A Guide to the FuzzyNumbers 0.02 Package for R

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/

December 27, 2012

The package, as well as this tutorial, is still in its early days – any suggestions are welcome!

Contents

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 [6], form a particular subclass of fuzzy sets of the real line. They play a significant role in many important theoretical and practical considerations (cf. [12]) 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:

```
install.packages("FuzzyNumbers")
```

Alternatively, we may fetch its current development snapshot (without man pages, but see our homepage[1]) from *RForge*:

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

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 [8]. 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 fuzzy numbers

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 that deal with FNs assume we provide at least the latter, i.e. α-cuts.

2.1.1 Definition by side functions

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

[1] www.ibspan.waw.pl/~gagolews/?page=resources&subpage=FuzzyNumbers&manpage=00Index.

[2] Side functions are sometimes called branches or shape functions in the literature.

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

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 $\texttt{left}(0) \ge 0$, $\texttt{left}(1) \le 1$, $\texttt{right}(0) \le 1$, and $\texttt{right}(1) \ge 0$.

Please note that by using side generating functions defined on $[0,1]$ we really make (in author's humble opinion) the process of generating examples for our publications much easier. A similar concept was used e.g. in [13] (LR-fuzzy numbers).

An example: a fuzzy number A_1 with linear sides (a trapezoidal fuzzy number, see also Sec. 2.4).

```
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"
```

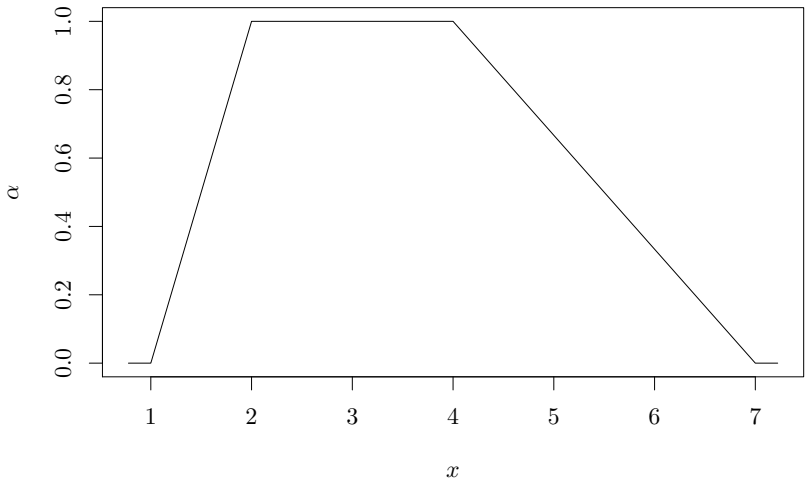
We may print some basic information on A_1 by calling `print(A1)` or simply by typing:

```
A1
```

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

To depict A_1 we call:

```
plot(A1)
```



If we would like to generate figures for our publications, then we will be interested in storing them as PDF files. 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
```

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

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

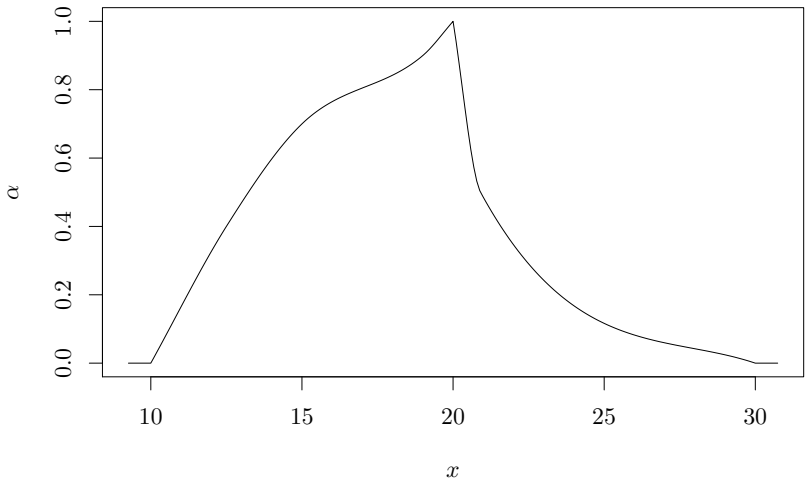
```
f <- splinefun(c(-4,-3.5,-3,-2.2,-2), c(0,0.4,0.7,0.9,1), method="monoH.FC")
g <- splinefun(c(-1,0,10), c(1,0.5,0), method="monoH.FC")
```

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

```
convert.side(f, -4, -2)(c(0,1))
## [1] 0 1
convert.side(g, -1, 10)(c(0,1))
## [1] 1 0
convert.side(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.

```
A <- FuzzyNumber(10,20,20,30,
    left=convert.side(f, -4, -2),
   right=convert.side(g, -1, 10)
)
plot(A, xlab="$x$", ylab="$\\alpha$")
```



