FuzzyNumbers Package for R: A Quick Tutorial

Marek Gągolewski
Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
Email: gagolews@ibspan.waw.pl
www.ibspan.waw.pl/∼gagolews/FuzzyNumbers/

June 7, 2012

THIS IS A PRELIMINARY VERSION OF THE TUTORIAL

Contents

Acknowledgments. This document has been generated with LaTeX, *knitr* and the `tikzDevice` package for R. Their authors’ wonderful work is fully appreciated. Many thanks also to P. Grzegorzewski and L. Coroianu for stimulating discussion.

1 Getting started

Fuzzy set theory lets us effectively and quite intuitively represent imprecise or vague information. Fuzzy numbers (FNs), introduced by Dubois and Prade in [4], form a particular subclass of fuzzy sets of the real line. They play a significant role in many important theoretical and practical considerations 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.".

R is a free, open sourced software environment for statistical computing and graphics, which includes an implementation of a very powerful and quite popular high-level language called S. It runs on all major operating systems, i.e. *Windows*, *Linux*, and *MacOS*. To install R and/or find some information on the S language please visit R Project's Homepage at www.R-project.org. Perhaps you may also wish to install *RStudio*, a convenient development environment for R. It is available at www.rsudio.org.

FuzzyNumbers is an Open Source (licensed under GNU LGPL 3) package for R$\geq$2.15 to which anyone can contribute. It has been created in order to deal with fuzzy numbers. To install its latest "official" release available on *CRAN* we type:

```r
# ... TO DO ... | NOT YET AVAILABLE ON CRAN
```

Alternatively, we may fetch its current development snapshot (perhaps unstable) from *RForge*:

```r
install.packages("FuzzyNumbers", repos = "http://R-Forge.R-project.org")
```

Fortunately, the installation process is done only once.

Each session with FuzzyNumbers should be preceded by a call to:

```r
require("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 [6]. In case of any problems, comments, or suggestions feel free to contact the author. Good luck!

2 How to create instances of fuzzy numbers

2.1 Arbitrary FNs

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

Definition by side functions. A fuzzy number A specified by side functions has a membership function of the form:

$$\mu_A(x) = \begin{cases} 0 & \text{if } \quad x < \mathtt{a1}, \\ \mathtt{left}\left(\frac{x-\mathtt{a1}}{\mathtt{a2}-\mathtt{a1}}\right) & \text{if } \mathtt{a1} \leq x < \mathtt{a2}, \\ 1 & \text{if } \mathtt{a2} \leq x \leq \mathtt{a3}, \\ \mathtt{right}\left(\frac{x-\mathtt{a3}}{\mathtt{a4}-\mathtt{a3}}\right) & \text{if } \mathtt{a3} < x \leq \mathtt{a4}, \\ 0 & \text{if } \mathtt{a4} < x, \end{cases} \tag{1}$$

where $\texttt{a1}, \texttt{a2}, \texttt{a3}, \texttt{a4} \in \mathbb{R}$, $\texttt{a1} \le \texttt{a2} \le \texttt{a3} \le \texttt{a4}$, $\texttt{left} : [0,1] \to [0,1]$ is a nondecreasing function (called *left side generator of* A), and $\texttt{right} : [0,1] \to [0,1]$ is a nonincreasing function (*right side generator of* A). In our package, it is assumed that these functions fulfill the conditions $\texttt{left}(0) = 0$, $\texttt{left}(1) = 1$, $\texttt{right}(0) = 0$, and $\texttt{right}(0) = 1$.

Please note that by using side generating functions defined on $[0,1]$ (instead of the most common approach, i.e. ordinary side functions such that $l_A(x) = \frac{x-\texttt{a1}}{\texttt{a2}-\texttt{a1}}$ and $r_A(x) = \frac{x-\texttt{a3}}{\texttt{a4}-\texttt{a3}}$) we really make (in author's humble opinion) the process of generating examples for our publications much easier.

An example: a fuzzy number A_1 with linear sides (a trapezoidal fuzzy number, see also Sec. 2.2).

```
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:

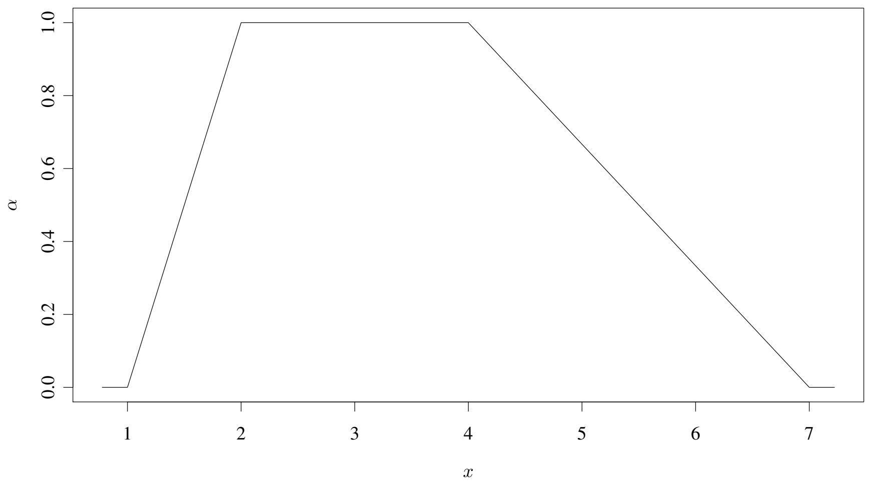
```
class(A1)

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

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

