A Guide to the FuzzyNumbers Package for R
(FuzzyNumbers version 0.4-2)

Marek Gagolewski\textsuperscript{1,2}
\textsuperscript{1} Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
\textsuperscript{2} Rexamine, Email: gagolews@rexamine.com
Jan Caha\textsuperscript{3}
\textsuperscript{3} Institute of Geoinformatics, VŠB - Technical University of Ostrava
Email: jan.caha@klikni.cz

\url{http://FuzzyNumbers.rexamine.com}

February 13, 2018

Any suggestions and contributions are welcome!

Contents

1 Getting Started 2

2 How to Create Instances of Fuzzy Numbers 3
  2.1 Arbitrary Fuzzy Numbers .................................. 3
    2.1.1 Definition by Side Functions .......................... 3
    2.1.2 Definition by $\alpha$-cut Bounds ..................... 5
    2.1.3 Definition with Generating Functions Omitted: Shadowed Sets 6
  2.2 Using Numeric Approximations of $\alpha$-cut or Side Generators .......... 7
  2.3 Trapezoidal Fuzzy Numbers ................................ 8
  2.4 Piecewise Linear Fuzzy Numbers ........................... 10
  2.5 Fuzzy Numbers with Sides Given by Power Functions ................. 14

3 Depicting Fuzzy Numbers 15

4 Basic Computations on and Characteristics of Fuzzy Numbers 18
  4.1 Support and Core, and Other $\alpha$-cuts .......................... 19
  4.2 Membership Function Evaluation ................................ 19
  4.3 “Typical” Value .............................................. 20
  4.4 Measures of “Nonspecificity” .................................. 21

5 Operations on Fuzzy Numbers 21
  5.1 Arithmetic Operations ........................................ 21
  5.2 Applying Functions ............................................ 22
    5.2.1 Special Functions ..................................... 23
1 Getting Started

Fuzzy set theory lets us quite intuitively represent imprecise or vague information. Fuzzy numbers (FNs), introduced by Dubois and Prade in [10], form a particular subclass of fuzzy sets of the real line. Formally, a fuzzy set \( A \) with membership function \( \mu_A : \mathbb{R} \to [0, 1] \) is a fuzzy number, if it possess at least the three following properties:

(i) it is a normalized fuzzy set, i.e. \( \mu_A(x_0) = 1 \) for some \( x_0 \in \mathbb{R} \),

(ii) it is fuzzy convex, i.e. for any \( x_1, x_2 \in \mathbb{R} \) and \( \lambda \in [0, 1] \) it holds \( \mu_A(\lambda x_1 + (1 - \lambda) x_2) \geq \mu_A(x_1) \wedge \mu_A(x_2) \),

(iii) the support of \( A \) is bounded, where \( \text{supp}(A) = \text{cl}(\{x \in \mathbb{R} : \mu_A(x) > 0\}) \).

Fuzzy numbers play a significant role in many practical applications (cf. [20]) since we often describe our knowledge about objects through numbers, e.g. “I’m about 180 cm tall” or “The rocket was launched between 2 and 3 p.m.”.

FuzzyNumbers is an Open Source (licensed under GNU LGPL 3) package for R – a free software environment for statistical computing and graphics, which runs on all major operating systems, i.e. Windows, Linux, and MacOS X.

FuzzyNumbers has been created in order to deal with fuzzy numbers conveniently and effectively. To install latest “official” release of the package available on CRAN we type:

```r
install.packages('FuzzyNumbers')
```

Alternatively, we may fetch its current development snapshot from GitHub:

```r
install.packages('devtools')
library('devtools')
install_github('FuzzyNumbers', 'gagolews')
```

1 Please visit R Project’s homepage at [www.R-project.org](http://www.R-project.org) for more details. Perhaps you may also wish to install RStudio, a convenient development environment for R. It is available at [www.rstudio.com/ide](http://www.rstudio.com/ide).
Each session with FuzzyNumbers should be preceded by a call to:

```r
library('FuzzyNumbers') # Load the package
```

To view the main page of the manual we type:

```r
library(help='FuzzyNumbers')
```

For more information please visit the package’s homepage [13]. In case of any problems, comments, or suggestions feel free to contact the author. Good luck!

# How to Create Instances of Fuzzy Numbers

## 2.1 Arbitrary Fuzzy Numbers

A fuzzy number $A$ may be defined by specifying its core, support, and either its left/right side functions or lower/upper $\alpha$-cut bounds. Please note that many algorithms that deal with FNs assume we provide at least the latter, i.e. $\alpha$-cuts.

### 2.1.1 Definition by Side Functions

A fuzzy number $A$ specified by side functions[^2] has membership function of the form:

$$
\mu_A(x) = \begin{cases} 
0 & \text{if } x < a_1, \\
\text{left} \left( \frac{x-a_1}{a_2-a_1} \right) & \text{if } a_1 \leq x < a_2, \\
1 & \text{if } a_2 \leq x \leq a_3, \\
\text{right} \left( \frac{x-a_3}{a_4-a_3} \right) & \text{if } a_3 < x \leq a_4, \\
0 & \text{if } a_4 < x,
\end{cases}
$$

(1)

where $a_1, a_2, a_3, a_4 \in \mathbb{R}$, $a_1 \leq a_2 \leq a_3 \leq a_4$, $\text{left} : [0, 1] \rightarrow [0, 1]$ is a nondecreasing function (called the left side generator of $A$), and $\text{right} : [0, 1] \rightarrow [0, 1]$ is a nonincreasing function (right side generator of $A$). In our package, it is assumed that these functions fulfill the conditions $\text{left}(0) \geq 0$, $\text{left}(1) \leq 1$, $\text{right}(0) \leq 1$, and $\text{right}(1) \geq 0$. Note that this is a so-called L-R representation of FNs, see [13].

An example: a fuzzy number $A_1$ with linear sides (a trapezoidal fuzzy number, see also Sec. 2.3).

```r
A1 <- FuzzyNumber(1, 2, 4, 7, 
  left=function(x) x, 
  right=function(x) 1-x
)
```

This object is an instance of the following R class:

```r
class(A1)
## [1] "FuzzyNumber" 
## attr(,"package")
## [1] "FuzzyNumbers"
```

We may print some basic information on $A_1$ by calling `print(A1)` or simply by typing:

[^2]: Side functions are sometimes called branches or shape functions in the literature.
To depict $A_1$ we call:

```r
plot(A1)
```

![Plot of A1](image)

**Remark.** Please note that by using side generating functions defined on $[0, 1]$ we really make (in the author’s humble opinion) the process of generating examples for our publications much easier. A similar concept was used e.g. in [21] (LR-fuzzy numbers).

