# 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 1, 2012

---

## 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

R is a free software environment for statistical computing and graphics, which includes an implementation of a very powerful and quite popular high-level language called S. To install R and/or find some information on the S language please visit R Project's Homepage at www.R-project.org. Note that the FuzzyNumbers package requires R version $\geq 2.15$.

To install the latest "official" release of FuzzyNumbers available on CRAN we type:

```
# ... TO DO ...
```

Alternatively, we may fetch a development snapshot (perhaps unstable, but newer) 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 at www.ibspan.waw.pl/~gagolews/FuzzyNumbers/. 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 Trapezoidal FNs

First let us create a trapezoidal fuzzy number $T$ such that $\mathrm{core}(T) = [2, 4]$ and $\mathrm{supp}(T) = [1, 7]$:

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

This object is an instance of the following R class:

```
class(T)

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

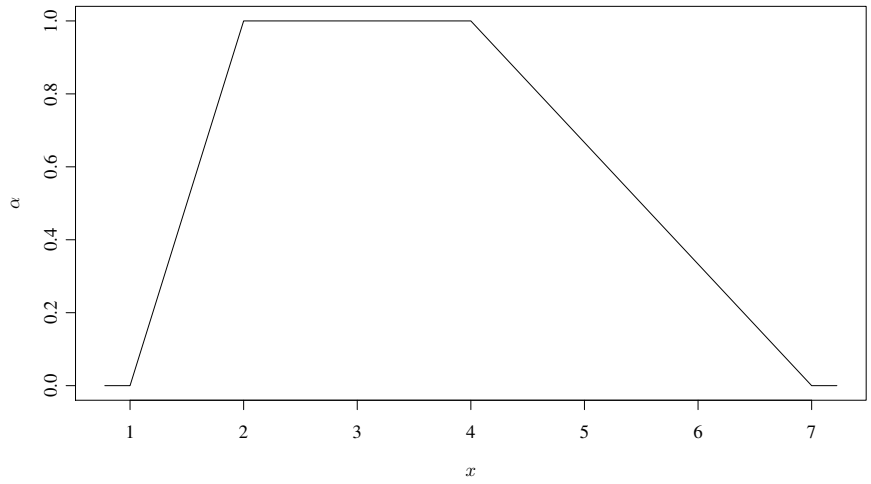
We may print some basic information by calling `print(T)` or simply by typing:

```
T

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

Bingo! :-)

To depict $T$ we call:

```
plot(T)
```




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 FNs 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].
```

## 2.2 Arbitrary FNs

Generally, an arbitrary 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 assume we provide at least the latter, i.e. $\alpha$-cuts.

**Side functions.** A fuzzy number $A$ specified by side functions has a membership function:

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

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:

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

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

A1

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

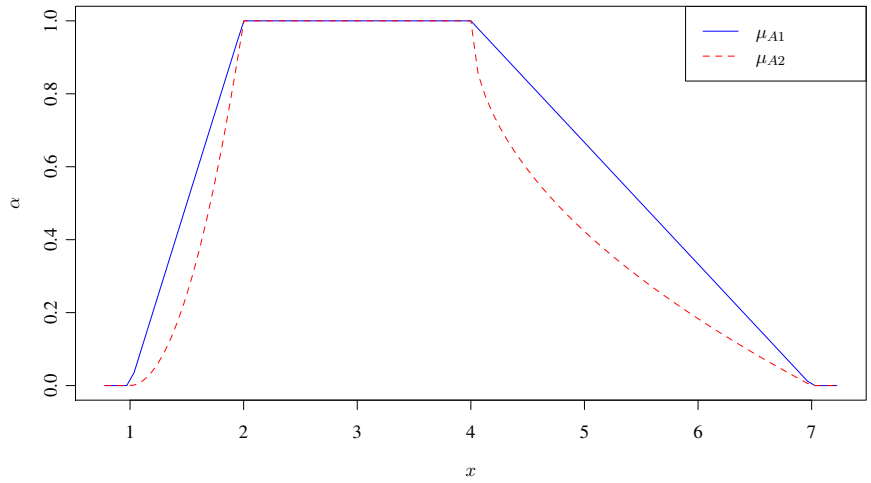
The above is (roughly — we will see why in a moment) equivalent to the trapezoidal fuzzy number $T$ defined in Sec. 2.1. The `TrapezoidalFuzzyNumber` class inherits all the goodies from the `FuzzyNumber` class, but is more specific (guarantees faster computations, contains more detailed information, etc.). Thus, is we wanted to define a trapezoidal FN next time, we would rather not do it like in the previous section.

Another example:

```
A2 <- FuzzyNumber(1,2,4,7,
    left=function(x) x^2,
   right=function(x) 1-sqrt(x)
)
```

One picture is worth thousand words, so...