```
A1

## Fuzzy number with support=[1,7] and core=[2,4].
```

To depict A_1 we call:

```
plot(A1)
```



If you would like to use this figure in your publications, perhaps you may be interested in storing it in a PDF file on your disk (in a current working directory). This may be done by calling:

```
pdf("figure1.pdf", width=8, height=5); # create file
plot(A1);
dev.off(); # close graphical device and save the file
```

Additionally, Postscript files may be generated by substituting the call to `pdf()` for the call to the `postcript()` function.

Definition by α-cut bounds. Alternatively, a fuzzy number A may be defined by specifying its α-cuts. We have (for $\alpha \in (0,1)$ and $\mathtt{a1} \le \mathtt{a2} \le \mathtt{a3} \le \mathtt{a4}$):

$$\begin{aligned} A_\alpha &:= [A_L(\alpha), A_R(\alpha)] \qquad (2)\\ &= \left[\mathtt{a1} + (\mathtt{a2} - \mathtt{a1}) \cdot \mathtt{lower}(\alpha), \mathtt{a3} + (\mathtt{a4} - \mathtt{a3}) \cdot \mathtt{upper}(\alpha)\right], \qquad (3)\end{aligned}$$

where $\mathtt{lower} : [0,1] \to [0,1]$ is a nondecreasing function (called *lower α-cut bound generator of A*), and $\mathtt{upper} : [0,1] \to [0,1]$ is a nonincreasing function (*upper bound generator*). In our package, we assumed that $\mathtt{lower}(0) = 0$, $\mathtt{lower}(1) = 1$, $\mathtt{upper}(0) = 0$, and $\mathtt{upper}(0) = 1$.

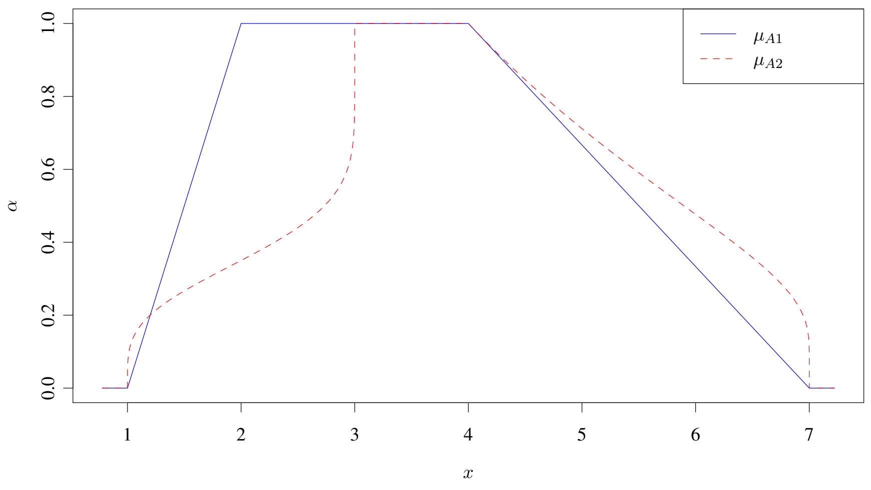
It is easily seen that for $\alpha \in (0,1)$ we have the following relationship between generating functions:

$$\begin{aligned} \mathtt{lower}(\alpha) &= \inf\{x : \mathtt{left}(x) \ge \alpha\}, \qquad (4)\\ \mathtt{upper}(\alpha) &= \sup\{x : \mathtt{right}(x) \ge \alpha\}. \qquad (5)\end{aligned}$$

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

An example:

```
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)
```

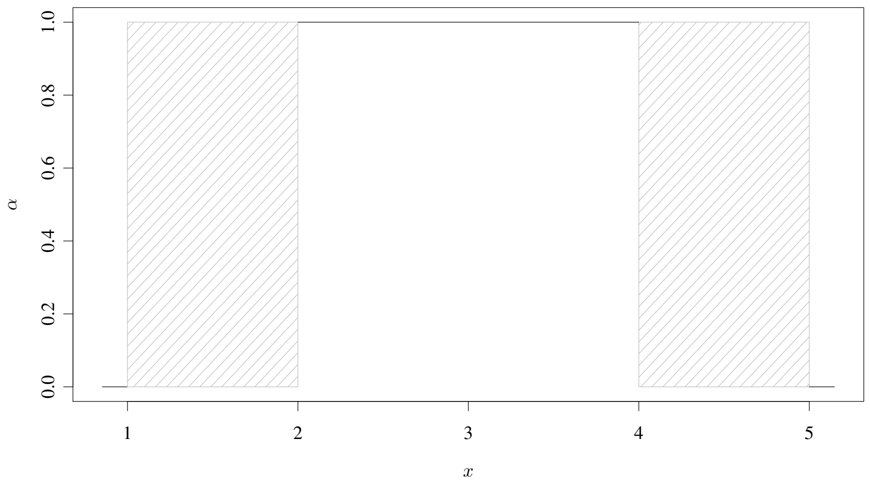


Definition with generating functions omitted. Please note that in the above examples we passed to the constructor of each `FuzzyNumber` class instance either side generating functions or α-cut generators. Let us study what happens if we omit both of them.

```
A3 <- FuzzyNumber(1, 2, 4, 5)
A3

## Fuzzy number with support=[1,5] and 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. If fact, the object has been initialized as:

```
A3 <- FuzzyNumber(1, 2, 4, 5,
           left=function(x) NA,
          right=function(x) NA,
          lower=function(x) NA,
          upper=function(x) NA
)
```

that is, with 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 α-cuts? Indeed!

```
alphacut(A2, 0.5)  # A2 has alpha-cut generators defined

## [1] 2.733 5.896

alphacut(A1, 0.5)  # A1 has not

## [1] NA NA
```

Another example: evaluation of the membership function.

```
evaluate(A1, 6.5)  # A1 has side generators defined

## [1] 0.1667

evaluate(A2, 6.5)  # A2 has not

## [1] NA
```

The reason for setting `NA` as return values of generators by default (when omitted) 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 into the internals of `A2`:

```
A2["lower"]

## function(alpha) pbeta(alpha, 5, 9)
## <environment: 0x3457138>

A2["upper"]

## function(alpha) pexp(1/alpha-1)
## <environment: 0x3457138>

A2["left"]

## function (x)
## NA
## <environment: 0x4371ef0>

A2["right"]

## function (x)
## NA
## <environment: 0x4371ef0>
```

Thus, general rules are as follows. If you want α-cuts (e.g. for finding trapezoidal approximations of FNs) — specify them. If you would like to access side functions (what for? luckily, the `plot()` function automatically detects what kind of knowledge we have) — assure they are provided.

2.2 Trapezoidal FNs

A trapezoidal fuzzy number (TFN) is a FN which has linear side generators and linear α-cut bound generators.

To create a trapezoidal fuzzy number T_1 with, for example, $\mathrm{core}(T_1) = [2, 4]$ and $\mathrm{supp}(T_1) = [1, 7]$ we call:

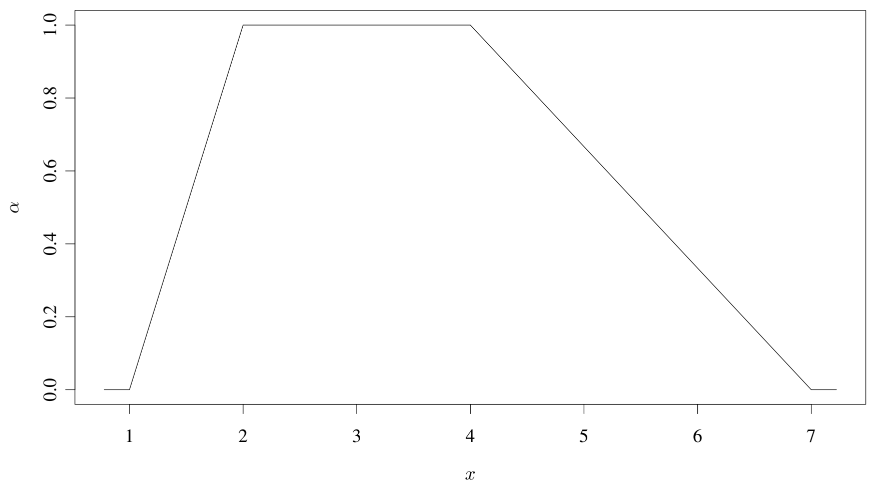
```
T1 <- TrapezoidalFuzzyNumber(1, 2, 4, 7)
```

This object is an instance of the following `R` class:

```
class(T1)

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

To depict T_1 we call:

```
plot(T1)
```



T_1 is (roughly) equivalent to the trapezoidal fuzzy number A_1 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* (A_1 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, is we wanted to define a trapezoidal FN next time, we would rather not do it like with A_1 but as with T_1.