2.1.2 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 $\texttt{a1} \leq \texttt{a2} \leq \texttt{a3} \leq \texttt{a4}$):

$$A_\alpha := [A_L(\alpha), A_U(\alpha)] \tag{2}$$

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

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

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

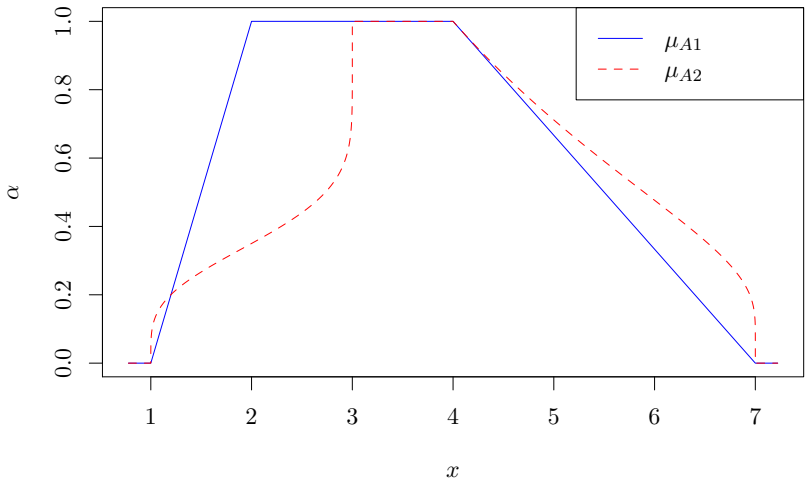
$$\texttt{lower}(\alpha) = \inf\{x : \texttt{left}(x) \geq \alpha\}, \tag{4}$$

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

Moreover, if side generating functions are continuous and strictly monotonic, then 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)
legend("topright", c(expression(mu[A1]), expression(mu[A2])),
   col=c("blue", "red"), lty=c(1,2));
```



Remark. The `convert.alpha()` function works similarly to `convert.side()`. This tool, however, 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

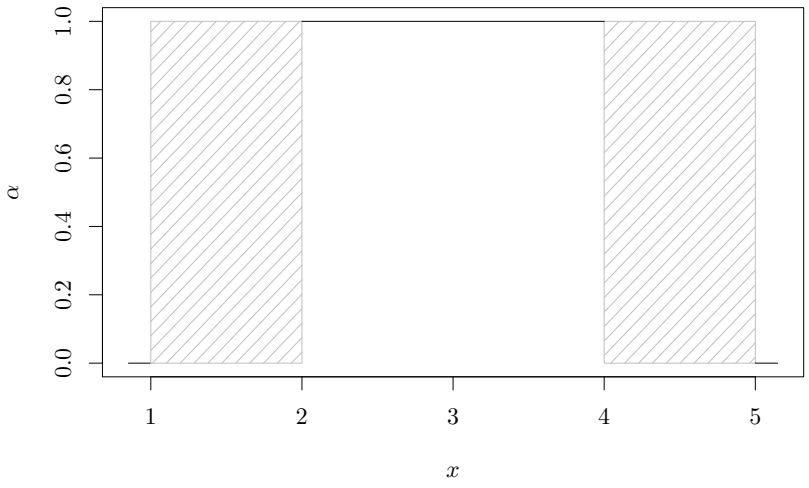
Please note that in the above examples we passed to the constructor of each `FuzzyNumber` class instance either side generating functions or α-cut generators. Let us study what happens, if we omit both of them.

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

```
## Fuzzy number with:
##    support=[1,5],
##       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 α-cuts? Indeed!

```
alphacut(A2, 0.5) # A2 has alpha-cut generators defined
## [1] 2.733154 5.896362
alphacut(A1, 0.5) # A1 hasn't got them
## [1] NA NA
```

Another example: evaluation of the membership function.

```
evaluate(A1, 6.5) # A1 has side generators defined
## [1] 0.1666667
evaluate(A2, 6.5) # A2 hasn't got them
## [1] NA
```

2.2 Using numeric approximations of α-cut or side generators

The reason for setting by default `NA`s[3] as return values of generators (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 at the internals of `A2`:

```
A2["lower"]
## function(alpha) pbeta(alpha, 5, 9)
## <environment: 0x39f0fd8>
A2["upper"]
```

[3]To be precise, it's `NA_real_`.

```
## function(alpha) pexp(1/alpha-1)
## <environment: 0x39f0fd8>
```

```
A2["left"]
```

```
## function (x)
## rep(NA_real_, length(x))
## <environment: 0x2399140>
```

```
A2["right"]
```

```
## function (x)
## rep(NA_real_, length(x))
## <environment: 0x2399140>
```

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 α-cuts (e.g. for finding trapezoidal approximations of FNs), specify them. If you would like to access side functions (by the way, the `plot()` function automatically detects what kind of knowledge we have), assure they are provided.