```
plot(A1, col="blue")
plot(A2, col="red", lty=2, add=TRUE)
legend("topright", legend=c(expression(mu[A1]), expression(mu[A2])),
                   col=c("blue", "red"), lty=c(1,2))
```




**$\alpha$-cut bounds.** Alternatively, $A$ may be defined by specifying its $\alpha$-cuts. We have (for $\alpha \in (0,1)$ and $\texttt{a1} \leq \texttt{a2} \leq \texttt{a3} \leq \texttt{a4}$):

$$A_\alpha := [A_L(\alpha), A_R(\alpha)] \tag{2}$$
$$= \big[\texttt{a1} + (\texttt{a2} - \texttt{a1}) \cdot \texttt{lower}(\alpha), \texttt{a3} + (\texttt{a4} - \texttt{a3}) \cdot \texttt{upper}(\alpha)\big], \tag{3}$$

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

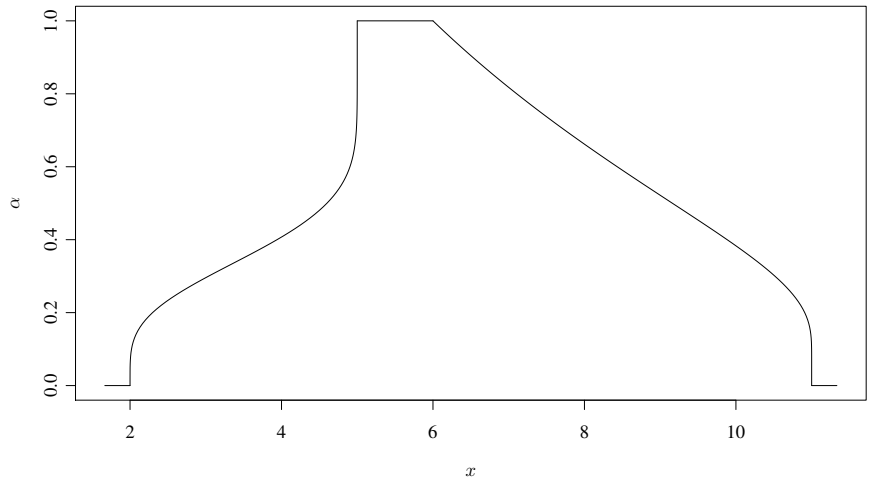
It is easily seen that for $\alpha \in (0,1)$ it holds:

$$\texttt{lower}(\alpha) = \inf\{x : \texttt{left}(x) \geq \alpha\}, \tag{4}$$
$$\texttt{upper}(\alpha) = \sup\{x : \texttt{right}(x) \geq \alpha\}. \tag{5}$$

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

An example:

```
A3 <- FuzzyNumber(2,5,6,11,
         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(A3)
```

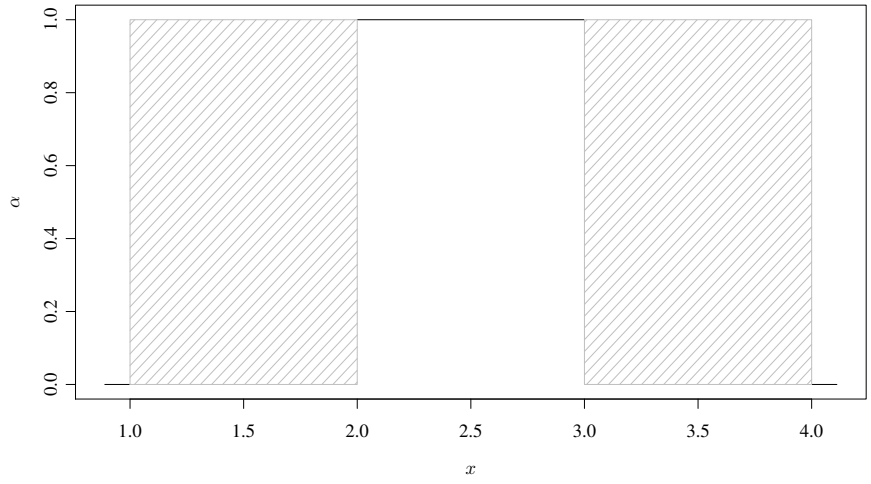



Please note that — up to now — we were passing to the constructor of each `FuzzyNumber` class instance either side generating functions or $\alpha$-cut generators. Let us study what will happen if we omit both of them.

