8, 9]$, TFNs.

We have implemented the basic FWA algorithm as a routine, called FZWA, that works on the definition of a fuzzy function and on the related set of fuzzy arguments.

2.4.2. *The Yang-Yao-Dewey algorithm.* Yang, Yao and Dewey [10] provide a more effective FWA based algorithm to compute functions of fuzzy numbers. The basic FWA algorithm works on the permutations of the lower and upper bounds of the confidence intervals introduced by discretizing the $[0, 1]$ presumption interval. By this way it misses the possible minimal or maximal values that are internal into the examined confidence intervals (this algorithm works on regular and non-regular functions only). The Yang and al. algorithm overcomes this problem by working on particular values, called *poles*, that are candidate to be extremal points for the given function.

Let $f : R^n \longrightarrow R$ be a given function and let $\tilde{\mathcal{X}}_i, i = 1, ..., n$, be n fuzzy variables, each of them occurring as many times as we want into the definition of the f function. If we discretize the $[0, 1]$ presumption interval into a set of α levels, the core of the FWA algorithm is to solve the relations

$$Y_\alpha = f(X_{1\alpha}, ..., X_{n\alpha}) = [y^L, y^R]$$

for each $\alpha \in [0, 1]$, where y^L and y^R are the global minimum and maximum in the $X_\alpha = X_{1\alpha} \times ... \times X_{n\alpha}$ space. Yang et al. introduced the following classification on X_α spaces:

uniform spaces: when X_α has no subspace in which $\Phi_i = 0, i = 1, ..., n$, with Φ_i partial first derivative of the given function.
The global minimum (maximum) of the given f function in the X_α space is the $(x_1, ..., x_n)$ vector where each x_i is the lower (upper) bound of the X_i

interval if $\Phi_i > 0$, or its upper (lower) bound if $\Phi_i < 0$.

quasi-uniform spaces: when X_α has some subspaces in which $\Phi_i = 0, i = 1, ..., n$, with Φ_i partial first derivative of the given function and independent of the x_i variable.
The global minimum and maximum of the given f function in the X_α space are computed by the standard FWA argorithm.

non-uniform spaces: otherwise.
The global minimum and maximum of the given f function in the X_α space have to be computed solving a non-linear optimization problem. Nevertheless, if the given f function is a polynomial of degree less than three, we have implemented an algorithm that computes the extremal values without solving a non-linear problem (see Figure 6).

3. A working tool for fuzzy arithmetic

Using fuzzy arithmetic can be made simpler and more popular if supported by an adequate working tool. As a fuzzy number is just another type of numerical object, users should manage it as any other number (e.g., integers or reals). That is, what the users' community needs is a nice working tool to define and use the fuzzy number new data type.

This working tool for fuzzy arithmetic has to provide users with all the numerical operations and functions they need for fuzzy numbers (e.g., arithmetical operators, trigonometrical functions, and so on). Moreover, this working tool has to provide users with a set of functionalities related to fuzzy numbers own features (e.g., I/O functionalities, graphic representation and manipulation functionalities, and so on). At the same time, this working tool has to hide the implementation of these functionalities and the underlying internal structure used to represent fuzzy numbers. By such a working tool, users could really manage fuzzy numbers in a very simple and transparent way, exactly as they do with other traditional numbers.

The best environment to produce such a working tool is an object-oriented programming one. This environment allows us to easily define new abstract data types, to cleanly manage them by overloading standard language operators, and to powerfully extend them by inheritance mechanisms. In this context, the $C{+}{+}$ programming language is a worldwide supported industry standard to produce highly maintainable and portable products. So, we choose the $C++$ programming language to build our implementation of a programmers' library defining a fuzzy number new data type.

