

FuzzyNumbers Package for R: A Quick Tutorial

Marek Gagolewski

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 2, 2012

THIS IS A PRELIMINARY VERSION OF THE TUTORIAL

Contents

Contents	1
1 Getting started	2
2 How to create instances of fuzzy numbers	2
2.1 Arbitrary FNs	2
2.2 Trapezoidal FNs	6
2.3 Piecewise linear FNs	8
3 Depicting FNs	8
4 Basic computations on and characteristics of FNs	8
4.1 Support and core, and other α -cuts	9
4.2 Evaluation of the membership function	10
4.3 “Typical” value	10
4.4 Measures of “nonspecificity”	10
5 Approximation of FNs	11
5.1 Approximation by trapezoidal FNs	11
5.1.1 Naive approximator	11
5.1.2 Expected interval preserving approximator	11
5.1.3 Approximator with restrictions on support and core	13
Bibliography	14

Acknowledgments. This document has been generated with L^AT_EX, *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 [1], 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.rstudio.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:

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

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

```
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:

```
require("FuzzyNumbers") # Load the package
```

To view the main page of the manual we type:

```
library(help = "FuzzyNumbers")
```

For more information please visit the package’s homepage [3]. 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:

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

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 *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 $\text{left}(0) = 0$, $\text{left}(1) = 1$, $\text{right}(0) = 0$, and $\text{right}(1) = 1$.

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 $l_A(x) = \frac{x-a_1}{a_2-a_1}$ and $r_A(x) = \frac{x-a_3}{a_4-a_3}$) 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"
```

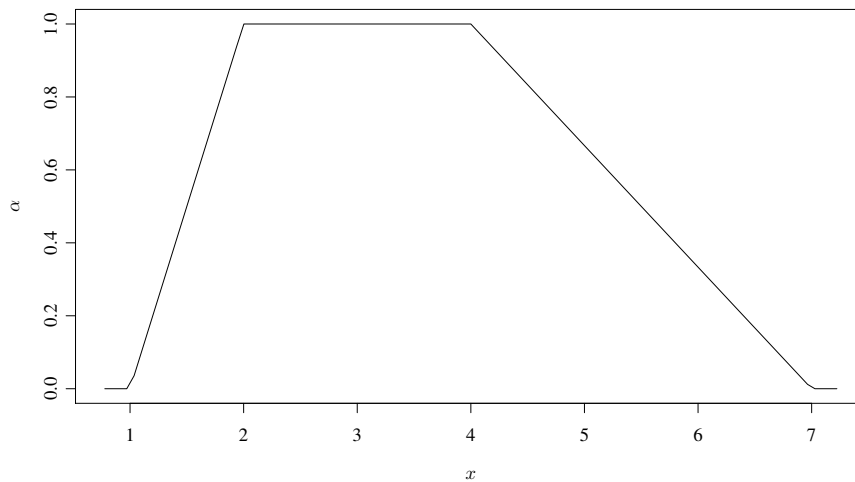
We may print some basic information on A_1 by calling `print(A1)` or simply by typing:

```
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 `postscript()` 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 $a1 \leq a2 \leq a3 \leq a4$):

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

$$= [a1 + (a2 - a1) \cdot \text{lower}(\alpha), a3 + (a4 - a3) \cdot \text{upper}(\alpha)], \quad (3)$$

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

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

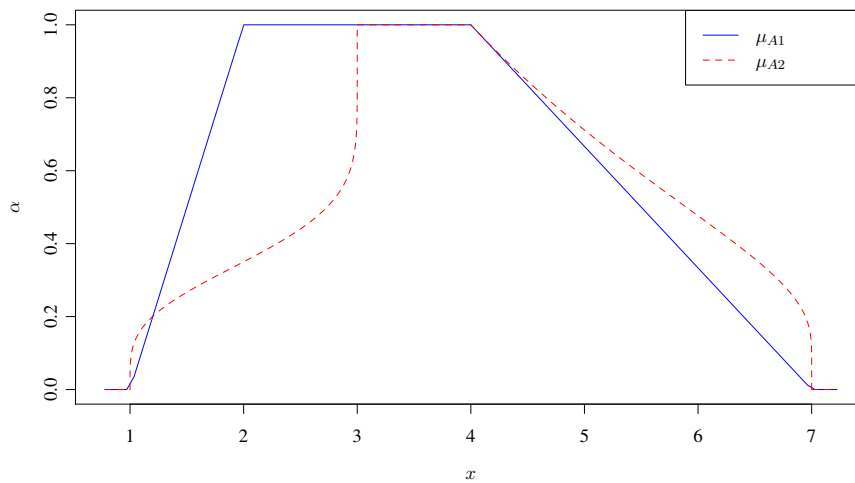
$$\text{lower}(\alpha) = \inf\{x : \text{left}(x) \geq \alpha\}, \quad (4)$$

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

Moreover, if side generating functions are continuous and strictly monotonic then 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)
```