However, we we provide some convenient short-cut methods to *interpolate* generating functions of one type to get some crude numeric approximations of their inverses. These are simple wrappers to R's **`approxfun()`** (piecewise linear interpolation, the *`"linear"` method*) and **`splinefun()`** (monotonic splines: *`method`*s *`"hyman"`* and *`"monoH.FC"`*; the latter is default and recommended). They are available as the **`approx.invert()`** function[4], and may of course be used on results returned by **`convert.alpha()`** and **`convert.side()`**.

```
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 = approx.invert(l),
   upper = approx.invert(r)
)

x <- seq(0,1,length.out=1e5)
max(abs(qbeta(x, 1, 2) - A4["lower"](x)))     # sup-error
```

```
## [1] 0.0001389811
```

```
max(abs(qbeta(1-x, 1, 0.1) - A4["upper"](x))) # sup-error
```

```
## [1] 0.0008607773
```

2.3 Fuzzy numbers with discontinuities

... TO BE DONE ....

```
A1 <- FuzzyNumber(0,1,1,1,
         lower=function(a) floor(3*a)/3,
         upper=function(a) 1-a
) # no info on discontinuities

A2 <- DiscontinuousFuzzyNumber(0,1,1,1,
```

[4]The 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 "any" is better than "nothing" sometimes.

```
        lower=function(a) floor(3*a)/3,
        upper=function(a) 1-a,
        discontinuities.lower=c(0, 1/3, 2/3, 1),
        discontinuities.upper=numeric(0)
) # discontinuities info included
```

... TO BE DONE ....

2.4 Trapezoidal fuzzy numbers

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)
```

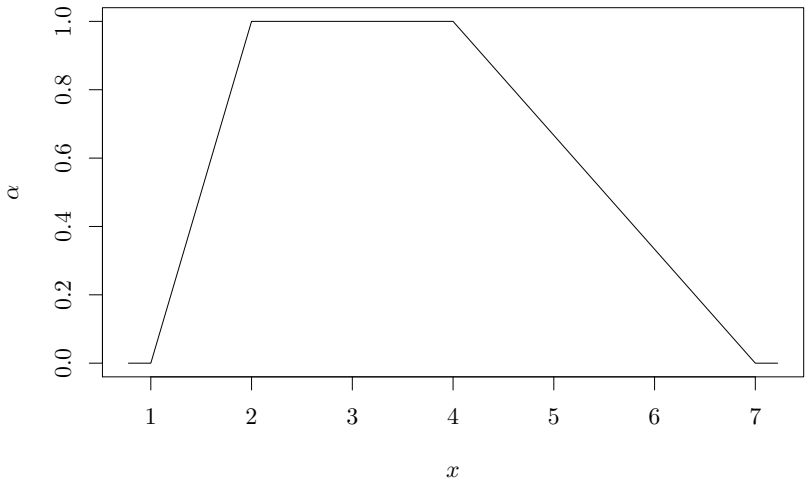
This object is an instance of the following R class:

```
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, 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],
##       core=[2,2].
TrapezoidalFuzzyNumber(2,2,3,3)  # `crisp' interval
## 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].
```

2.5 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 $\texttt{a1} \leq \texttt{a2} \leq \texttt{a3} \leq \texttt{a4}$ defining its support and core,
- the number of "knots", $\texttt{knot.n} \geq 0$,
- 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 $\in [\texttt{a1}, \texttt{a2}]$, defining interpolation points for the left side function, and
- a nondecreasingly sorted vector `knot.right` consisting of `knot.n` elements $\in [\texttt{a2}, \texttt{a3}]$, 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:

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

and its α-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 $n =$ `knot.n`, $(l_1, \ldots, l_{n+2}) =$ (`a1`, `knot.left`, `a2`), $(r_1, \ldots, r_{n+2}) =$ (`a3`, `knot.right`, `a4`), and $(\alpha_1, \ldots, \alpha_{n+2}) = (0,$ `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)
class(P1)
```

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

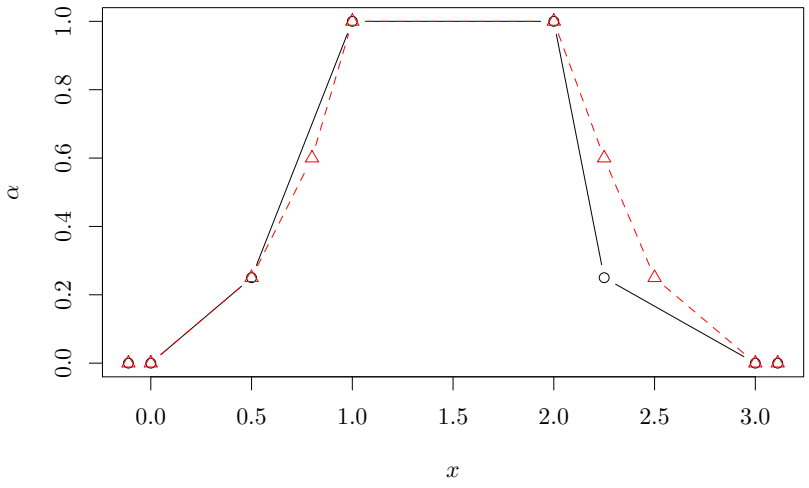
```
P1
```

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

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



The following operators return matrices with all knots of a PLFN. Each of them have three columns, in order: α-cuts, left side coordinates, and right side coordinates.

```
P2["knots"]
```

```
##        alpha left right
## knot_1  0.25  0.5  2.50
## knot_2  0.60  0.8  2.25
```

```
P2["allknots"] # including a1,a2,a3,a4
```