Assume, however, that we are given two fancy side functions $f : [a_1, a_2] = [-4, -2] \rightarrow [0, 1]$, and $g : [a_3, a_4] = [-1, 10] \rightarrow [1, 0]$, for example:

```r
f <- splinefun(c(-4, -3.5, -3, -2.2, -2), c(0, 0.4, 0.7, 0.9, 1), method='monoH.FC')
g <- splinefun(c(-1, 0, 10), c(1, 0.5, 0), method='monoH.FC')
```

We should convert them to side *generating* functions, which shall be defined on the interval $[0, 1]$. This may easily be done with the `convertSide()` function. It returns a new function that calls the original one with linearly transformed input.

```r
convertSide(f, -4, -2)(c(0,1))
# [1] 0 1
convertSide(g, -1, 10)(c(0,1))
# [1] 1 0
convertSide(g, 10, -1)(c(0,1)) # interesting!
# [1] 0 1
```

These functions may be used to define a fuzzy number, now with arbitrary support and core.

```r
B <- FuzzyNumber(10, 20, 20, 30, 
    left=convertSide(f, -4, -2), 
    right=convertSide(g, -1, 10) 
)
plot(B, xlab=expression(x), ylab=expression(alpha))
```
2.1.2 Definition by α-cut Bounds

Alternatively, a fuzzy number $A$ may be defined by specifying its $\alpha$-cuts. We have (for $\alpha \in (0, 1)$ and $a_1 \leq a_2 \leq a_3 \leq a_4$):

$$A_\alpha := [A_L(\alpha), A_U(\alpha)] \quad (2)$$

$$= [a_1 + (a_2 - a_1) \cdot \text{lower}(\alpha), a_3 + (a_4 - a_3) \cdot \text{upper}(\alpha)], \quad (3)$$

where \(\text{lower} : [0, 1] \rightarrow [0, 1]\) is a nondecreasing function (called \textit{lower $\alpha$-cut bound generator of $A$}), and \(\text{upper} : [0, 1] \rightarrow [0, 1]\) is a nonincreasing function (\textit{upper bound generator}). In our package, we assume that \(\text{lower}(0) = 0, \text{lower}(1) = 1, \text{upper}(0) = 1, \text{and upper}(1) = 0\).

It is easily seen that for $\alpha \in (0, 1)$ we have the following relationship between generating functions:

$$\text{lower}(\alpha) = \inf\{x : \text{left}(x) \geq \alpha\}, \quad (4)$$

$$\text{upper}(\alpha) = \sup\{x : \text{right}(x) \geq \alpha\}. \quad (5)$$

Moreover, if side generating functions are continuous and strictly monotonic, then $\alpha$-cut bound generators are their inverses.

An example:

```r
A1 <- FuzzyNumber(1, 2, 4, 7,
    left=function(x) x,
    right=function(x) 1-x
)
A2 <- FuzzyNumber(1, 3, 4, 7,
    lower=function(alpha) pbeta(alpha, 5, 9), # CDF of a beta distr.
    upper=function(alpha) pexp(1/alpha-1) # transformed CDF of an exp. distr.
)
plot(A1, col='blue')
plot(A2, col='red', lty=2, add=TRUE)
legend('topright', c(expression(mu[A1]), expression(mu[A2])),
    col=c('blue', 'red'), lty=c(1,2))
```
Remark. The `convertAlpha()` function works similarly to `convertSide()`. It scales the output values of a given function, thus it may be used to create an $\alpha$-cut generator conveniently.

2.1.3 Definition with Generating Functions Omitted: Shadowed Sets

In the above examples either side generating functions or $\alpha$-cut generators were passed to the `FuzzyNumber()` function. Let us note what will happen if we omit both of them.

```
A3 <- FuzzyNumber(1, 2, 4, 5)
A3
## Fuzzy number with:
## support=[1,5],
## core=[2,4].
```

The object seems to be defined correctly: R does not make any complaints. However...

```
plot(A3)
```

It turns out that we have obtained a shadowed set! Indeed, this behavior is quite reasonable: we have provided no information on the “partial knowledge” part of our fuzzy number. In fact, the object has been initialized with generating functions always returning `NA` (Not-Available or any value). Does it mean that when we define a FN solely by side generators, we cannot compute its $\alpha$-cuts? Indeed!
Another example: evaluation of the membership function.

evaluate(A1, 6.5)  # A1 has side generators defined
## 6.5
## 0.1666667
evaluate(A2, 6.5)  # A2 hasn't got them
## 6.5
## NA

2.2 Using Numeric Approximations of \( \alpha \)-cut or Side Generators

The reason for setting `NA` as return values of omitted generators is simple. Finding a function inverse numerically requires lengthy computations and is always done locally (for a given point, not for “whole” the function at once). R is not a symbolic mathematical solver. If we had defined such procedures (it is really easy to do by using the `uniroot()` function), then an inexperienced user would have used it in his/her algorithms and wondered why everything runs so slow. To get more insight, let us look at the internals of `A2`:

A2['lower']
## function(alpha) pbeta(alpha, 5, 9)
## <bytecode: 0x5613b87afd60>
A2['upper']
## function(alpha) pexp(1/alpha-1)
## <bytecode: 0x5613b8897030>
A2['left']
## function (x)
## rep(NA_real_, length(x))
## <environment: 0x5613b6633060>
A2['right']
## function (x)
## rep(NA_real_, length(x))
## <environment: 0x5613b6633060>

Note that all generators are properly vectorized (for input vectors of length \( n \) they always give output of the same length). Thus, general rules are as follows. If you want \( \alpha \)-cuts (e.g. for finding trapezoidal approximations of FNs), specify them. If you would like to calculate the membership function (by the way, the `plot()` function automatically detects what kind of knowledge we have), assure the side generators are provided.

However, we also provide a convenient short-cut method to interpolate generating functions of one type to get some crude numeric approximations of their inverses: the `approxInvert()` func-

To be precise, it’s `NA_real_`. 

tion, which may of course be applied on results returned by `convertAlpha()` and `convertSide()`. This is a simple wrapper to R’s `approxfun()` (piecewise linear interpolation, the ‘linear’ method) and `splinefun()` (monotonic splines: methods ‘hyman’ and ‘monoH.FC’; the latter is default and recommended).

```r
l <- function(x) pbeta(x, 1, 2)
r <- function(x) 1 - pbeta(x, 1, 0.1)
A4 <- FuzzyNumber(-2, 0, 0, 2,
    left = l,
    right = r,
    lower = approxInvert(l),
    upper = approxInvert(r))

x <- seq(0, 1, length.out=1e5)
max(abs(qbeta(x, 1, 2) - A4['lower'](x))) # sup-error estimator
## [1] 0.0001389811
max(abs(qbeta(1-x, 1, 0.1) - A4['upper'](x))) # sup-error estimator
## [1] 0.0008607773
```

## 2.3 Trapezoidal Fuzzy Numbers

A trapezoidal fuzzy number (TFN) is a FN which has linear side generators and linear \( \alpha \)-cut bound generators. To create a trapezoidal fuzzy number \( T_1 \) with, for example, core\( (T_1) = [1.5, 4] \) and \( \text{supp}(T_1) = [1, 7] \) we call:

```r
T1 <- TrapezoidalFuzzyNumber(1, 1.5, 4, 7)
```

Thus, we have:

\[
\mu_{T_1}(x) = \begin{cases} 
0 & \text{for } x \in (-\infty, 1], \\
(x - 1)/0.5 & \text{for } x \in [1, 1.5), \\
1 & \text{for } x \in [1.5, 4], \\
(7 - x)/3 & \text{for } x \in (4, 7], \\
0 & \text{for } x \in (7, +\infty). 
\end{cases}
\]

\[ T_{1\alpha} = [1 + 0.5 \alpha, 7 - 3 \alpha]. \]

Note that the above equations have been automatically generated by `knitr` and `\LaTeX` by calling `cat(as.character(T1, toLaTeX=TRUE, varnameLaTeX='T_1'), see Sec. 3`.