3.1. New data types definitions. We represent a fuzzy number as an ordered set (of fixed cardinality) of confidence intervals [5]. Each of these intervals is the value of the number at a given presumption level $\alpha \in [0, 1]$, with the condition that the lower (upper) bound value of the intervals is a monotonic increasing (decreasing) functions of α. By this way we can reduce computations on fuzzy numbers to sequences of

equivalent computations on the correspondent sets of confidence intervals. Thus, our library should contain at least two new data type implementation: fuzzy numbers and confidence intervals.

3.1.1. *The confidence interval data type.* Confidence intervals are the first abstract data type we need to define. The generic confidence interval representation is trivial, as we need to know only its lower and upper bound values. Initializing these objects is very simple and conversion functions from singletons to intervals are easily provided. Rules for performing interval arithmetic and interval analysis are well-formalized [4] [5] [7] and not difficult to implement. In general, when applying a function to a given set of intervals, the lower (upper) bound of the resulting interval will be the minimum (maximum) value computed applying that function to all the possible combination of the values belonging to the argument intervals.

Our current implementation of the confidence interval new data type provides all the basic arithmetic unary and binary operators and the basic trigonometric functions. In particular, the arithmetic operators work either on couples of confidence intervals or on couples of confidence intervals and singletons. Additional functionalities are provided to compute the minimum and the maximum between two confidence intervals, and to evaluate the distance between them. Other functions (such as power, square root, exponential, logarithm, and so on) are not difficult to compute and will be implemented as soon as needed.

Moreover, our current implementation manages only finite confidence intervals. A version able to manage also infinite confidence intervals can be easily produced, using tools which support the IEEE standard for floating-point arithmetic. This standard provides a representation of the $+\infty$ and $-\infty$ values and then we will only have to modify our code to correctly manage these values [4].

3.1.2. *The fuzzy number data type.* Once defined the confidence interval data type, we can define the fuzzy number data type. The fuzzy number internal representation is not very complex, and is described by a fixed-length ordered set (i.e., an array) of confidence intervals, one for each considered presumption level $\alpha \in [0, 1]$. The number of used presumption levels and the corresponding α values have to be the same for all the objects belonging to the fuzzy number data type. Moreover, this number of used presumption levels must be large enough to give a comprehensive description of the whole represented fuzzy number.

Initializing such an internal structure is not a simple task, as we do not want to force users to provide all the values for the required confidence intervals. Our tool should be so friendly to allow users to specify only the values they want, and so powerful to derive the missing data from the provided ones. Using the $C++$ name overloading capabilities we can get this result by simply implementing as many initializing functions as we need. Each of these functions will allow users to specify only a particular subset of the required data, and will correctly use them to initialize

the whole structure. At the moment we have implemented such functions to convert singletons to fuzzy numbers and to initialize TFNs (Triangular Fuzzy Numbers). More general initializing functions would require adequate interpolation routines and will be implemented as soon as needed.

When applying a function to a given set of fuzzy numbers, the resulting fuzzy number will be built computing, one by one, its values of presumption levels. Each of these values will be computed applying the correspondent interval function to the confidence intervals of each operand at the given presumption level. Thus, our implementation of operators and functionalities available for the fuzzy number data type reflects the confidence interval data type one. In particular, we provide all the basic arithmetic unary and binary operators and the basic trigonometric functions. The arithmetic binary operators work either on couples of fuzzy numbers or on couples of fuzzy numbers and singletons. Additional functionalities are provided to compute the minimum and the maximum between two fuzzy numbers, and to evaluate a dissemblance index between them. Other functions (such as power, square root, exponential, logarithm, and so on) are not difficult to compute and will be implemented as soon as needed.