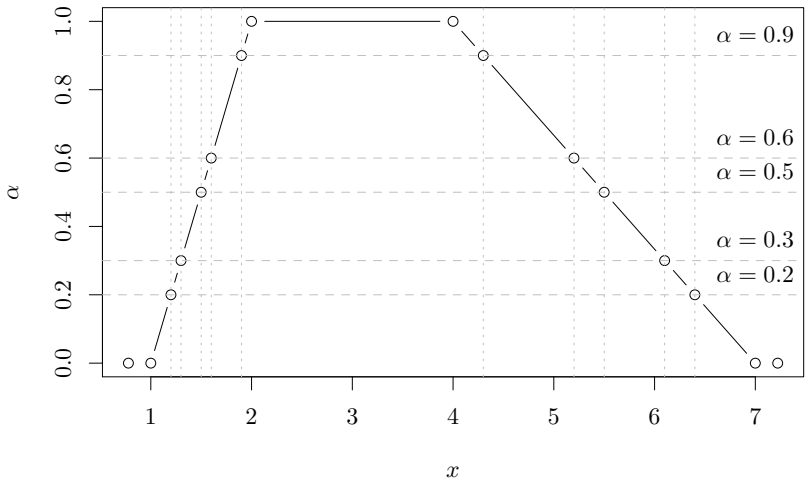
```
##        alpha left right
## supp    0.00  0.0  3.00
## knot_1  0.25  0.5  2.50
## knot_2  0.60  0.8  2.25
## core    1.00  1.0  2.00
```

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, however, we wanted to convert an object of the first mentioned class to the other, we would do that by calling:

```
alpha <- c(0.2, 0.3, 0.5, 0.6, 0.9);
P3 <- as.PiecewiseLinearFuzzyNumber(T1,
        knot.n=5, knot.alpha=alpha);
P3
```

```
## Piecewise linear fuzzy number with 5 knot(s),
##    support=[1,7],
##       core=[2,4].
```

```
plot(P3)
abline(h=alpha, col="gray", lty=2)
abline(v=P3["knot.left"], col="gray", lty=3)
abline(v=P3["knot.right"], col="gray", lty=3)
text(7, alpha, sprintf("a=%g", alpha), pos=3)
```



More generally, each PLFN or TFN may be converted to a direct `FuzzyNumber` class instance if needed (hope we will newer not).

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

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

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. 5.

2.6 Fuzzy numbers with sides given by power functions

Fuzzy numbers which sides are given by power functions are defined with four coefficients $\texttt{a1} \leq \texttt{a2} \leq \texttt{a3} \leq \texttt{a4}$, and parameters $\texttt{p.left}, \texttt{p.right} > 0$ which determine exponets for the side functions:

$$\texttt{left}(x) = x^{\texttt{p.left}}, \tag{9}$$

$$\texttt{right}(x) = (1-x)^{\texttt{p.right}}. \tag{10}$$

We also have:

$$\texttt{lower}(\alpha) = \sqrt[\texttt{p.left}]{\alpha}, \tag{11}$$

$$\texttt{upper}(\alpha) = 1 - \sqrt[\texttt{p.right}]{\alpha}. \tag{12}$$

These fuzzy numbers are another natural generalization of trapezoidal FNs.

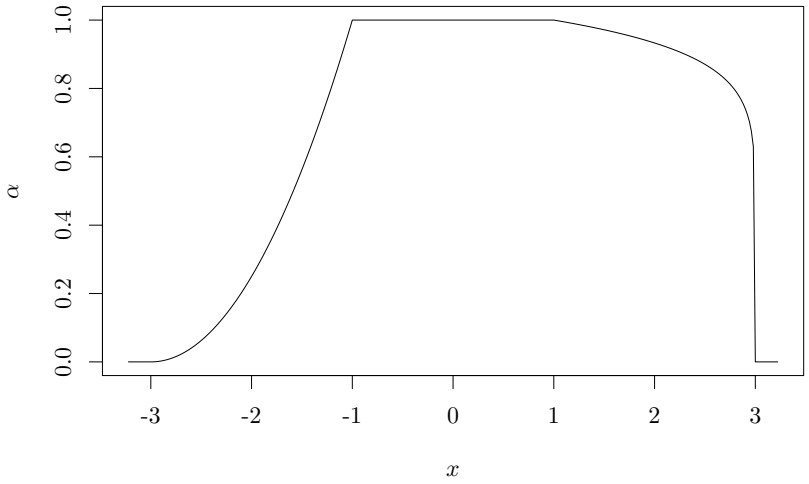
An example:

```
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)
```