The \( T_1 \) object is an instance of the following R class:

```r
class(T1)
## [1] "TrapezoidalFuzzyNumber"
## attr(,"package")
## [1] "FuzzyNumbers"
```

To depict \( T_1 \) we call:

\[4The \ n \ argument, which sets the number of interpolation points, controls the trade-off between accuracy and computation speed. Well, world’s not ideal, remember that “some” is better than “nothing” sometimes.\]
T1 is (roughly) equivalent to the trapezoidal fuzzy number A₁ defined in the previous subsection. The TrapezoidalFuzzyNumber class inherits all the goodies from the FuzzyNumber class, but is more specific (guarantees faster computations, contains more detailed information, etc.). Of course, in this case the generating functions are known a priori (A₁ had no α-cut generators) so there is no need to provide them manually (what is more, this has been disallowed for safety reasons). Thus, if we wanted to define a trapezoidal FN next time, we would do it not like with A₁ but rather as with T₁.

Trapezoidal fuzzy numbers are among the simplest FNs. Despite their simplicity, however, they include triangular FNs, “crisp” real intervals, and “crisp” reals. Please note that currently no separate classes for these particular TFNs types are implemented in the package.
TrapezoidalFuzzyNumber(1,2,2,3)  # triangular FN
   ## Trapezoidal fuzzy number with:
   ## support=[1,3],
   ## core=[2,2].
TriangularFuzzyNumber(1,2,3)      # the same
   ## Trapezoidal fuzzy number with:
   ## support=[1,3],
   ## core=[2,2].
TrapezoidalFuzzyNumber(2,2,3,3) # 'crisp' interval
   ## Trapezoidal fuzzy number with:
   ## support=[2,3],
   ## core=[2,3].
as.TrapezoidalFuzzyNumber(c(2,3)) # the same
   ## Trapezoidal fuzzy number with:
   ## support=[2,3],
   ## core=[2,3].
TrapezoidalFuzzyNumber(5,5,5,5) # 'crisp' real
   ## Trapezoidal fuzzy number with:
   ## support=[5,5],
   ## core=[5,5].
as.TrapezoidalFuzzyNumber(5) # the same
   ## Trapezoidal fuzzy number with:
   ## support=[5,5],
   ## core=[5,5].

2.4 Piecewise Linear Fuzzy Numbers

Trapezoidal fuzzy numbers are generalized by piecewise linear FNs (PLFNs), i.e. fuzzy numbers which side generating functions and α-cut generators are piecewise linear functions. Each PLFN is given by:

- four coefficients $a_1 \leq a_2 \leq a_3 \leq a_4$ defining its support and core,
- the number of “knots”, $\text{knot.n} \geq 0$,
- a vector of α-cut coordinates, $\text{knot.alpha}$, consisting of $\text{knot.n}$ elements $\in [0,1]$,
- a nondecreasingly sorted vector $\text{knot.left}$ consisting of $\text{knot.n}$ elements $\in [a_1,a_2]$, defining interpolation points for the left side function, and
- a nondecreasingly sorted vector $\text{knot.right}$ consisting of $\text{knot.n}$ elements $\in [a_2,a_3]$, defining interpolation points for the right side function.

If $\text{knot.n} \geq 1$, then the membership function of a piecewise linear fuzzy number $P$ is defined as:
\[ \mu_P(x) = \begin{cases} 
0 & \text{if } x < a_1, \\
\alpha_i + (\alpha_{i+1} - \alpha_i) \left( \frac{x-l_i}{l_{i+1}-l_i} \right) & \text{if } l_i \leq x < l_{i+1} \\
1 & \text{for some } i \in \{1, \ldots, n+1\}, \\
\alpha_{n-i+2} + (\alpha_{n-i+3} - \alpha_{n-i+2}) \left( 1 - \frac{x-r_i}{r_{i+1}-r_i} \right) & \text{if } a_2 \leq x \leq a_3, \\
0 & \text{for some } i \in \{1, \ldots, n+1\}, \\
0 & \text{if } a_4 < x, 
\end{cases} \]

and its \( \alpha \)-cuts for \( \alpha \in [\alpha_i, \alpha_{i+1}] \) (for some \( i \in \{1, \ldots, n+1\} \)) are given by:

\[
P_L(\alpha) = l_i + (l_{i+1} - l_i) \left( \frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i} \right), \tag{7}
\]

\[
P_U(\alpha) = r_{n-i+2} + (r_{n-i+3} - r_{n-i+2}) \left( 1 - \frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i} \right), \tag{8}
\]

where \( n = \text{knot.n, (l_1, \ldots, l_{n+2}) = (a_1, \text{knot.left}, a_2), (r_1, \ldots, r_{n+2}) = (a_3, \text{knot.right}, a_4)}, \) and \( (\alpha_1, \ldots, \alpha_{n+2}) = (0, \text{knot.alpha}, 1) \).

PLFNs in our package are represented by the \texttt{PiecewiseLinearFuzzyNumber} class.

```r
P1 <- PiecewiseLinearFuzzyNumber(1, 2, 3, 4,
    knot.n=1, knot.alpha=0.25, knot.left=1.5, knot.right=3.25)
class(P1)
## [1] "PiecewiseLinearFuzzyNumber"
## attr(,"package")
## [1] "FuzzyNumbers"
P1
## Piecewise linear fuzzy number with 1 knot(s),
##   support=[1,4],
##   core=[2,3].

P2 <- PiecewiseLinearFuzzyNumber(1, 2, 3, 4,
    knot.n=2, knot.alpha=c(0.25,0.6),
    knot.left=c(1.5,1.8), knot.right=c(3.25, 3.5))
P2
## Piecewise linear fuzzy number with 2 knot(s),
##   support=[1,4],
##   core=[2,3].

plot(P1, type='b', from=0, to=5, xlim=c(0.5,4.5))
plot(P2, type='b', col=2, lty=2, pch=2, add=TRUE, from=0, to=5)
```
The following operators return matrices with all knots of a PLFN. Each matrix has three columns: $\alpha$-cuts, left side coordinates, and right side coordinates.

\[
P_1['knots']
\]

# alpha L U
# knot_1 0.25 1.5 3.25

\[
P_1['allknots'] # including a1,a2,a3,a4
\]

# alpha L U
# supp 0.00 1.0 4.00
# knot_1 0.25 1.5 3.25
# core 1.00 2.0 3.00

We have, for example:

\[
\mu_{P_1}(x) = \begin{cases} 
0 & \text{for } x \in (-\infty, 1), \\
0 + 0.25 (x + 1)/0.5 & \text{for } x \in [1, 1.5), \\
0.25 + 0.75 (x + 1.5)/0.5 & \text{for } x \in [1.5, 2), \\
1 & \text{for } x \in [2, 3], \\
0.25 + 0.75 (3.25 - x)/0.25 & \text{for } x \in [3, 3.25), \\
0 + 0.25 (4 - x)/0.75 & \text{for } x \in [3.25, 4), \\
0 & \text{for } x \in (4, +\infty). 
\end{cases}
\]

\[
P_{1\alpha} = [P_{1L}(\alpha), P_{1U}(\alpha)],
\]

where

\[
P_{1L}(\alpha) = \begin{cases} 
1 + 0.5 (\alpha - 0)/0.25 & \text{for } \alpha \in [0, 0.25], \\
1.5 + 0.5 (\alpha - 0.25)/0.75 & \text{for } \alpha \in [0.25, 1], 
\end{cases}
\]

\[
P_{1U}(\alpha) = \begin{cases} 
3.25 + 0.75 (0.25 - \alpha)/0.25 & \text{for } \alpha \in [0, 0.25], \\
3 + 0.25 (1 - \alpha)/0.75 & \text{for } \alpha \in [0.25, 1]. 
\end{cases}
\]

If you want to obtain a PLFN with equally distributed knots, then you may use the more convenient version of the \texttt{PiecewiseLinearFuzzyNumber()} function.

\[
\texttt{PiecewiseLinearFuzzyNumber(knot.left=c(0,0.5,0.7,1),}
\]

\[
\texttt{knot.right=c(2,2.2,2.4,3))['allknots']}\]
Note that if \( a_1, \ldots, a_4 \) are omitted, then they are taken from \( \text{knot.left} \) and \( \text{knot.right} \) (their lengths should then be equal to \( \text{knot.n+2} \)).