```
T1["lower"]

## function (alpha)
## alpha
## <environment: namespace:FuzzyNumbers>

T1["upper"]

## function (alpha)
## 1 - alpha
## <environment: namespace:FuzzyNumbers>

T1["left"]

## function (x)
## x
## <environment: namespace:FuzzyNumbers>

T1["right"]

## function (x)
## 1 - x
## <environment: namespace:FuzzyNumbers>
```

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] and core=[2,2].

TrapezoidalFuzzyNumber(2, 2, 3, 3)  # crisp interval

## Trapezoidal fuzzy number with support=[2,3] and core=[2,3].

TrapezoidalFuzzyNumber(5, 5, 5, 5)  # crisp real

## Trapezoidal fuzzy number with support=[5,5] and core=[5,5].
```

2.3 Piecewise linear FNs

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 $\mathtt{a1} \le \mathtt{a2} \le \mathtt{a3} \le \mathtt{a4}$ defining its support and core,
- the number of “knots”, `knot.n`≥ 0,
- a vector of α-cut coordinates, `knot.alpha`, consisting of `knot.n` elements $\in [0, 1]$,
- a nondecreasingly sorted vector `knot.left` consisting of `knot.n` elements $\in [\mathtt{a1}, \mathtt{a2}]$, defining interpolation points for the left side function, and
- a nondecreasingly sorted vector `knot.right` consisting of `knot.n` elements $\in [\mathtt{a2}, \mathtt{a3}]$, defining interpolation points for the right side function.

If `knot.n`≥ 1 then the membership function of a piecewise linear fuzzy number P is defined as:

$$\mu_P(x) = \begin{cases} 0 & \text{if } \quad x < \mathtt{a1}, \\ \alpha_i + (\alpha_{i+1} - \alpha_i)\left(\frac{x - l_i}{l_{i+1} - l_i}\right) & \text{if } \ l_i \le x < l_{i+1} \\ & \ \text{for some } i \in \{1, \dots, n+1\}, \\ 1 & \text{if } \mathtt{a2} \le x \le \mathtt{a3}, \\ \alpha_{n-i+2} + (\alpha_{n-i+3} - \alpha_{n-i+2})\left(1 - \frac{x - r_i}{r_{i+1} - r_i}\right) & \text{if } \ r_i < x \le r_{i+1} \\ & \ \text{for some } i \in \{1, \dots, n+1\}, \\ 0 & \text{if } \mathtt{a4} < x, \end{cases} \tag{6}$$

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

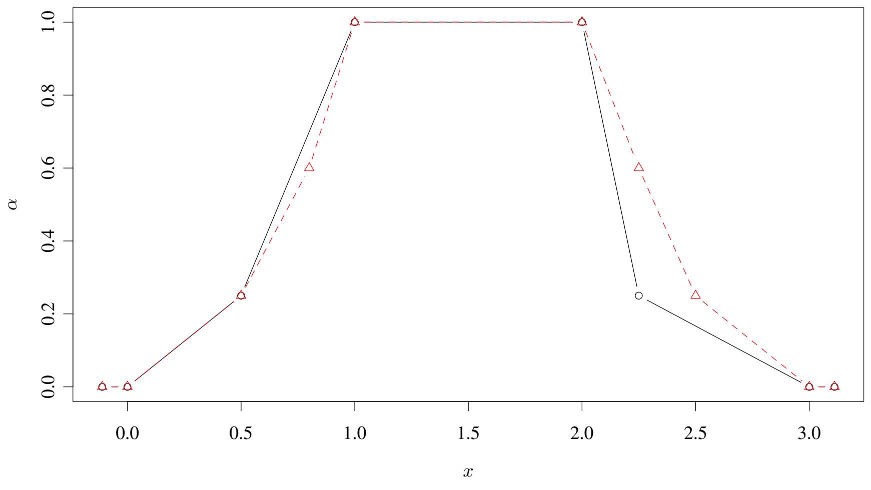
$$P_L(\alpha) = l_i + (l_{i+1} - l_i)\left(\frac{\alpha - \alpha_i}{\alpha_{i+1} - \alpha_i}\right), \tag{7}$$

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

where $n = \mathtt{knot.n}$, $(l_1, \dots, l_{n+2}) = (\mathtt{a1}, \mathtt{knot.left}, \mathtt{a2})$, $(r_1, \dots, r_{n+2}) = (\mathtt{a3}, \mathtt{knot.right}, \mathtt{a4})$, and $(\alpha_1, \dots, \alpha_{n+2}) = (0, \mathtt{knot.alpha}, 1)$.