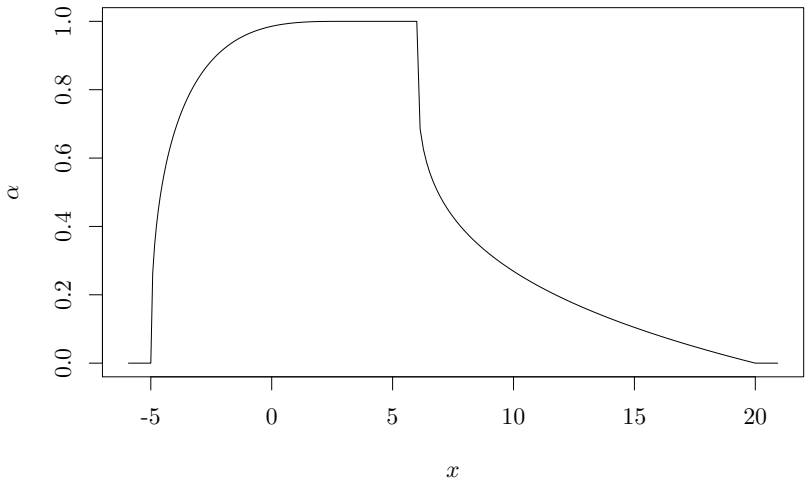


3 Depicting fuzzy numbers

To depict FNs we use the `plot()` method, which uses similar parameters as the R-built-in `curve()` function. 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).

In this and subsequent sections we consider the following fuzzy number for the sake of illustration:

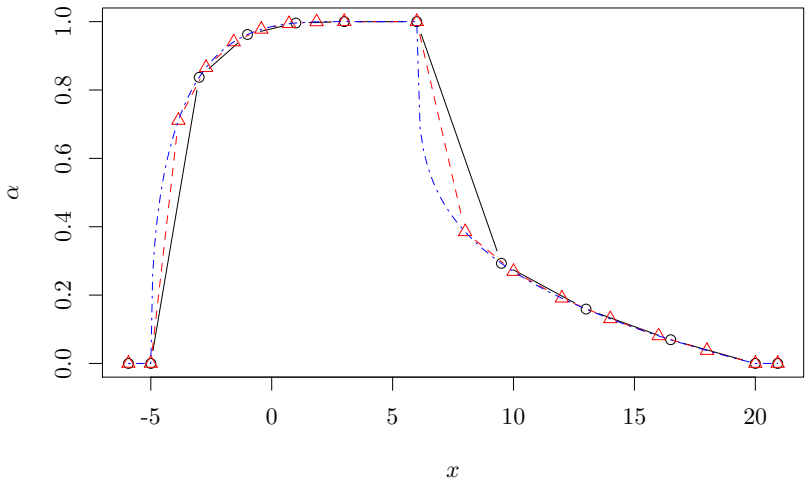
```
A <- FuzzyNumber(-5, 3, 6, 20,
    left=function(x) pbeta(x,0.4,3),
   right=function(x) 1-x^(1/4),
   lower=function(alpha) qbeta(alpha,0.4,3),
   upper=function(alpha) (1-alpha)^4
)
plot(A)
```



Side functions or α-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` parameter (which defaults to `101` = quite accurate).

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

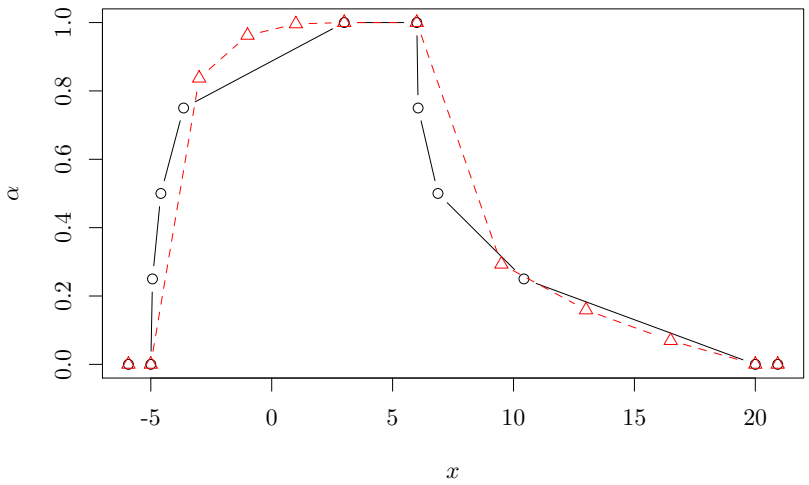
```
plot(A, n=3, type='b')
plot(A, n=6, add=TRUE,  lty=2, col=2, type='b', pch=2)
plot(A, n=101, add=TRUE, lty=4, col=4) # default n
```



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 α-cuts will also suffice. The function is smart enough to detect the internal representation of the FN and use the kind representation it