If \( \text{knot.n} \) is equal to 0 or all left and right knots lie on common lines, then a PLFN reduces to a TFN. Please note that, however, the \text{TrapezoidalFuzzyNumber} class does not inherit from \text{PiecewiseLinearFuzzyNumber} for efficiency reasons. If, however, we wanted to convert an object of the first mentioned class to the other, we would do that by calling:

```r
alpha <- c(0.3, 0.5, 0.7)
P3 <- as.PiecewiseLinearFuzzyNumber(
  TrapezoidalFuzzyNumber(1,2.5,4,7),
  knot.n=3, knot.alpha=alpha
)
P3
```

More generally, each PLFN or TFN may be converted to a direct \text{FuzzyNumber} class instance if needed (hope we will never not).

```r
(as.FuzzyNumber(P3))
```

On the other hand, to “convert” (with possible information loss) more general FNs to TFNs or PLFNs, we may use the approximation procedures described in Sec. 6.
2.5 Fuzzy Numbers with Sides Given by Power Functions

Bodjanova-type fuzzy numbers which sides are given by power functions are defined using four coefficients \( a_1 \leq a_2 \leq a_3 \leq a_4 \), and parameters \( p_{\text{left}}, p_{\text{right}} > 0 \) which determine exponents for the side functions:

\[
\begin{align*}
\text{left}(x) &= x^{p_{\text{left}},} \\
\text{right}(x) &= (1 - x)^{p_{\text{right}}}.
\end{align*}
\]

We also have:

\[
\begin{align*}
\text{lower}(\alpha) &= p_{\text{left}}^{\sqrt[2]{\alpha}}, \\
\text{upper}(\alpha) &= 1 - p_{\text{right}}^{\sqrt[2]{\alpha}}.
\end{align*}
\]

These fuzzy numbers are another natural generalization of trapezoidal FNs.

An example:

```r
X <- PowerFuzzyNumber(-3, -1, 1, 3, p.left=2, p.right=0.1)
class(X)
## [1] "PowerFuzzyNumber"
## attr(,"package")
## [1] "FuzzyNumbers"
X
## Fuzzy number given by power functions, and:
## support=[-3,3],
## core=[-1,1].
plot(X)
```

We have:

\[
\mu_X(x) = \begin{cases} 
0 & \text{for } x \in (-\infty, -3), \\
(x + 3)^2 & \text{for } x \in [-3, -1), \\
1 & \text{for } x \in [-1, 1], \\
((3 - x)^2)^{0.1} & \text{for } x \in (1, 3], \\
0 & \text{for } x \in (3, +\infty),
\end{cases}
\]

\[
X_\alpha = [-3 + 2 \alpha^{0.5}, 1 + 2 (1 - \alpha^{10})].
\]
3 Depicting Fuzzy Numbers

To draw FNs we call the `plot()` method, which uses similar parameters as the R-built-in `curve()` function / `plot.default()` method. If you are new to R, you may wish to read the manual on the most popular graphical routines by calling `?plot`, `?plot.default`, `?curve`, `?abline`, `?par`, `?lines`, `?points`, `?legend`, `?text` (some of these functions have already been called in this tutorial).

Let us consider the following FN:

```r
A <- FuzzyNumber(-5, 3, 6, 20,
  left=function(x) pbeta(x, 0.4, 3),
  right=function(x) 1-x^(1/4),
  lower=function(alpha) qbeta(alpha, 0.4, 3),
  upper=function(alpha) (1-alpha)^4
)
plot(A)
```

![Plot of fuzzy number A](image)

**Plotting issues: discretization.** Side functions or \( \alpha \)-cut bounds of objects of the `FuzzyNumber` class (not including its derivatives) when plotted are naively approximated by piecewise linear functions with equidistant knots at one of the axes. Therefore, if we probe them at too few points, we may obtain very rough graphical representations. To control the number of points at which the interpolation takes place, we use the `n` argument (which defaults to 101, i.e. “quite accurate”).

All three calls to the `plot()` method below depict the membership function of the same fuzzy number, but with different accuracy.

```r
plot(A, n=3, type='b')
plot(A, n=6, add=TRUE, lty=2, col=2, type='b', pch=2)
plot(A, n=101, add=TRUE, lty=4, col=4) # default n
```
Making use of different generating functions’ types. Please note (if you have not already) that to draw the membership function we do not need to provide necessarily the FN with side generators: the $\alpha$-cuts will also suffice. The function is smart enough to detect the internal representation of the FN and use the kind representation it has. It both types of generators are given, then side functions are used. If we want, for some reasons, to use $\alpha$-cuts, then we may do as follows:

```r
plot(A, n=3, at.alpha=numeric(0), type='b') # use alpha-cuts
plot(A, n=3, type='b', col=2, lty=2, pch=2, add=TRUE) # use sides
```

We may also illustrate an $\alpha$-cut representation of a fuzzy number:

```r
plot(A, draw.alphacuts=TRUE)
```
Exporting figures. If we would like to generate figures for our publications, then we will surely be interested in storing them e.g. as PDF files. This may be done by calling:

```r
pdf('figure1.pdf', width=8, height=5) # create file
plot(A)
dev.off() # close graphical device and save the file
```

Postscript (PS) files are generated by substituting the call to `pdf()` for the call to the `postscript()` function.

Conversion to \LaTeX. Another way to depict a FN is to... give a mathematical expression which defines it.

```r
cat(as.character(A, toLaTeX=TRUE, varnameLaTeX='A'))
```

This gives the following \LaTeX code...

\[
\mu_A(x) = \begin{array}{lll}
0 & \text{for} & x \in (-\infty,-5), \\
{\left}_A((x+5)/8) & \text{for} & x \in [-5,3), \\
1 & \text{for} & x \in [3,6], \\
{\right}_A((x-6)/14) & \text{for} & x \in (6,20], \\
0 & \text{for} & x \in (20,\infty), \\
\end{array}
\]

where $l_A=\texttt{left}_A((x+5)/8)$, $r_A=\texttt{right}_A((x-6)/14)$.

\[
{A}_\alpha = [{A}_L(\alpha), {A}_U(\alpha)],
\]

where $A_L(\alpha)=-5+8/\texttt{lower}_A(\alpha)$, $A_U(\alpha)=6+14/\texttt{upper}_A(\alpha)$.

...and, after compiling:
\[
\mu_A(x) = \begin{cases} 
0 & \text{for } x \in (-\infty, -5), \\
l_A(x) & \text{for } x \in [-5, 3), \\
1 & \text{for } x \in [3, 6], \\
r_A(x) & \text{for } x \in (6, 20], \\
0 & \text{for } x \in (20, +\infty), 
\end{cases}
\]

where \( l_A = \text{left}_A((x + 5)/8) \), \( r_A = \text{right}_A((x - 6)/14) \).

\[
A_\alpha = [A_L(\alpha), A_U(\alpha)],
\]

where \( A_L(\alpha) = -5 + 8 \text{lower}_A(\alpha) \), \( A_U(\alpha) = 6 + 14 \text{upper}_A(\alpha) \).