3.2. Goodies and blames. The definitions of the confidence interval and fuzzy number data type are the core of our working tool for fuzzy arithmetic. Even without adding any other functionality, the resulting $C++$ library will provide users with all the necessary features for the required working tool. To fully understand the powerness and the friendliness of this tool, let us look at the following code fragment:

```
...
#include "fuzzy.h"
...
Fuzzy fzNum1, fzNum2;
Fuzzy fzNum3( 2, 3.5, 6.8);
...
fzNum1 = Fuzzy( 3.8, 4.2, 4.9);
...
fzNum2 = sin( 3 * fzNum1);
...
fzNum2 = fzNum1 - fzNum3;
...
```

It is not necessary to be either a $C++$ guru or an expert programmer to understand it: first the new data type definitions are included into the program (first line); then, some instructions later, three objects are declared belonging to the fuzzy number data type, two without any explicit initialization (second line) and one initialized with a TFN value (third line); next, a TFN value is assigned to a fuzzy number variable (fourth line); at last, some simple computation involving fuzzy numbers and

α	interval value
0.0	$[1.0, 7.0]$
0.3	$[2.0, 6.0]$
0.7	$[3.0, 5.0]$
1.0	$[4.0, 4.0]$

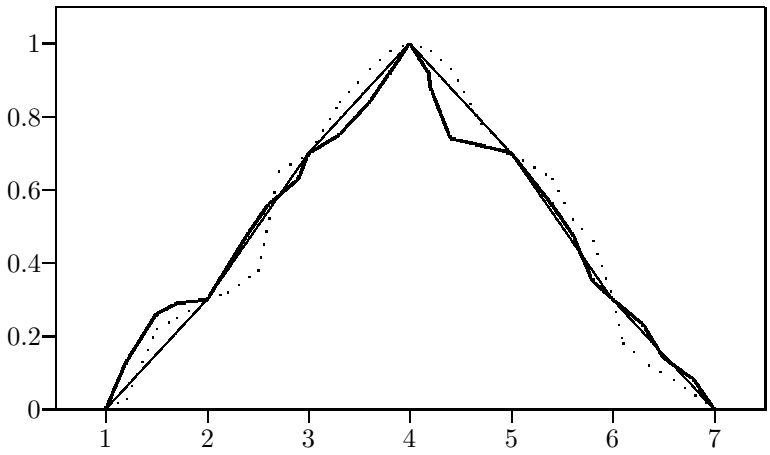

FIGURE 7. A simple numerical representation of a fuzzy number and some different graphical representations that comply with it.

an integer value are performed (fifth and sixth lines). All a generic user needs to know to understand or to produce such a code is what is a TFN. All the used functions and arithmetic operators keep their original syntax and semantics. No knowledge at all is needed about internal representation of generic fuzzy numbers and TFNs.

The only problem we experimented with our tool is something inherent in the definition itself of interval arithmetic. When computing an algebraic expression where the same variable appears more than once, the identity of this variable in its different occurrences is lost and the resulting interval is wider than it should be, although it contains the correct interval [4]. The problem is well-known in interval analysis and some algorithms have been developed to reduce this effect [2] [3] [10] (we implemented many of them in our early high accuracy fuzzy arithmetic subroutines). However, all these algorithms require a pre-processing of the whole algebraic expression, while our tool is not able to do that as each operator is evaluated by itself and not as a part of an expression.

Clearly, this problem limits the application range of our tool and we plan to develop a companion library to provide sharper computational methods when needed. Nevertheless, there are many applications that can be managed by our current tool without running into such a problem. For all these cases the described library will

provide users with a simple and effective way to manage imprecision.

3.3. The graphical interface. To fully manage imprecise knowledge about numerical quantities we not only need to provide users with computational capabilities for fuzzy numbers, but we also need to implement some utilities to allow users to loosely manipulate these objects. Indeed, there are many circumstances in which the concept of fuzzy number is better managed by a vague visual representation rather than by a rigid numerical one.