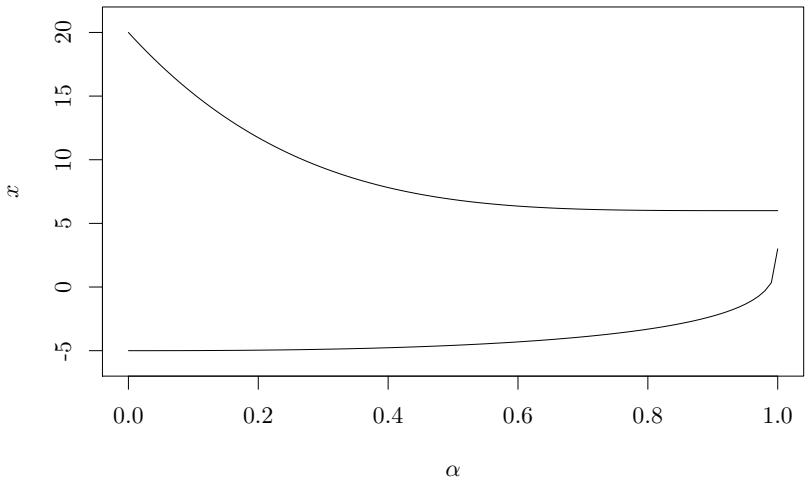
has. It both types of generators are given then side functions are used. If we want to, for some reasons, use α-cuts, then we may do as follows:

```
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 α-cut representation of a fuzzy number:

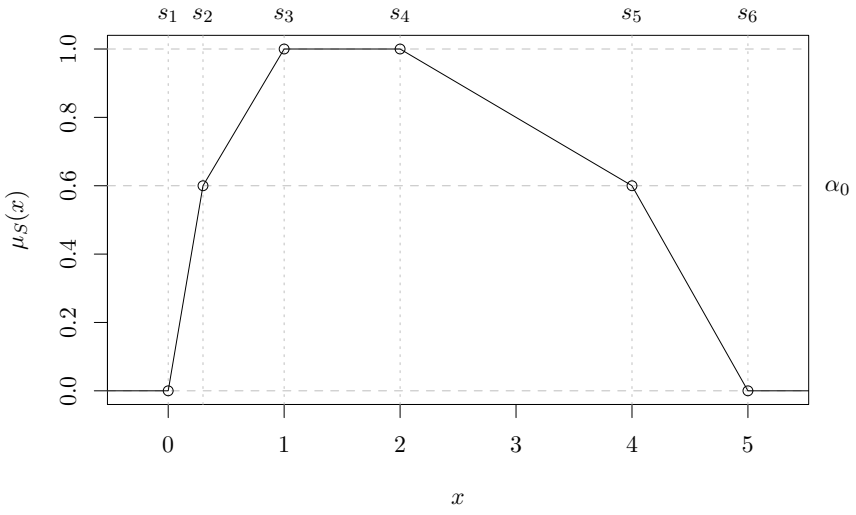
```
plot(A, draw.alphacuts=TRUE)
```



Finally, we leave you with a quite complex example from one of our papers:

```
X <- PiecewiseLinearFuzzyNumber(0, 1, 2, 5, knot.n=1,
   knot.alpha=0.6, knot.left=0.3, knot.right=4)

plot.default(NA, xlab="$x$", ylab="$\\mu_S(x)$",
   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 <- c("$s_1$", "$s_2$", "$s_3$",
          "$s_4$", "$s_5$", "$s_6$");
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(5.55, X["knot.alpha"], sprintf("$\\alpha_0$"), pos=4, xpd=TRUE)

plot(X, add=TRUE, type='l', from=-1, to=6)
plot(X, add=TRUE, type='p', from=-1, to=6)
```



Please note that we use TEX commands in plot labels. They are interpreted by the `tikzDevice` package for R to generate beautiful figures, but setting this all up requires higher level of skills... and patience.

4 Basic computations on and characteristics of fuzzy numbers

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] -5 20
```

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

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

To compute arbitrary α-cuts we use:

```
alphacut(A, 0) # same as supp(A) (if alpha-cut generators are defined)
## [1] -5 20
alphacut(A, 1) # same as core(A)
```

```
## [1] 3 6
alphacut(A, c(0,0.5,1))
##           [,1]   [,2]
## [1,] -5.000000 20.000
## [2,] -4.583591  6.875
## [3,]  3.000000  6.000
```

Note that if we request to compute more than one α-cut at once, then a matrix with 2 columns (instead of a numeric vector of length 2) is returned. The `alphacut()` method may only be used when α-cut generators are provided by the user during the declaration of A, even for $\alpha = 0$ or $\alpha = 1$.