PLFNs in our package are represented by the `PiecewiseLinearFuzzyNumber` class.

```
P1 <- PiecewiseLinearFuzzyNumber(0, 1, 2, 3,
   knot.n=1, knot.alpha=0.25, knot.left=0.5, knot.right=2.25)
P1

## Piecewise linear fuzzy number with 1 knot(s), support=[0,3] and core=[1,2].

P2 <- PiecewiseLinearFuzzyNumber(0, 1, 2, 3,
   knot.n=2, knot.alpha=c(0.25,0.6),
   knot.left=c(0.5,0.8), knot.right=c(2.25, 2.5))
P2

## Piecewise linear fuzzy number with 2 knot(s), support=[0,3] and core=[1,2].

plot(P1, type='b')
plot(P2, type='b', col=2, lty=2, pch=2, add=TRUE)
```



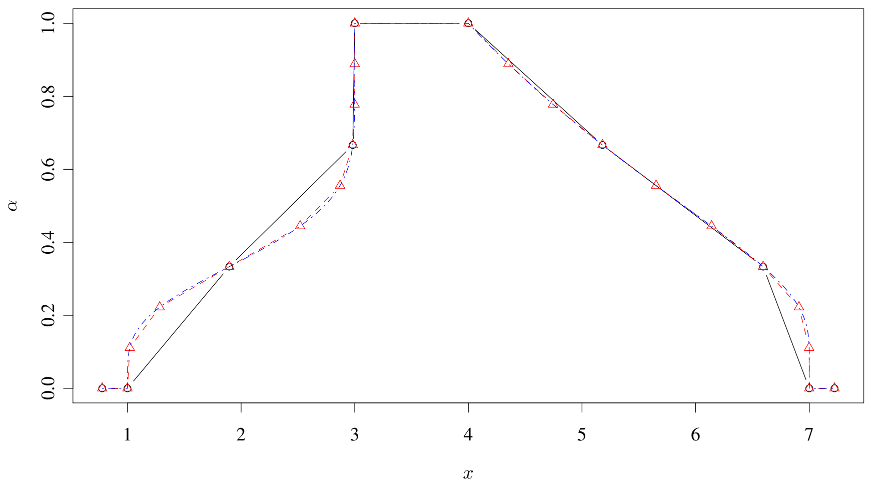
If `knot.n` is equal to 0 or all left and right knots lie on common lines then a PLFN reduces to a TFN. Please note that, however, the `TrapezoidalFuzzyNumber` class does not inherit from `PiecewiseLinearFuzzyNumber` for efficiency reasons. If for some reasons, however, we wanted to convert an object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

```
# CONVERSION: TO BE DONE....
```

3 Depicting FNs

TO BE DONE....

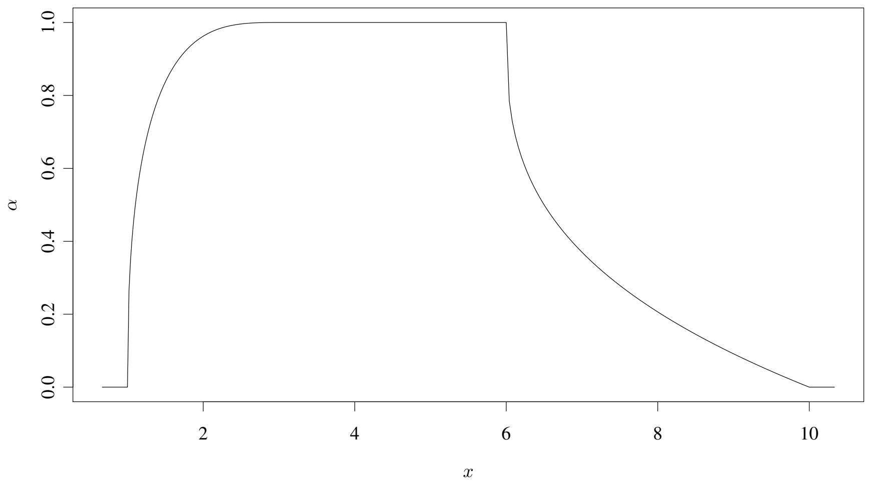
```
plot(A2, n = 4, type = "b")
plot(A2, n = 10, add = TRUE, lty = 2, col = 2, type = "b", pch = 2)
plot(A2, n = 101, add = TRUE, lty = 4, col = 4)  # default n
```



4 Basic computations on and characteristics of FNs

In this section we will consider the following fuzzy number for the sake of illustration:

```
A <- FuzzyNumber(1, 3, 6, 10,
     left=function(x) pbeta(x,0.4,3),
   right=function(x) 1-x^(1/3),
   lower=function(alpha) qbeta(alpha,0.4,3),
   upper=function(alpha) (1-alpha)^3
)
plot(A)
```



4.1 Support and core, and other α-cuts

The support of A, i.e. $\text{supp}(A) = [\texttt{a1}, \texttt{a4}]$, may be obtained by calling:

```
supp(A)

## [1]  1 10
```