As a case in point, let us to examine the initialization of a fuzzy number. Generally, users do not know all the required values for the used presumption levels, so we introduced some functionalities able to derive the missing values from the provided ones. Nevertheless, users' knowledge about a fuzzy number is often enhanced by some vague information on the shape of the membership function of that number. This knowledge is not characterizable by any numerical value and there is no way to effectively represent such a kind of information other than by using a graphical environment (see Figure 7). Thus, what users need to really exploit all the knowledge they have about a fuzzy number is a set of graphical utilities to input the values of the known presumption levels, to visualize the resulting membership function after deriving the missing values, and to modify it in order to comply to the expected shape. Once introduced, these graphical utilities will be useful not only for initialization of fuzzy numbers, but also in all cases involving loosely manipulations of these values.

To make our tool as powerful and user-friendly as possible we have to add these graphical utilities to the defined new data types. In particular, we need to provide each fuzzy number with the capabilities of drawing itself on a given canvas and, more generally, of complying to the behaviour of the given canvas (i.e., the graphical representation of a fuzzy number has to adjust itself to some varying parameters of the used canvas, such as the canvas size or the canvas scale factors). Moreover, we need to provide users with functionalities to graphically initialize and manipulate a fuzzy number by means of a mouse or an equivalent pointing device.

Using an object-oriented environment we can enhance our fuzzy number data type without redefining it. The inheritance mechanisms allow us to define a new data type, a graphical fuzzy number data type, by derivation from the early defined one. This new graphical fuzzy number data type will provide users with the same computational capabilities of the original fuzzy number data type; in addition, it will provide all the functionalities required to graphically manage the fuzzy number. The early fuzzy number data type definition will be the only repository of the fuzzy number numerical representation and all the related computational capabilities. The new graphical fuzzy number data type definition will be the only repository of the fuzzy number graphical representation and all the related manipulation capabilities, and will automatically refer to the early fuzzy number data type definition for the numerical calculations. Users interested in pure numerical computations can refer to

the simple fuzzy number data type definition, while user interested in graphical manipulation have to refer to the graphical fuzzy number data type definition, getting at the same time all the computational capabilities.

At the moment our graphical extensions of the fuzzy number working tool are at an advanced development stage. As we want to build a very general programming tool, we don't like to closely rely on any existing GUI environment, but we want to let the user free to use its preferred one. So, we have chosen to work on a general application framework that assures us the platform independence of the produced code. By this way, we hope to make soon available a graphical version of our working tool for the main existing GUI environments.

4. A simple application to environmental impact analysis

Environmental Impact Analysis provides us a good case in point to test the powerness and the effectiveness of practical use of fuzzy arithmetic (in general) and our tool (in particular). To briefly describe it, this analysis is a technical procedure to evaluate how human buildings (from the simple ones, such as an irrigation system, to the very complex ones, such as a hydroelectric central) will affect the natural and social environment in which they will be realized. Environmental Impact Analysis is now widely accepted around the world, and many states issued specific laws to require and to regulate its application. In particular, when necessary, we will refer to the italian laws about this subject. The described approach to the problem is more deeply discussed in [1].

The core of an environmental impact analysis process is the construction and the evaluation of a particular Leopold matrix, a bidimensional check-list relating the main actions of a given project and their considered effects on some analyzed environmental components. Each item $A_{m,n}$ of this matrix describes the impact of the m-th action for the n-th effect. Null items stand for unrelated couples of actions and effects, while non-null items provide evaluations for the related ones. For each impact evaluation there are four parameters to be considered: sign, magnitude, possibility, and persistence in time. The degree of an impact is computed as the product (with its given sign, used to point out positivity or negativity of the impact) of its correspondent magnitude, possibility and persistence. Once computed all the items of the matrix, it is necessary to aggregate them to get a global evaluation of the environmental impact of the given project. Usually, this is obtained by computing the mean values of the columns (rows) of the matrix, getting an evaluation vector for the considered effects (actions) on the analyzed environmental factors. This vector is the final output of the whole process.