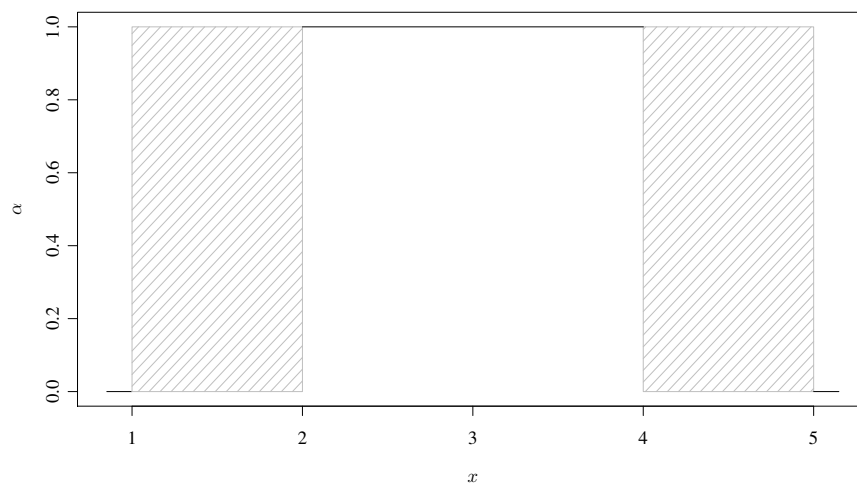
Definition with generating functions omitted. Please note that in the above examples we passed to the constructor of each `FuzzyNumber` class instance either side generating functions or α -cut generators. Let us study what happens if we omit both of them.

```
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. In 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: 0x37534e8>

A2["upper"]

## function(alpha) pexp(1/alpha-1)
## <environment: 0x37534e8>

A2["left"]

## function (x)
## NA
## <environment: 0x3fdae60>

A2["right"]

## function (x)
## NA
## <environment: 0x3fdae60>

```

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, $\text{core}(T_1) = [2, 4]$ and $\text{supp}(T_1) = [1, 7]$ we call:

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

This object is an instance of the following R class:

```

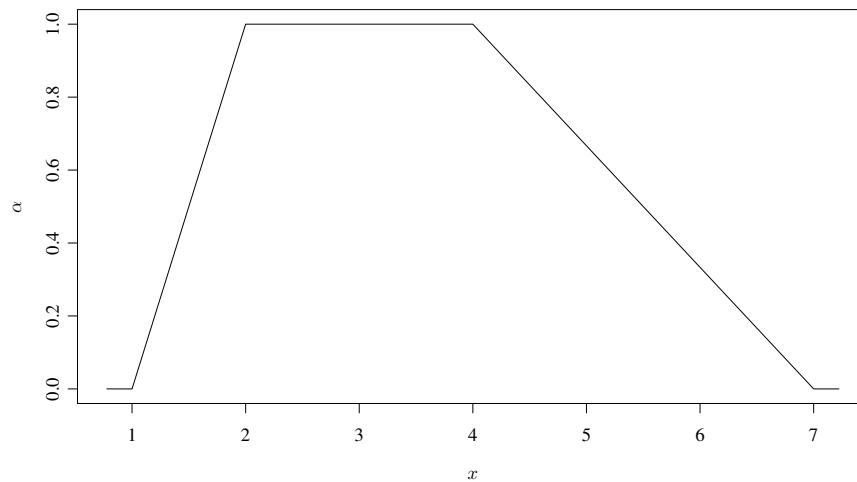
class(T1)

## [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, if 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

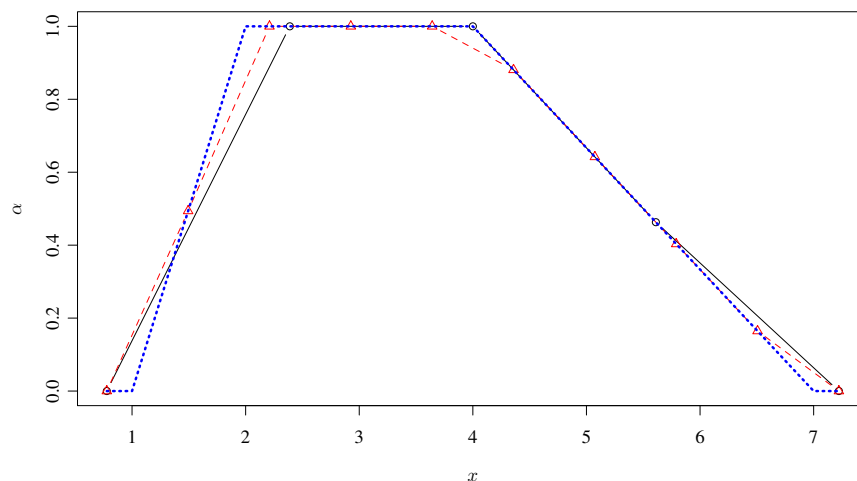
Trapezoidal fuzzy numbers are generalized by piecewise linear FNs (PLFNs).

TO DO NOT YET IMPLEMENTED....

3 Depicting FNs

TO BE DONE....

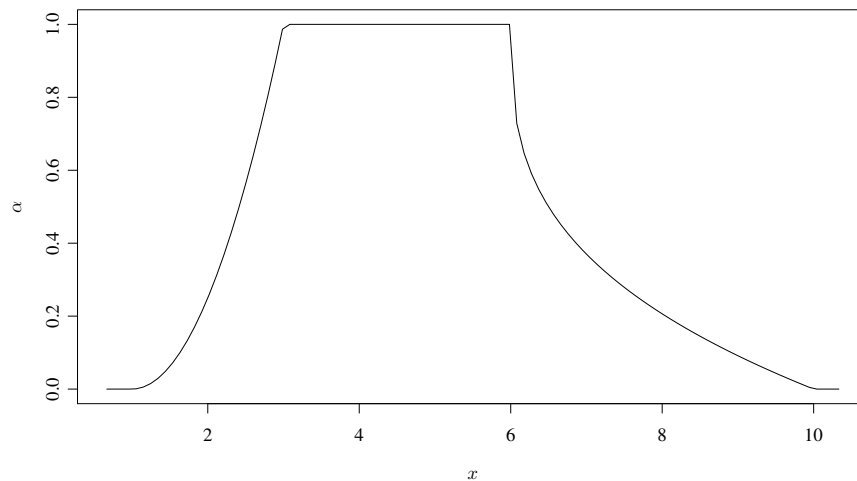
```
plot(A1, n = 5, type = "b")
plot(A1, n = 10, add = TRUE, lty = 2, col = 2, type = "b", pch = 2)
plot(T1, add = TRUE, lty = 3, col = 4, lwd = 4) # TFNs are always depicted
'accurately'
```



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) x^2,
  right=function(x) 1-x^(1/3),
  lower=function(alpha) alpha^(1/2),
  upper=function(alpha) (1-alpha)^3
)
plot(A)
```



4.1 Support and core, and other α -cuts

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

```
supp(A)
## [1] 1 10
```

We get the core of A , i.e. $\text{core}(A) = [a2, a3]$, 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,] 2.414  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 0.0625 0.2500 0.5625 1.0000
```

4.3 “Typical” value

This is approximated numerically...

TO BE DONE...