We get the core of A, i.e. $\mathrm{core}(A) = [\mathtt{a2}, \mathtt{a3}]$, with:

```
core(A)

## [1] 3 6
```

To compute arbitrary α-cuts we use:

```
alphacut(A, 0)  # same as supp(A)

## [1]  1 10

alphacut(A, 1)  # same as core(A)

## [1] 3 6

alphacut(A, c(0, 0.5, 1))

##       [,1] [,2]
## [1,] 1.000 10.0
## [2,] 1.104  6.5
## [3,] 3.000  6.0
```

Note that if we request to compute more than one α-cut at once, a matrix with 2 columns (instead of a numeric vector of length 2) is returned. This function requires α-cut generators to be provided by the user during the definition of A, even for $\alpha = 0$ or $\alpha = 1$.

4.2 Evaluation of the membership function

TO BE DONE...

```
evaluate(A, 1)

## [1] 0

evaluate(A, c(1, 5, 10))

## [1] 0 1 0

evaluate(A, seq(1, 3, by = 0.5))

## [1] 0.0000 0.8371 0.9625 0.9960 1.0000
```

4.3 “Typical” value

Let us first introduce the notion of the *expected interval* of A [5].

$$\mathrm{EI}(A) := [\mathrm{EI}_L(A), \mathrm{EI}_R(A)] \tag{9}$$

$$= \left[\int_0^1 A_L(\alpha)\,d\alpha, \int_0^1 A_R(\alpha)\,d\alpha\right]. \tag{10}$$

To compute the expected interval of A we call:

```
expectedInterval(A)

## [1] 1.235 7.000
```

Please note that in case of objects of the `FuzzyNumber` class the expected interval is approximated by numerical integration. This method calls the `integrate()` function and its accuracy (quite fine by default) may be controlled by the `subdivisions`, `rel.tol`, and `abs.tol` parameters (call `?integrate` for more details). On the other hand, for 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:

$$\mathrm{EV}(A) := \frac{\mathrm{EI}_L(A) + \mathrm{EI}_R(A)}{2}. \tag{11}$$

Let us calculate $\mathrm{EV}(A)$.

```
expectedValue(A)

## [1] 4.118
```

Note that this method uses a call to `expectedInterval(A)`, thus in case of `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:

$$\mathrm{EV}_w(A) := (1 - w)\mathrm{EI}_L(A) + w\mathrm{EI}_R(A). \tag{12}$$

It is easily seen that $\mathrm{EV}_{0.5}(A) = \mathrm{EV}(A)$.
Some examples:

```
weightedExpectedValue(A, 0.5)  # equivalent to expectedValue(A)

## [1] 4.118

weightedExpectedValue(A, 0.25)

## [1] 2.676
```

The *value* of A [3] is defined by:

$$\mathrm{val}(A) := \int_0^1 \alpha \left(A_L(\alpha) + A_R(\alpha)\right)\, d\alpha. \tag{13}$$

It may be calculated by calling:

```
value(A)

## [1] 3.892
```

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

4.4 Measures of "nonspecificity"

The *width* of A [2] is defined as:

$$\mathrm{width}(A) := \mathrm{EI}_R(A) - w\mathrm{EI}_L(A). \tag{14}$$

An example:

```
width(A)

## [1] 5.765
```

The *ambiguity* of A [3] is defined as:

$$\mathrm{amb}(A) := \int_0^1 \alpha \left(A_R(\alpha) - A_L(\alpha)\right) \, d\alpha. \tag{15}$$

```
ambiguity(A)

## [1] 2.892
```

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

```
diff(supp(A))

## [1] 9
```

5 Approximation of FNs

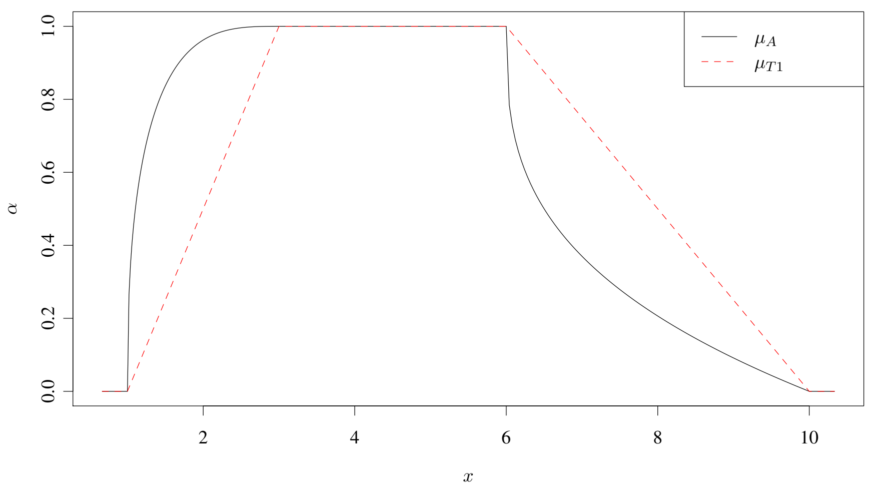
5.1 Approximation by trapezoidal FNs

TO BE DONE... Problem statement... Given a fuzzy number A we seek for a trapezoidal fuzzy number $\mathcal{T}(A)$....