Environmental Impact Analysis is a very complex procedure, involving contributions from several technical, economic and human sciences. The evaluation of a Leopold matrix is based on the subjective judgement of an expert. The parameters required to compute impact evaluations have imprecise value and are often expressed

	farming	developing properties	social agreement	free time	prevention
irrigation	positive middle certain permanent	positive middle certain permanent	positive middle likely permanent	positive low likely permanent	positive low unlikely permanent
drinking water	positive low likely permanent	positive high certain permanent	positive high certain permanent	positive middle likely permanent	positive low unlikely permanent
river controlling	positive middle certain permanent	positive middle certain permanent	positive middle likely permanent	positive low unlikely permanenet	positive high certain permanent
sports and recreative use	positive middle likely permanent	positive low likely permanent	positive middle certain permanent	positive middle certain permanenet	positive low unlikely permanent
environmental use	positive low likely permanent	positive middle likely permanent	positive middle certain permanent	positive middle certain permanenet	positive middle certain permanent

FIGURE 8. A Leopold sub-matrix that estimates the effects of river's use on economic and social factors.

using linguistic values such as *high*, *middle* or *low*. Using traditional computational tecniques we have to choose a crisp value in the $[0, 1]$ interval for each used linguistic value. As these choices will influence final results, it is very important to determine the results' sensitivity to their change. This process is computationally heavy as we have to change the chosen values (one by one or by groups, using a random number generator if possible), to compute each time the evaluation vector, and to compare all the resulting vectors by adequate metrics. Using fuzzy arithmetic, instead, we can manage the used linguistic values with numerical values that transparently take into account the imprecison due to a subjective judgement. That is, we can evaluate the Leopold matrix getting a resulting vector of fuzzy numbers that keeps all the hints of the original subjective judgements. If the used fuzzy arithmetic tool provides graphical representation too, the whole process of building and evaluating the Leopold matrix can be realized in a very simple, effective and user-friendly way.

4.1. A practical case in point. Let us suppose to evaluate an environmetal impact analysis related to some fluvial building. One of the main subset of possible actions will be the one related to the river's use, while one of the main subset of possible effects will be the one related to the economic and social factors. To illustrate the use of fuzzy arithmetic into environmental impact analysis, we will evaluate the Leopold sub-matrix that estimates the effects of river's use on economic and social factors (see Figure 8).

Once we have built the Leopold matrix, in order to evaluate it we need to assign a numerical value to each used linguistic value. The used fuzzy numbers are shown in Figure 9.

Then we can use fuzzy arithmetic to compute the evaluation vector (see Figure 10).

5. Conclusions

Fuzzy arithmetic provides users with the instruments to effectively manage imprecise knowledge about numerical quantities. Our experience in designing, implementing and using the described working tools has proved the feasibility and the suitability of practical use of fuzzy arithmetic. The planned integration between computational capabilities and graphical representation will furtherly increment the power and the friendliness of fuzzy arithmetic working tools, extending the number of potential users too.

References

1. A. M. Anile, G. Cammarata and A. M. Greco, *Metodologia della teoria degli insiemi fuzzy per le elaborazioni di valutazioni di impatto ambientale* in *Proceedings of the Conference "Metodologia della teoria degli insiemi fuzzy per le elaborazioni di impatto ambientale"*, Accademia Gioenia di Scienze Naturali, Catania, 1993.
2. W. M. Dong and F. S. Wong, *Fuzzy weighted averages and implementation of the extension principle*, Fuzzy Sets and Systems 21 (1987) 183-199.
3. E. R. Hansen, *A generalized interval arithmetic* in K. L. Nickel (ed.), *Interval Mathematics*, Springer-Verlag, New York, 1975.
4. E. R. Hansen, *Global optimization using interval analysis*, Marcel Dekker, Inc., New York, 1992.
5. A. Kaufmann and M. M. Gupta, *Introduction to Fuzzy Arithmetic: Theory and Applications*, Van Nostrand Reinhold, New York, 1991
6. W. J. Kim, J. H. Ko and M. J. Chung, *Uncertain robot environment modelling using fuzzy numbers*, Fuzzy Sets and Systems 61 (1994) 55-62.
7. R. E. Moore, *Interval Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.
8. S. K. Pal and A. Ghosh, *Fuzzy Geometry in image analysis*, Fuzzy Sets and Systems 48 (1992) 23-40.
9. K. L. Wood, K. N. Otto and E. K. Antonsson, *Engineering design calculations with fuzzy parameters*, Fuzzy Sets and Systems 52 (1992) 1-20.
10. H. Q. Yang, H. Yao and J. D. Jones, *Calculating functions of fuzzy numbers*, Fuzzy Sets and Systems 55 (1993) 273-283.