```
expectedInterval(A)

## [1] 2.333 7.000
```

TO BE DONE...

```
expectedValue(A)

## [1] 4.667

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

## [1] 4.667

weightedExpectedValue(A, 0.25)

## [1] 3.5

value(A)

## [1] 4.5
```

Note that these may be used to “defuzzify” A .

4.4 Measures of “nonspecificity”

TO BE DONE...

```
width(A)

## [1] 4.667
```

```
ambiguity(A)
```

```
## [1] 3.5
```

5 Approximation of FNs

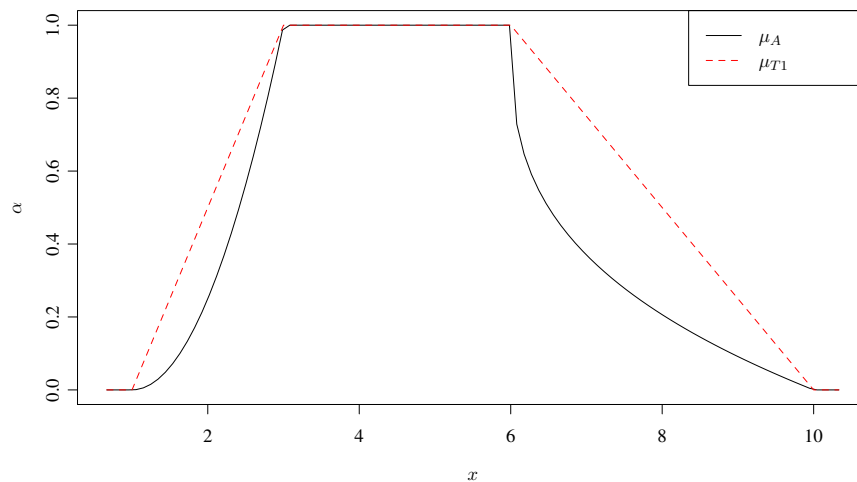
5.1 Approximation by trapezoidal FNs

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 $[4, 2, 6]$.

TO BE DONE...

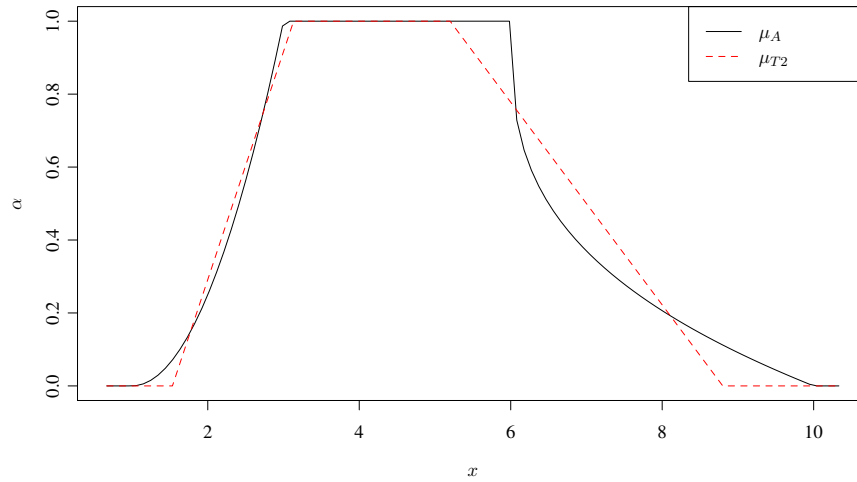
```
(T2 <- trapezoidalApproximation(A, method="ExpectedIntervalPreserving"))  
  
## Trapezoidal fuzzy number with support=[1.53333,8.8] and core=[3.13333,5.2].  
  
expectedInterval(A)
```

```
## [1] 2.333 7.000

expectedInterval(T2)

## [1] 2.333 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 $\text{Val}(B) < \text{EV}_{1/3}(B)$ or $\text{Val}(B) > \text{EV}_{2/3}(B)$ then it may happen that the core of the output and the core of the original fuzzy number B are disjoint, cf. [5].