4.2 Evaluation of the membership function

If side generators are defined, we may calculate the values of the membership function at different points by calling:

```
evaluate(A, 1)
## [1] 0.9960291
evaluate(A, c(-3,0,3))
## [1] 0.8371139 0.9855322 1.0000000
evaluate(A, seq(-1, 2, by=0.5))
## [1] 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 [7].

$$\begin{aligned} \mathrm{EI}(A) &:= [\mathrm{EI}_L(A), \mathrm{EI}_U(A)] && (13) \\ &= \left[\int_0^1 A_L(\alpha)\,d\alpha, \int_0^1 A_U(\alpha)\,d\alpha\right]. && (14) \end{aligned}$$

To compute the expected interval of A we call:

```
expectedInterval(A)
## [1] -4.058824  8.800000
```

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:

$$\mathrm{EV}(A) := \frac{\mathrm{EI}_L(A) + \mathrm{EI}_U(A)}{2}. \tag{15}$$

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

```
expectedValue(A)
## [1] 2.370588
```

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:

$$\mathrm{EV}_w(A) := (1 - w)\mathrm{EI}_L(A) + w\mathrm{EI}_U(A). \tag{16}$$

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

```
weightedExpectedValue(A, 0.5) # equivalent to expectedValue(A)
## [1] 2.370588
weightedExpectedValue(A, 0.25)
## [1] -0.8441176
```

The *value* of A [5] is defined by:

$$\mathrm{val}(A) := \int_0^1 \alpha \left(A_L(\alpha) + A_U(\alpha)\right) d\alpha. \tag{17}$$

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 [3] is defined as:

$$\mathrm{width}(A) := \mathrm{EI}_U(A) - \mathrm{EI}_L(A). \tag{18}$$

An example:

```
width(A)
## [1] 12.85882
```

The *ambiguity* of A [5] is defined as:

$$\mathrm{amb}(A) := \int_0^1 \alpha \left(A_U(\alpha) - A_L(\alpha)\right) d\alpha. \tag{19}$$

```
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 Approximation of fuzzy numbers

5.1 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. [9]), d, defined by the equation:

$$d^2(A,B) = \int_0^1 (A_L(\alpha) - B_L(\alpha))^2 \, d\alpha + \int_0^1 (A_U(\alpha) - B_U(\alpha))^2 \, d\alpha. \tag{20}$$

```
distance(A, T1, type="Euclidean") # L2 distance (default)
## [1] 7.20234
distance(A, T1, type="EuclideanSquared") # Squared L2 distance (default)
## [1] 51.8737
```

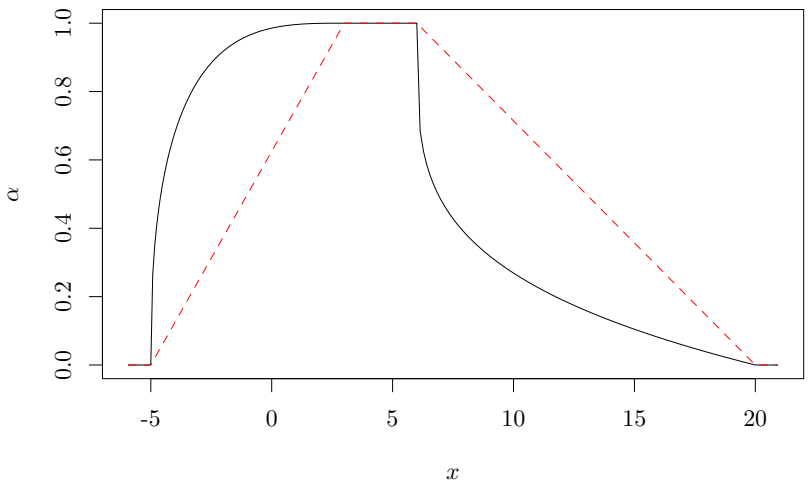
Types available `type`: `Euclidean`, `EuclideanSquared`...

5.2 Approximation by trapezoidal fuzzy numbers

5.2.1 Naïve approximation

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

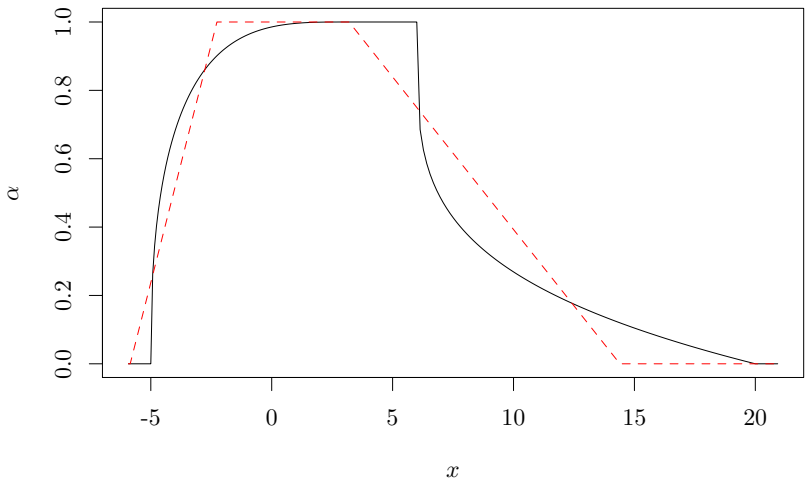
```
(T1 <- trapezoidalApproximation(A, method="Naive"))
## Trapezoidal fuzzy number with:
##    support=[-5,20],
##       core=[3,6].
distance(A, T1)
## [1] 5.761482

plot(A)
plot(T1, col="red", lty=2)
```