The code may of course be modified manually to suit your needs.

**Tuning your figures.** Finally, we leave you with a quite more complex graphical example from one of our papers:

```r
X <- PiecewiseLinearFuzzyNumber(0, 1, 2, 5, knot.n=1, knot.alpha=0.6, knot.left=0.3, knot.right=4)
plot.default(NA, xlab=expression(x), ylab=expression(mu[S](x)), xlim=c(-0.3,5.3), ylim=c(0,1)) # empty window
xpos <- c(X['a1'], X['knot.left'], X['a2'], X['a3'], X['knot.right'], X['a4'])
xlab <- expression(s[1], s[2], s[3], s[4], s[5], s[6])
abline(v=xpos, col='gray', lty=3)
text(xpos, 1.05, xlab, pos=3, xpd=TRUE)
abline(h=c(0, X['knot.alpha'], 1), col='gray', lty=2)
text(5.1, X['knot.alpha'], expression(alpha[0]), pos=4, xpd=TRUE)
plot(X, add=TRUE, type='l', from=-1, to=6)
plot(X, add=TRUE, type='p', from=-1, to=6)
```

![Graphical example](image)

4 Basic Computations on and Characteristics of Fuzzy Numbers

In this section we consider the following FN:
A <- FuzzyNumber(-5, 3, 6, 20,
    left=function(x) pbeta(x,0.4,3),
    right=function(x) 1-x^((1/4)),
    lower=function(alpha) qbeta(alpha,0.4,3),
    upper=function(alpha) (1-alpha)^4)

4.1 Support and Core, and Other $\alpha$-cuts

The support of $A$, i.e. $\text{supp}(A) = [a_1, a_4]$, may be obtained by calling:

```
supp(A)
## [1] -5 20
```

We get the core of $A$, i.e. $\text{core}(A) = [a_2, a_3]$, with:

```
core(A)
## [1] 3 6
```

To compute arbitrary $\alpha$-cuts we use:

```
alphacut(A, 0) # same as supp(A) (if alpha-cut generators are defined)
## L U
## 0 -5 20

alphacut(A, 1) # same as core(A)
## L U
## 1 3 6

(a <- alphacut(A, c(0, 0.5, 1)))
## L U
## 0.0 -5.000000 20.0000
## 0.5 -4.583591 6.8750
## 1.0 3.000000 6.0000

a[1,]
## L U
## -5 20

a[2,2]
## [1] 6.875

a[, "L"]
## 0.0 0.5 1.0
## -5.000000 -4.583591 3.000000
```

Note that `alphacut()` always outputs a matrix with two columns. The matrix has named dimensions (names stand for only auxiliary information). The `alphacut()` method may only be used when $\alpha$-cut generators are provided by the user during the declaration of $A$, even for $\alpha = 0$ or $\alpha = 1$.

4.2 Membership Function Evaluation

If side generators are defined, we may calculate the values of the membership function at different points by calling:
evaluate(A, 1)
## 0.9960291

evaluate(A, c(-3, 0, 3))
## -3 0 3
## 0.8371139 0.9855322 1.0000000

evaluate(A, seq(-1, 2, by=0.5))
## -1.0 -0.5 0.0 0.5 1.0 1.5 2.0
## 0.9624800 0.9760168 0.9855322 0.9919531 0.9960291 0.9983815 0.9995357

4.3 “Typical” Value

Let us first introduce the notion of the expected interval of \( A \) [13].

\[
\begin{align*}
\text{EI}(A) &:= [\text{EI}_L(A), \text{EI}_U(A)] \\
&= \left[ \int_0^1 A_L(\alpha) d\alpha, \int_0^1 A_U(\alpha) d\alpha \right].
\end{align*}
\]  

To compute the expected interval of \( A \) we call:

\[
\text{expectedInterval}(A)
\]

## [1] -4.058824 8.800000

In case of objects of the \texttt{FuzzyNumber} class, the expected interval is approximated by numerical integration. This method calls the \texttt{integrate()} function and its accuracy (quite fine by default) may be controlled by the \texttt{subdivisions}, \texttt{rel.tol}, and \texttt{abs.tol} parameters (call \texttt{?integrate} for more details). On the other hand, for e.g. TFNs and PLFs this method returns exact results.

The midpoint of the expected interval is called the expected value of a fuzzy number. It is given by:

\[
\text{EV}(A) := \frac{\text{EI}_L(A) + \text{EI}_U(A)}{2}.
\]  

Let us calculate \( \text{EV}(A) \).

\[
\text{expectedValue}(A)
\]

## [1] 2.370588

Note that this method uses a call to \texttt{expectedInterval}(A), thus in case of \texttt{FuzzyNumber} class instances it also uses numerical approximation.

Sometimes a generalization of the expected value, called weighted expected value, is useful. For given \( w \in [0, 1] \) it is defined as:

\[
\text{EV}_w(A) := (1 - w)\text{EI}_L(A) + w\text{EI}_U(A).
\]  

It is easily seen that \( \text{EV}_{0.5}(A) = \text{EV}(A) \).

Some examples:

\[
\text{weightedExpectedValue}(A, 0.5) \quad \text{# equivalent to expectedValue(A)}
\]

## [1] 2.370588

\[
\text{weightedExpectedValue}(A, 0.25)
\]

## [1] -0.8441176
The value of $A$ is defined by:

$$
\text{val}(A) := \int_0^1 \alpha (A_L(\alpha) + A_U(\alpha)) \, d\alpha.
$$

(17)

It may be calculated by calling:

```
value(\$a\$
```

## [1] 1.736177

Please note that the expected value or value may be used for example to “defuzzify” $A$.

### 4.4 Measures of “Nonspecificity”

The width of $A$ is defined as:

$$
\text{width}(A) := EI_U(A) - EI_L(A).
$$

(18)

An example:

```
width(\$a\$
```

## [1] 12.85882

The ambiguity of $A$ is defined as:

$$
\text{amb}(A) := \int_0^1 \alpha (A_U(\alpha) - A_L(\alpha)) \, d\alpha.
$$

(19)

```
ambiguity(\$a\$
```

## [1] 5.197157

Additionally, to express “nonspecificity” of a fuzzy number we may use e.g. the width of its support:

```
diff(supp(\$a\$
```

## [1] 25

### 5 Operations on Fuzzy Numbers

#### 5.1 Arithmetic Operations

The basic binary arithmetic operations for FNs are often defined by means of the so-called extension principle (see [20]) and interval arithmetic. For each $\alpha \in [0, 1]$:

$$(A \odot B)_\alpha = A_\alpha \odot B_\alpha,$$

where $\odot = +, -, \times$ or $/$, and $A, B$ are arbitrary FNs.

For example, we define the sum $A + B$ for every $\alpha \in [0, 1]$ as:

$$(A + B)_\alpha = A_\alpha + B_\alpha = [A_L(\alpha) + B_L(\alpha), A_U(\alpha) + B_U(\alpha)],$$

see [10, 9]. Moreover, for $\lambda \in \mathbb{R}$, the scalar multiplication is given by:

$$(\lambda \cdot A)_\alpha = \lambda A_\alpha = \begin{cases} 
[\lambda A_L(\alpha), \lambda A_U(\alpha)], & \text{if } \lambda \geq 0, \\
[\lambda A_U(\alpha), \lambda A_L(\alpha)], & \text{if } \lambda < 0,
\end{cases}$$
for each $\alpha \in [0, 1]$.