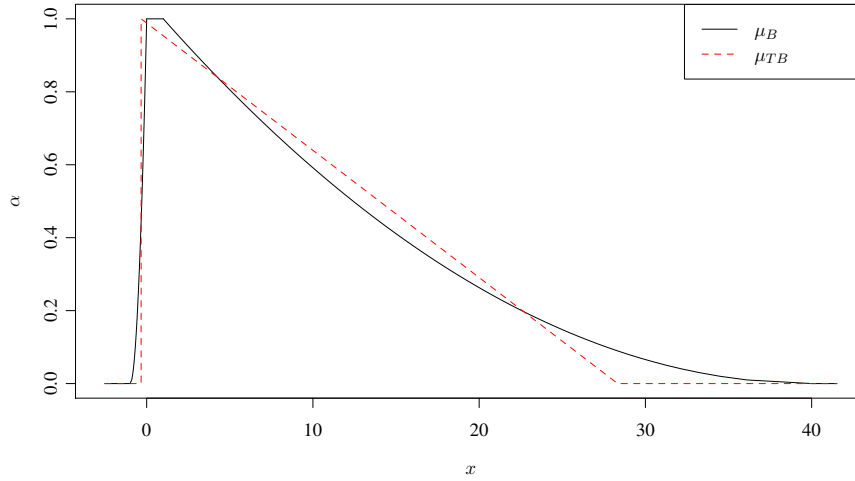
```
(B <- FuzzyNumber(-1, 0, 1, 40,
  lower=function(x) sqrt(x),
  upper=function(x) 1-sqrt(x)))

## 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))
```



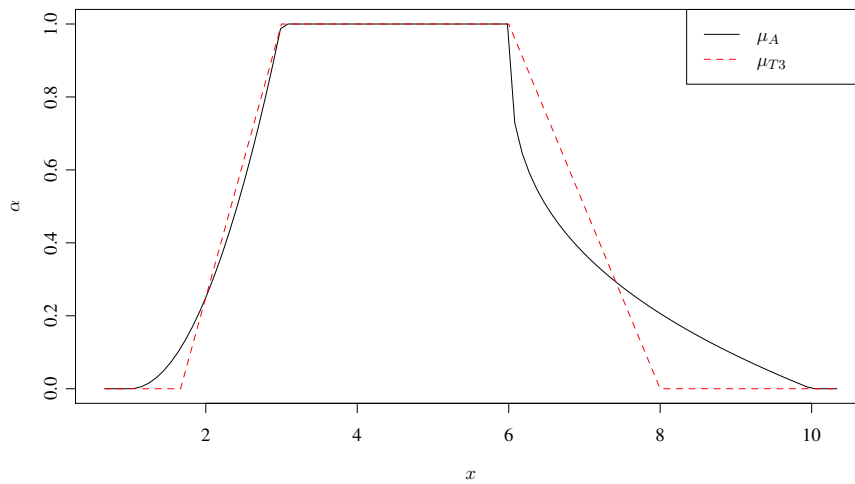
5.1.3 Approximator with restrictions on support and core

The "SupportCoreRestricted" method was proposed in [5]. It gives the L_2 -nearest trapezoidal approximation with constraints $\text{core}(A) \subseteq \text{core}(\mathcal{T}(A))$ and $\text{supp}(\mathcal{T}(A)) \subseteq \text{supp}(A)$, 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=[1.66667,8] and core=[3,6].

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] Dubois D., Prade H., Operations on fuzzy numbers, *Int. J. Syst. Sci.* **9**, 1978, pp. 613–626.
- [2] Ban A.I., Approximation of fuzzy numbers by trapezoidal fuzzy numbers preserving the expected interval, *Fuzzy Sets and Systems* **159**, 2008, pp. 1327–1344.
- [3] Gagolewski M., FuzzyNumbers: Tools to deal with fuzzy numbers in R, www.ibspan.waw.pl/~gagolews/FuzzyNumbers/, 2012.
- [4] 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.
- [5] 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.
- [6] Yeh C.-T., Trapezoidal and triangular approximations preserving the expected interval, *Fuzzy Sets and Systems* **159**, 2008, pp. 1345–1353.