5.1.1 Naive approximator

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

```
(T1 <- trapezoidalApproximation(A, method="Naive"))

## Trapezoidal fuzzy number with support=[1,10] and core=[3,6].

plot(A)
plot(T1, col="red", lty=2)
legend("topright", legend=c(expression(mu[A]), expression(mu[T1])),
                   col=c("blue", "red"), lty=c(1,2))
```



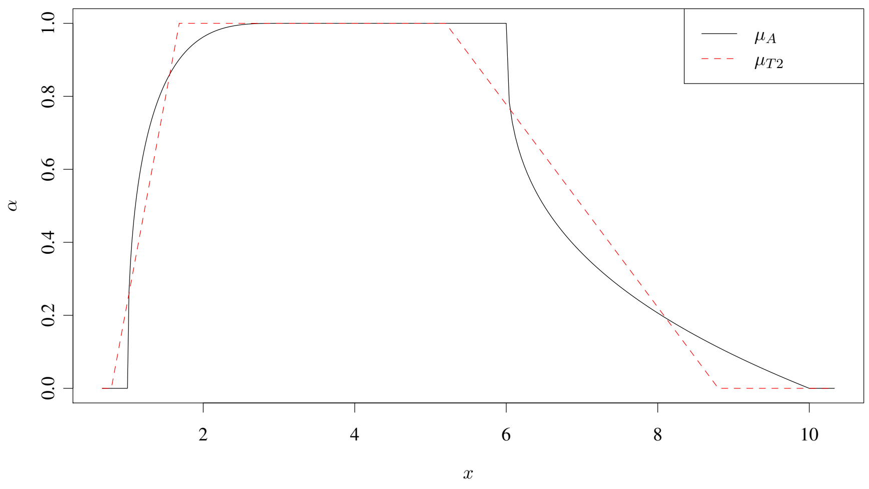
5.1.2 Expected interval preserving approximator

L_2 distance....

The "`ExpectedIntervalPreserving`" method gives the nearest L_2-approximator of A preserving the expected interval [7, 1, 9].

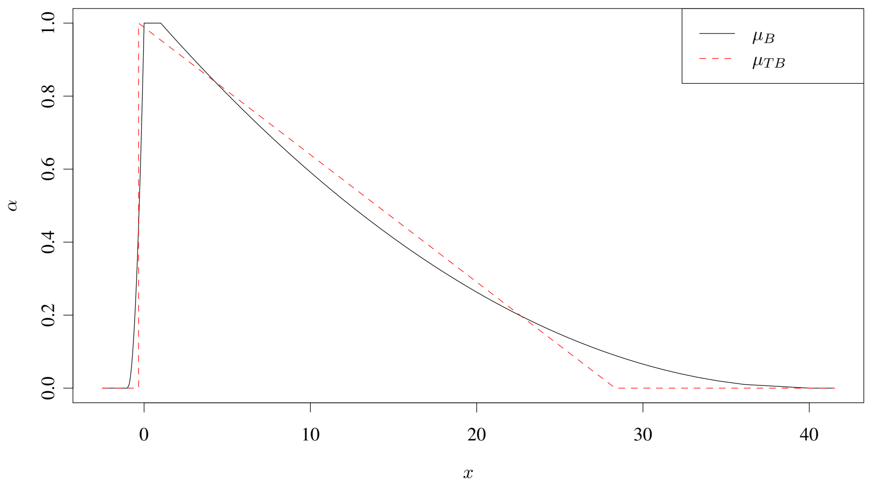
TO BE DONE...

```
(T2 <- trapezoidalApproximation(A, method="ExpectedIntervalPreserving"))

## Trapezoidal fuzzy number with support=[0.786911,8.8] and core=[1.68368,5.2].

expectedInterval(A)

## [1] 1.235 7.000

expectedInterval(T2)

## [1] 1.235 7.000

plot(A)
plot(T2, col="red", lty=2)
legend("topright", legend=c(expression(mu[A]), expression(mu[T2])),
                   col=c("blue", "red"), lty=c(1,2))
```



Unfortunately, for highly skewed membership functions this method reveals sometimes quite unfavorable behavior. E.g. if B is a FN such that $\mathrm{Val}(B) < \mathrm{EV}_{1/3}(B)$ or $\mathrm{Val}(B) > \mathrm{EV}_{2/3}(B)$ then it may happen that the core of the output and the core of the original fuzzy number B are disjoint, cf. [8].