In the FuzzyNumbers package we have defined the $+$, $-$, $\ast$ and $/$ operators, which implements the basic arithmetic operations as defined in [20].

```r
A <- TrapezoidalFuzzyNumber(0, 1, 1, 2)
B <- TrapezoidalFuzzyNumber(1, 2, 2, 3)
plot(A, xlim=c(0,6))
plot(B, add=TRUE, col=2, lty=2)
plot(A+B, add=TRUE, col=4, lty=4)
```

Currently all the operations are available for piecewise linear FNs only, and addition and scalar multiplication is also implemented for trapezoidal FNs. Note that the computer arithmetic has anyway a discrete nature, and a PLFN with large number of knots often approximates (cf. Sec. 6) an arbitrary FN sufficiently well. The computations are always exact (well, up to the computer floating-point arithmetic errors) at knots.

In theory the class of PLFNs is not closed under the operations $\ast$ and $/$. However, if you operate on a large number of knots, the results should be satisfactory.

```r
A <- piecewiseLinearApproximation(PowerFuzzyNumber(1,2,3,4, p.left=2, p.right=0.5),
    method="Naive", knot.n=20)
B <- piecewiseLinearApproximation(PowerFuzzyNumber(2,3,4,5, p.left=0.1, p.right=3),
    method="Naive", knot.n=40)
A+A # the same as 2*A
## Piecewise linear fuzzy number with 20 knot(s),
##   support=[2,8],
##   core=[4,6].
A+B # note the number of knots has increased
## Piecewise linear fuzzy number with 60 knot(s),
##   support=[3,9],
##   core=[5,7].
```

### 5.2 Applying Functions

To apply a monotonic transformation on a piecewise linear fuzzy number (using the extension principle) we call `fapply()`.

```r
A <- as.PiecewiseLinearFuzzyNumber(TrapezoidalFuzzyNumber(0,1,2,3), knot.n=100)
plot(fapply(A, function(x) sqrt(log(x+1))))
```
The operation being applied should be a properly vectorized R function object.

### 5.2.1 Special Functions

There are several functions that are not monotonic but useful for calculation. Amongst those integer powers of fuzzy numbers are probably the best example. While the odd powers could be implemented as monotonic functions it can not be done for even powers if the fuzzy number contains 0. Because of this issue the power is implemented as special function.

\[ A <- \text{as.PiecewiseLinearFuzzyNumber}(\text{TrapezoidalFuzzyNumber}(-2,-1,-1,2), \text{knot.n}=10) \]

\[
\text{plot}(A, \text{xlim}=c(-8,8)) \\
\text{plot}(A^2, \text{add=}TRUE, \text{col}=2, \text{lty}=2) \\
\text{plot}(A^3, \text{add=}TRUE, \text{col}=4, \text{lty}=4)
\]

### 6 Approximation of Fuzzy Numbers

Complicated membership functions are often very inconvenient for processing imprecise information modeled by fuzzy numbers. Moreover, handling too complex membership functions entails difficulties in interpretation of the results too. This is the reason why a suitable approximation of fuzzy numbers is so important. We would like to deal with functions that are simpler or more regular and hence more convenient for computing.
6.1 Metrics in the Space of Fuzzy Numbers

It seems that the most suitable metric for approximation problems is an extension of the Euclidean ($L_2$) distance (cf. [10]), $d$, defined by the equation:

$$d^2_E(A, B) = \int_0^1 (A_L(\alpha) - B_L(\alpha))^2 \, d\alpha + \int_0^1 (A_U(\alpha) - B_U(\alpha))^2 \, d\alpha.$$  

(20)

The following metric types are currently available in the distance() method: "Euclidean" (default), "EuclideanSquared".

T1 <- TrapezoidalFuzzyNumber(-5, 3, 6, 20)
T2 <- TrapezoidalFuzzyNumber(-4, 4, 7, 21)
distance(T1, T2, type='Euclidean') # L2 distance /default/
## [1] 1.414214
distance(T1, T2, type='EuclideanSquared') # Squared L2 distance
## [1] 2

6.2 Approximation by Trapezoidal Fuzzy Numbers

Our main task in this section is to, given a fuzzy number $A$, seek for a trapezoidal fuzzy number $T(A)$ that fulfills some desired properties. We will use the following FN for the sake of illustration:

A <- FuzzyNumber(-5, 3, 6, 20,
left=function(x) pbeta(x, 0.4, 3),
right=function(x) 1-x^(1/4),
lower=function(alpha) qbeta(alpha, 0.4, 3),
upper=function(alpha) (1-alpha)^4 )

The approximation procedure has been implemented in trapezoidalApproximation(). The method argument selects the algorithm used to project $A$ into the space of TFNs.

6.2.1 Naïve Approximation

The "Naive" method just generates a trapezoidal FN with the same core and support as $A$.

(T1 <- trapezoidalApproximation(A, method='Naive'))
## Trapezoidal fuzzy number with:
## support=[-5,20],
## core=[3,6].
distance(A, T1)
## [1] 5.761482
It is easily seen that the naïve approximator may not represent \( A \) well. Thus, we will often need some more reasonable approach.

### 6.2.2 \( L_2 \)-nearest Approximation

The "NearestEuclidean" method gives the nearest \( L_2 \)-approximation of \( A \) \cite{2} Corollary 8, i.e. a trapezoidal fuzzy number \( T(A) \) such that

\[
T(A) = \min_{T \in \text{TFN}} d_E(A, T).
\]

It may be shown that the solution to this problem always exists and is unique.

```r
(T2 <- trapezoidalApproximation(A, method='NearestEuclidean'))
```

## Trapezoidal fuzzy number with:
## support=[-5.85235,14.4],
## core=[-2.26529,3.2].

distance(A, T2)
## [1] 1.98043

Note that the implementation relies on numeric integration.
6.2.3 Expected Interval Preserving Approximation

The "ExpectedIntervalPreserving" method gives the nearest $L_2$-approximation of $A$ preserving the expected interval $[1, 17, 22]$, i.e. we get $T(A)$ such that $EI(A) = EI(T(A))$.