```
A4 <- FuzzyNumber(1, 2, 3, 4)
A4

## Fuzzy number with support=[1,4] and core=[2,3].
```

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

```
plot(A4)
```




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:

```
A4 <- FuzzyNumber(1, 2, 3, 4, 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 $\alpha$-cuts? Indeed!

```
alphacut(A3, 0.5)  # A3 has alpha-cut generators defined

## [1] 4.600 9.161

alphacut(A1, 0.5)  # A1 has not

## [1] NA NA

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

## [1] 0.1667

evaluate(A3, 6.5)  # A3 has not

## [1] NA
```

The reason why 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 a procedure (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 `A3`:

```
A3["lower"]

## function(alpha) pbeta(alpha, 5, 9)
## <environment: 0x386d238>

A3["upper"]

## function(alpha) pexp(1/alpha-1)
## <environment: 0x386d238>

A3["left"]

## function (x)
## NA
## <environment: 0x4209b08>

A3["right"]

## function (x)
## NA
## <environment: 0x4209b08>
```

Thus, a general rule is: if you want $\alpha$-cuts (e.g. for finding trapezoidal approximations of FNs) — specify them. If you want to access side functions (what for? luckily, the `plot()` function automatically detects what kind of knowledge we have) — provide them.

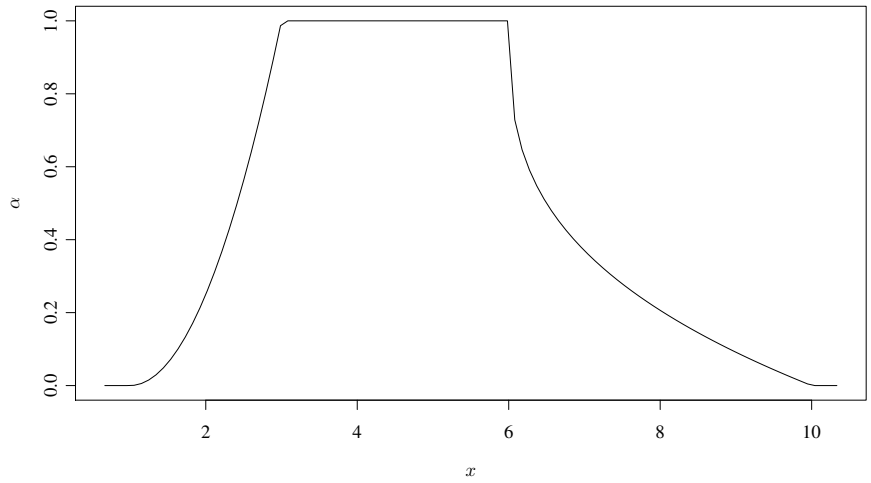
Of course, in case of e.g. trapezoidal fuzzy numbers their generating functions are known *a priori* and there is no need to provide them manually (what is more, this has been disallowed for safety reasons).

```
T["lower"]

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

T["upper"]

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

T["left"]

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

T["right"]

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

# 3 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)
```




### 3.1 Support and core, and other $\alpha$-cuts

The support of $A$, i.e. $\mathrm{supp}(A) = [\mathtt{a1}, \mathtt{a4}]$, 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 $\alpha$-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  9.5
## [3,] 3.000  6.0
```

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

### 3.2 Evaluation of the membership function

TO BE DONE...

```r
evaluate(A, 1)
```

```
## [1] 0
```

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

```
## [1] 0 1 0
```

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

```
## [1] 0.0000 0.0625 0.2500 0.5625 1.0000
```

### 3.3 "Typical" value

TO BE DONE...

```r
expectedInterval(A)
```

```
## [1] 2.333 9.000
```

TO BE DONE...

```r
expectedValue(A)
```

```
## [1] 5.667
```

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

```
## [1] 5.667
```

```r
weightedExpectedValue(A, 0.25)
```

```
## [1] 4
```

```r
value(A)
```

```
## [1] 5.5
```

Note that these may be used to "defuzzify" $A$.

### 3.4 Measures of "nonspecificity"

TO BE DONE...

```r
width(A)
```

```
## [1] 6.667
```

```r
ambiguity(A)
```

```
## [1] 4.5
```

TO BE CONTINUED