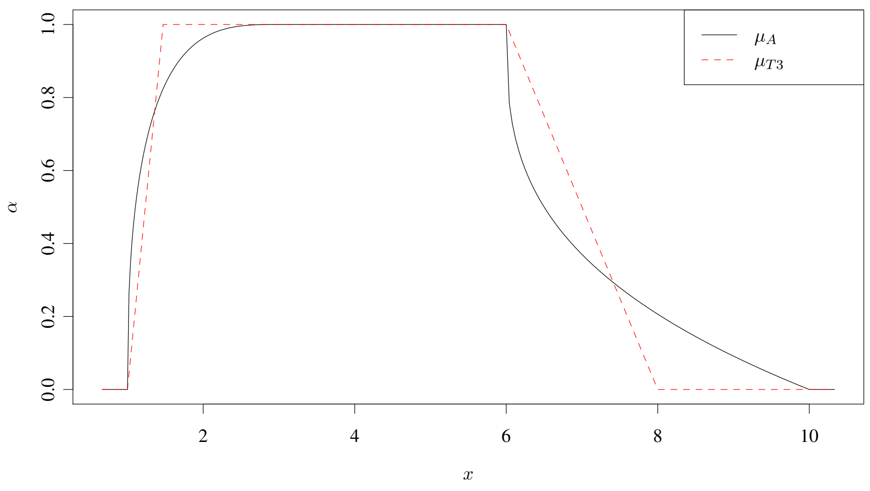
```
(B  <- FuzzyNumber(-1, 0, 1, 40,
   lower=function(x) sqrt(x),
   upper=function(x) 1-sqrt(x)))

## Fuzzy number with support=[-1,40] and core=[0,1].

(TB <- trapezoidalApproximation(B, "ExpectedIntervalPreserving"))

## Trapezoidal fuzzy number with support=[-0.333333,28.3333] and core=[-0.333333,-0.333333].

plot(B)
plot(TB, col="red", lty=2)
legend("topright", legend=c(expression(mu[B]), expression(mu[TB])),
                   col=c("blue", "red"), lty=c(1,2))
```



5.1.3 Approximator with restrictions on support and core

The "`SupportCoreRestricted`" method was proposed in [8]. It gives the L_2-nearest trapezoidal approximation with constraints $\mathrm{core}(A) \subseteq \mathrm{core}(\mathcal{T}(A))$ and $\mathrm{supp}(\mathcal{T}(A)) \subseteq \mathrm{supp}(A)$, i.e. for which each point that surely belongs to A also belongs to $\mathcal{T}(A)$, and each point that surely does not belong to A also does not belong to $\mathcal{T}(A)$.

```
(T3 <- trapezoidalApproximation(A, method="SupportCoreRestricted"))

## Trapezoidal fuzzy number with support=[1,8] and core=[1.47059,6].

plot(A)
plot(T3, col="red", lty=2)
legend("topright", legend=c(expression(mu[A]), expression(mu[T3])),
                   col=c("blue", "red"), lty=c(1,2))
```



TO BE CONTINUED

Bibliography

[1] Ban A.I., Approximation of fuzzy numbers by trapezoidal fuzzy numbers preserving the expected interval, *Fuzzy Sets and Systems* **159**, 2008, pp. 1327–1344.

[2] Chanas S., On the interval approximation of a fuzzy number, *Fuzzy Sets and Systems* **122**, 2001, pp. 353–356.

[3] Delgado M., Vila M.A., Voxman W., On a canonical representation of a fuzzy number, *Fuzzy Sets and Systems* **93**, 1998, pp. 125–135.

[4] Dubois D., Prade H., Operations on fuzzy numbers, *Int. J. Syst. Sci.* **9**, 1978, pp. 613–626.

[5] Dubois D., Prade H., The mean value of a fuzzy number, *Fuzzy Sets and Systems* **24**, 1987, pp. 279–300.

[6] Gągolewski M., `FuzzyNumbers`: Tools to deal with fuzzy numbers in `R`, www.ibspan.waw.pl/∼gagolews/FuzzyNumbers/, 2012.

[7] Grzegorzewski P., *Algorithms for trapezoidal approximations of fuzzy numbers preserving the expected interval*, In: Bouchon-Meunier B. et al (Eds.), *Foundations of Reasoning Under Uncertainty*, Springer, 2010, pp. 85–98.

[8] Grzegorzewski P, Pasternak-Winiarska K., *Trapezoidal approximations of fuzzy numbers with restrictions on the support and core*, In: *Proc. EUSFLAT/LFA 2011*, Atlantic Press, 2011, pp. 749–756.

[9] Yeh C.-T., Trapezoidal and triangular approximations preserving the expected interval, *Fuzzy Sets and Systems* **159**, 2008, pp. 1345–1353.