First of all, it may be shown that if $\text{amb}(A) \geq \text{width}(A)/3$, then we obtain the same result as in the "NearestEuclidean" method.

```r
ambiguity(A)
## [1] 5.197157
width(A)/3
## [1] 4.286275
(T3 <- trapezoidalApproximation(A, method='ExpectedIntervalPreserving'))
## Trapezoidal fuzzy number with:
## support=[-5.85235,14.4],
## core=[-2.26529,3.2].
distance(A, T3)
## [1] 1.98043
expectedInterval(A)
## [1] -4.058824 8.800000
expectedInterval(T3)
## [1] -4.058824 8.800000
```

On the other hand, for highly skewed membership functions this method (as well as the previous one) sometimes reveals quite unfavorable behavior. E.g. if $B$ is a FN such that $\text{val}(B) < EV_{1/3}(B)$ or $\text{val}(B) > EV_{2/3}(B)$, then it may happen that the cores of the output and of the original fuzzy number $B$ are disjoint, cf. [18].

```r
(B <- FuzzyNumber(1, 2, 3, 45,
   lover=function(x) sqrt(x),
   upper=function(x) 1-sqrt(x)))
## Fuzzy number with:
## support=[1,45],
## core=[2,3].
(TB1 <- trapezoidalApproximation(B, 'NearestEuclidean'))
## Trapezoidal fuzzy number with:
## support=[1.37333,33.2133],
## core=[1.37333,1.37333].
(TB2 <- trapezoidalApproximation(B, 'ExpectedIntervalPreserving'))
## Trapezoidal fuzzy number with:
## support=[1.66667,32.3333],
## core=[1.66667,1.66667].
distance(B, TB1)
## [1] 2.098994
distance(B, TB2)
## [1] 2.166239
```
6.2.4 Approximation with Restrictions on Support and Core

The "SupportCoreRestricted" method was proposed in [18]. It gives the $L_2$-nearest trapezoidal approximation $T(A)$ with constraints: $\text{core}(A) \subseteq \text{core}(T(A))$ and $\text{supp}(T(A)) \subseteq \text{supp}(A)$, i.e. for which each point that surely belongs to $A$ also belongs to $T(A)$, and each point that surely does not belong to $A$ also does not belong to $T(A)$.

```r
(T4 <- trapezoidalApproximation(A, method='SupportCoreRestricted'))
```

## Trapezoidal fuzzy number with:
## support=[-5,11.6],
## core=[-3.11765,6].

distance(A, T4)
## [1] 2.603383

6.3 Approximation by Piecewise Linear Fuzzy Numbers

When approximating arbitrary fuzzy numbers by trapezoidal ones we generally take care for the core and support of a fuzzy number (i.e. for values that surely belong or do not belong at all to the set under study), while the sides of a fuzzy number corresponding to all intermediate degrees of membership are linearized. This approach may not be suitable if we are also interested in focusing on some other degrees of uncertainty except for 0 or 1.
Thus, given a fuzzy number $A$ and a fixed $\text{knot.alpha}=\alpha$ vector, we are interested in finding a piecewise linear fuzzy number $P(A)$ that has some desirable properties.

In this subsection we will use the following fuzzy number $A$ for the sake of illustration:

```r
A <- FuzzyNumber(-5, 3, 6, 20,
    left=function(x) pbeta(x,0.4,3),
    right=function(x) 1-x^(1/4),
    lower=function(alpha) qbeta(alpha,0.4,3),
    upper=function(alpha) (1-alpha)^4)
```

The approximation procedure has been implemented in `piecewiseLinearApproximation()`. The `method` argument selects the algorithm used to project $A$ into the space of PLFNs (for given `knot.alpha`).

### 6.3.1 Naïve Approximation

The "Naive" method generates a PLFN with the same core and support as $A$ and with sides interpolating the membership function of $A$ at given $\alpha$-cuts.

```r
P1 <- piecewiseLinearApproximation(A, method='Naive',
    knot.n=1, knot.alpha=0.5)
```

```r
P1['allknots']
```

<table>
<thead>
<tr>
<th></th>
<th>alpha</th>
<th>L</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>supp</td>
<td>0.0 -5.000000 20.000</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>knot_1</td>
<td>0.5 -4.583591 6.875</td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>core</td>
<td>1.0 3.000000 6.000</td>
<td></td>
</tr>
</tbody>
</table>

```r
print(distance(A, P1), 8)
```

```
## [1] 2.4753305
```

The approximation error may be quite high. However, it may be shown that e.g. for equidistant knots if $\text{knot.n} \to \infty$, then it approaches 0.

### 6.3.2 $L_2$-nearest Approximation

Similarly to the $L_2$-nearest TFN case, here we are looking for

$$P(A) = \min_{T \in \text{PLFN}(\alpha)} d_E(A, T).$$
It may be shown that the solution to this problem always exists and is unique, see [5] and [6]. The "NearestEuclidean" method uses the algorithm described in [5] and [6]. This implementation relies on numeric integration, so for large knot.n may be slow.

```r
P2 <- piecewiseLinearApproximation(A, method="NearestEuclidean", knot.n=3, knot.alpha=c(0.25,0.5,0.75))
print(P2['allknots'], 6)
## alpha L U
## supp 0.00 -5.003841 19.22964
## knot_1 0.25 -4.966165 9.91416
## knot_2 0.50 -4.578596 6.66686
## knot_3 0.75 -3.941608 6.00278
## core 1.00 -0.494012 6.00278
```

Example: Convergence. As the naïve approximator’s error approaches 0 as knot.n → ∞ for equidistant knots, so does the error of the nearest $L_2$ approximator. Let us study the convergence behavior for our exemplary $A$.

```r
n <- 1:27
d <- matrix(NA, ncol=4, nrow=length(n))
# d[,1] - Naive approximator's error for given knot.n
# d[,2] - Best L2 approximator's error
# d[,3] - theoretical upper bound
for (i in seq_along(n)) {
  P1 <- piecewiseLinearApproximation(A, method='Naive', knot.n=n[i]) # equidistant knots
  P2 <- piecewiseLinearApproximation(A, method='NearestEuclidean', knot.n=n[i]) # equidistant knots
  d[i,1] <- distance(A, P1)
  d[i,2] <- distance(A, P2)
  acut <- alphacut(A, seq(0, 1, length.out=n[i]+2))
  # d[i,3] <- sqrt(sum((c(diff(acut[,1]), diff(acut[,2]))^2)/(n[i]+1))) # beter ubound
  d[i,3] <- sqrt(2)*max(abs(c(diff(acut[,1]), diff(acut[,2]))))
}
```
Example: Finding best knot.alpha for knot.n = 1 numerically. The approximation problem is stated using a fixed knot.alpha. However, we may e.g. depict the “best” $L_2$ distance as a function of $\alpha$, i.e. the $D_A(\alpha)$ function.

```r
a <- seq(1e-9, 1-1e-9, length.out=100) # many alphas from (0,1)
d <- numeric(length(a)) # distances /to be calculated/
for (i in seq_along(a))
{
  P1 <- piecewiseLinearApproximation(A, method='NearestEuclidean',
                                   knot.n=1, knot.alpha=a[i])
  d[i] <- distance(A, P1)
}
plot(a, d, type='l', xlab=expression(alpha), ylab=expression(D[A](alpha)))
```

For knot.n = 1 we may find best knot.alpha using numerical optimization. It may be shown, see [5], that the distance function $D_A(\alpha)$ is continuous, but in general the minimum is not necessarily unique.

```r
for (i in 1:5) # 5 iterations
{
```

```r
matplot(n, d, type='l', log="y", lty=c(1,2,4), col=c(1,2,4))
```
a0 <- runif(1,0,1) # random starting point
optim(a0, 
  function(a) 
  {    
P1 <- piecewiseLinearApproximation(A, method='NearestEuclidean', knot.n=1, knot.alpha=a)

    distance(A, P1)
  }, method='L-BFGS-B', lower=1e-9, upper=1-1e-9) -> res

cat(sprintf('%.9f %6g *%.9f* %.9f
', a0, res$counts[1], res$par, res$value))
}

## 0.240763798 49 *0.363080763* 0.650344833
## 0.620345437 22 *0.363035732* 0.650345318
## 0.152540508 47 *0.363090838* 0.650345217
## 0.617319523 52 *0.363076442* 0.650344626
## 0.179268566 47 *0.363078759* 0.650344741

6.3.3 \(L_2\)-nearest Approximation Preserving Support and Core

Fix \(\alpha\). The next method searches for:

\[
P(A) = \min_{T \in \text{PLFN}(\alpha)} d_E(A, T).
\]

such that \(\text{supp}(A) = \text{supp}(P(A))\) and \(\text{core}(A) = \text{core}(P(A))\), see [7].