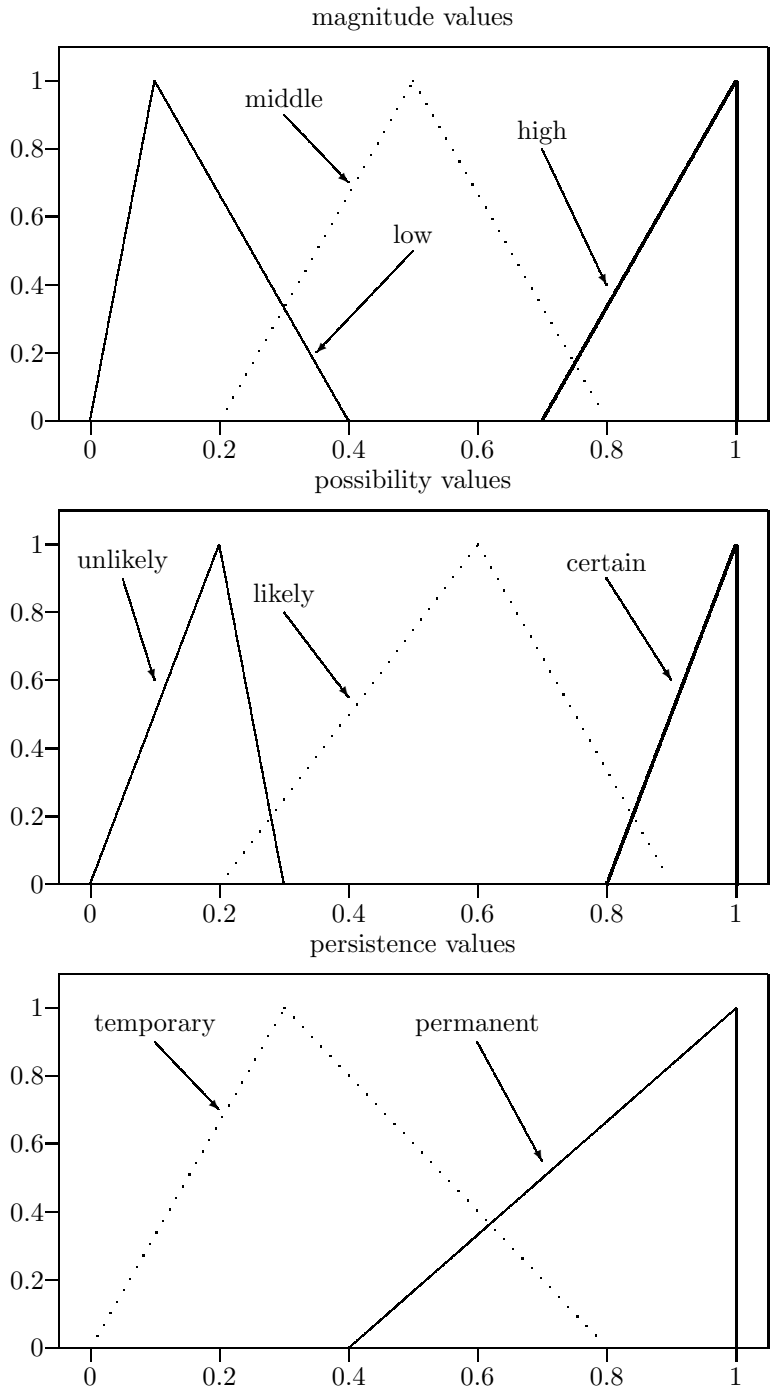

FIGURE 9. The fuzzy numbers chosen to quantify the linguistic values used in the considered Leopold matrix.