The "SupportCorePreserving" method of the piecewiseLinearApproximation() function currently implements only the \(\text{knot.n}=1\) case.

**Example 1.** Let us consider the following FN.

```r
A <- FuzzyNumber(0, 3, 4, 5, 
  lower=function(x) qbeta(x, 2, 1),
  upper=function(x) 1-x^3
)
```

Here are its unrestricted \((P_1)\) and supp-core-restricted \((P_2)\) PLFN approximations for \(\alpha = (0.2)\).

```r
knot.alpha <- 0.2
P1 <- piecewiseLinearApproximation(A, knot.alpha=knot.alpha)
P2 <- piecewiseLinearApproximation(A, method="SupportCorePreserving", knot.alpha=knot.alpha)
distance(A, P1)
## [1] 0.1050667

print(alphacut(P1, c(0, knot.alpha, 1)))
## L U
## 0.0 0.3416417 5.0782
## 0.2 1.4633436 5.0782
## 1.0 3.0854103 4.2577

distance(A, P2)
## [1] 0.2124758

print(alphacut(P2, c(0, knot.alpha, 1)))
```

31
Let us find $\alpha_0$ for which we get the minimal approximation distance (error).

```r
D <- function(a) distance(A, piecewiseLinearApproximation(A, method="SupportCorePreserving", knot.alpha=a))
optimize(D, lower=0, upper=1)
```

```
## $minimum
## [1] 0.2989562
##
## $objective
## [1] 0.1987527
```

Note that $D(\alpha) = d_E(A, \mathcal{P}_\alpha(A))$ may not be a well-behaving function, see [7] for discussion.

### 7 Ranking Fuzzy Numbers in the Setting of Possibility Theory

There are several possible approaches towards ranking of fuzzy numbers. Dubois and Prade [11] point out that some are counterintuitive or consider only one point of view on comparing fuzzy numbers. To overcome this issue the set of ranking indices was proposed by Dubois and Prade [11]. The ranking is done within the framework of Possibility theory which means that for each
comparison operator <=, >=, <, > there is a measure of possibility and necessity assessing the truthfulness of the statement (Possibility theory is in detail described in [12]). Both measures takes values from the interval [0,1]. Value 1 means complete truthfulness of the statement, 0 means absolute untruthfulness and values in between describe partial truthfulness.

If there are two fuzzy numbers $X$ and $Y$ then the possibility ($\Pi$) and necessity ($\mathcal{N}$) of $X \geq Y$ (named exceedance in FuzzyNumbers) is given by these equations [11]:

$$
\Pi_X([Y, \infty)) = \sup_{x} \min(\mu_X(x), \sup_{y \leq x} \mu_Y(y)),
$$

(21)

$$
\mathcal{N}_X([Y, \infty)) = \inf_{x} \max(1 - \mu_X(x), \sup_{y \leq x} \mu_Y(y)).
$$

(22)

The possibility and necessity of $X > Y$ (named strict exceedance in FuzzyNumbers) are defined as [11]:

$$
\Pi_X([Y, \infty)) = \sup_{x} \min(\mu_X(x), \inf_{y \geq x} 1 - \mu_Y(y)),
$$

(23)

$$
\mathcal{N}_X([Y, \infty)) = \inf_{x} \max(1 - \mu_X(x), \inf_{y \geq x} 1 - \mu_Y(y)).
$$

(24)

Changing the $[Y, \infty)$ to $(-\infty, Y]$ etc. in these formulas will produce other four indices indicating $X \leq Y$ and $X < Y$ [11][12].

Now lets create two PiecewiseLinearFuzzyNumbers that we will be comparing.

```r
x = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(0.2, 1.0, 2.8))
y = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(0, 1.8, 2.2))
```

We can visualized them together.

```r
plot(x, col=2)
plot(y, col=4, add=TRUE)
```

![Graph](image)

The comparison if $X \geq Y$ is done using functions `possibilityExceedance()` and `necessityExceedance()`.

```r
possibilityExceedance(x,y)
# [1] 0.7777778
necessityExceedance(x,y)
# [1] 0.3846154
```

In the same way we can determine the truthfulness of statement $X > Y$ with functions `possibilityStrictExceedance()` and `necessityStrictExceedance()`.

33
The image above shows that there is no clear domination of one of the fuzzy numbers and so tell us the indices.

To assess the inverse situations \( X \leq Y \) and \( X < Y \) the following code can be used.

The results show that again there is no strict domination of one fuzzy number by another (in such case all the indices would be either 1 or 0). But the values of all indices are higher for \( X \leq Y, X < Y \) than in cases \( X \geq Y, X > Y \). Such outcome can be interpreted as \( X \) being generally smaller than \( Y \) but the result is not conclusive. The amount of uncertainty is described by these indices.

A slightly more illustrative example is provided below. Note that third and fourth line provide limits for visualization of fuzzy numbers.

```r
x = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(1.7, 2.7, 2.8), knot.n = 9)
y = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(0, 1.8, 2.2), knot.n = 9)
min = min(x@a1, y@a1)
max = max(x@a4, y@a4)
plot(x, col=2, xlim = c(min, max))
plot(y, col=4, add=TRUE)
```
8 Minimum and Maximum of Fuzzy Numbers

In the same way as a minimum and maximum of two crisp numbers can be determined, a minimum and a maximum of two fuzzy numbers can be calculated [19]. A minimum and maximum of two fuzzy numbers is also a fuzzy number.

The minimum of fuzzy numbers is defined for $\alpha$ cuts as [19]:

$$ A_\alpha \land B_\alpha = [A_L(\alpha) \land B_L(\alpha), A_U(\alpha) \land B_U(\alpha)], $$

while the maximum is defined as:

$$ A_\alpha \lor B_\alpha = [A_L(\alpha) \lor B_L(\alpha), A_U(\alpha) \lor B_U(\alpha)]. $$

The definition allows determination of minimum and maximum only for Piecewise Linear Fuzzy Numbers. As such the result is only an approximation, which may cause problems with the result is the number of knots is too small (please see examples in the package help for the functions).

```r
x = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(-4.8, -3, -1.5), knot.n = 9)
y = as.PiecewiseLinearFuzzyNumber(TriangularFuzzyNumber(-5.5, -2.5, -1.1), knot.n = 9)
min = min(x@a1, y@a1)
max = max(x@a4, y@a4)
plot(x, col=1, xlim = c(min, max))
plot(y, col=2, add=TRUE)
maxFN = maximum(x,y)
minFN = minimum(x,y)
plot(minFN, col=4)
plot(maxFN, col=6, add=TRUE)
```
Acknowledgments. This document has been generated with \LaTeX and the knitr package for R. Their authors’ wonderful work is fully appreciated. Many thanks to Przemysław Grzegorzewski, Lucian Coroianu and Pablo Villacorta Iglesias for stimulating discussion.

The contribution of Marek Gagolewski was partially supported by the European Union from resources of the European Social Fund, Project PO KL “Information technologies: Research and their interdisciplinary applications”, agreement UDA-POKL.04.01.01-00-051/10-00 (March-June 2013), and by FNP START Scholarship from the Foundation for Polish Science (2013).

Bibliography