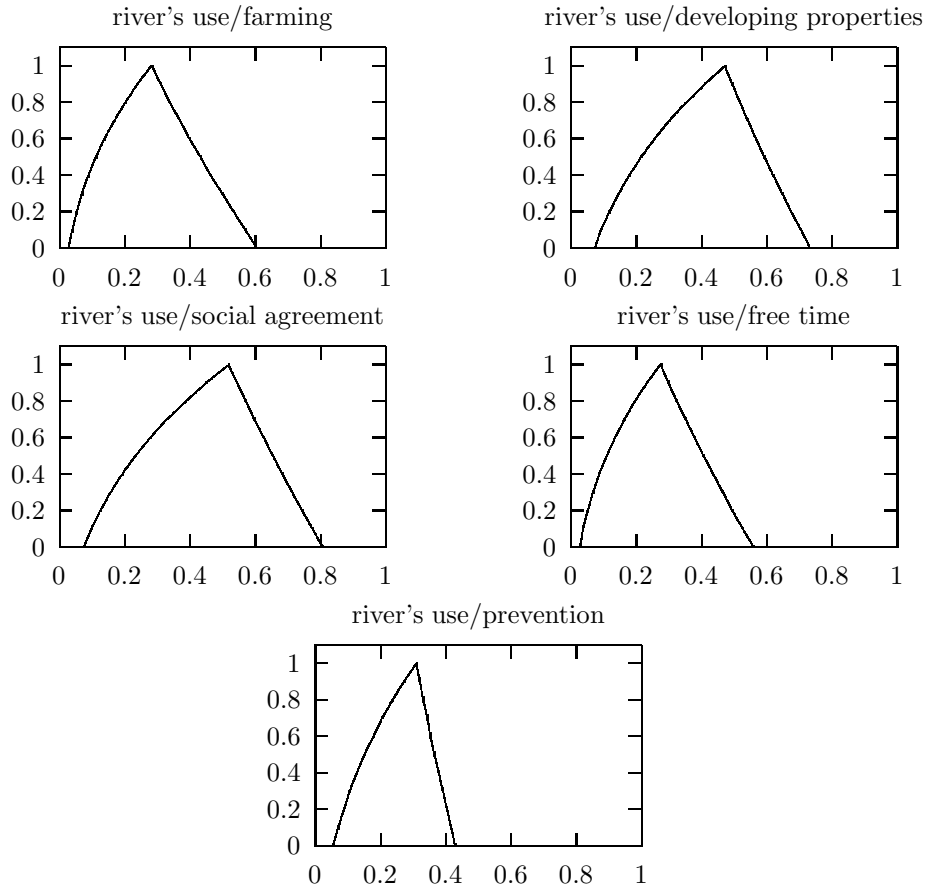

FIGURE 10. The components of the computed evaluation vector for the considered effects of river's use on economic and social factors.

11. L. A. Zadeh, *Fuzzy Logic, Neural Networks and Soft Computing*, Communication of the ACM, March 1994/Vol. 37, No. 3.

Dipartimento di Matematica, Universitá degli Studi di Catania, viale Andrea Doria 6, I-95125 Catania (Italy)
E-mail address: anile@dipmat.unict.it

Dipartimento di Matematica, Universitá degli Studi di Catania, viale Andrea Doria 6, I-95125 Catania (Italy)
E-mail address: deodato@dipmat.unict.it

Co.Ri.M.Me., stradale Primo Sole 50, I-95100 Catania (Italy)
E-mail address: guido@dipmat.unict.it