5.2.2 L_2-nearest approximation

The `"NearestEuclidean"` method gives the nearest L_2-approximation of A [2, Corollary 8].

```
(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
plot(A)
plot(T2, col="red", lty=2)
```



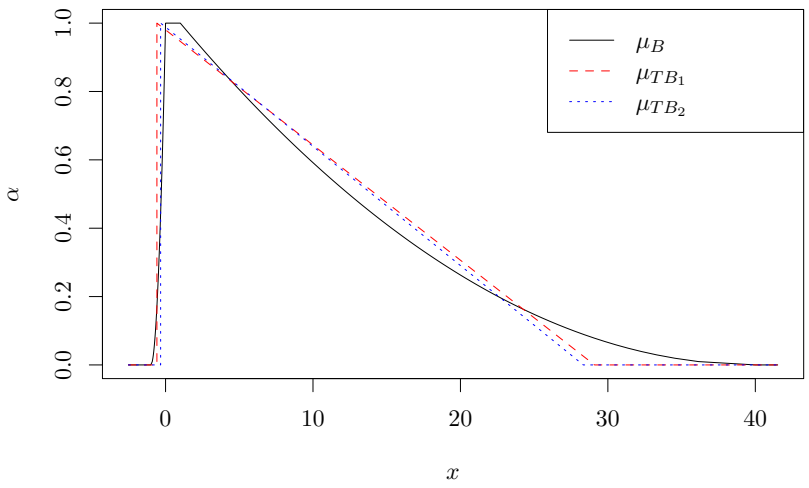
5.2.3 Expected interval preserving approximation

The `"ExpectedIntervalPreserving"` method gives the nearest L_2-approximation of A preserving the expected interval [1, 10, 14]. Note that if $\mathrm{amb}(A) \geq \mathrm{width}(A)/3$, then we get the same result as in the `"NearestEuclidean"` method.

```
(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
```

Unfortunately, for highly skewed membership functions this method (as well as the above one) 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. [11].

```
(B  <- FuzzyNumber(-1, 0, 1, 40,
   lower=function(x) sqrt(x),
   upper=function(x) 1-sqrt(x)))
## Fuzzy number with:
##    support=[-1,40],
##       core=[0,1].
(TB1 <- trapezoidalApproximation(B, "NearestEuclidean"))
## Trapezoidal fuzzy number with:
##    support=[-0.586667,29.0933],
##       core=[-0.586667,-0.586667].
(TB2 <- trapezoidalApproximation(B, "ExpectedIntervalPreserving"))
## Trapezoidal fuzzy number with:
##    support=[-0.333333,28.3333],
##       core=[-0.333333,-0.333333].
distance(B, TB1)
## [1] 1.938155
distance(B, TB2)
## [1] 1.992579
plot(B)
plot(TB1, col="red",  lty=2)
plot(TB2, col="blue", lty=3)
```

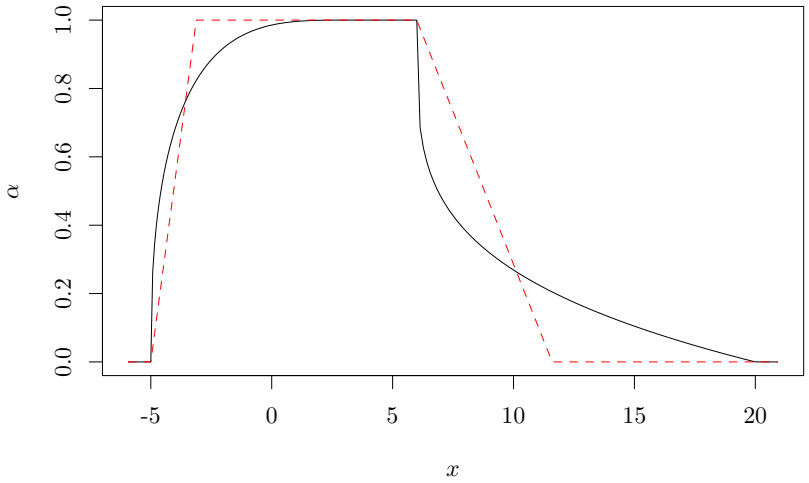


5.2.4 Approximation with restrictions on support and core

The `"SupportCoreRestricted"` method was proposed in [11]. 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 $\mathcal{T}(A)$.

```
(T4 <- trapezoidalApproximation(A, method="SupportCoreRestricted"))
## Trapezoidal fuzzy number with:
##    support=[-5,11.6],
```

```
##       core=[-3.11765,6].
distance(A, T4)
## [1] 2.603383
plot(A)
plot(T4, col="red", lty=2)
```

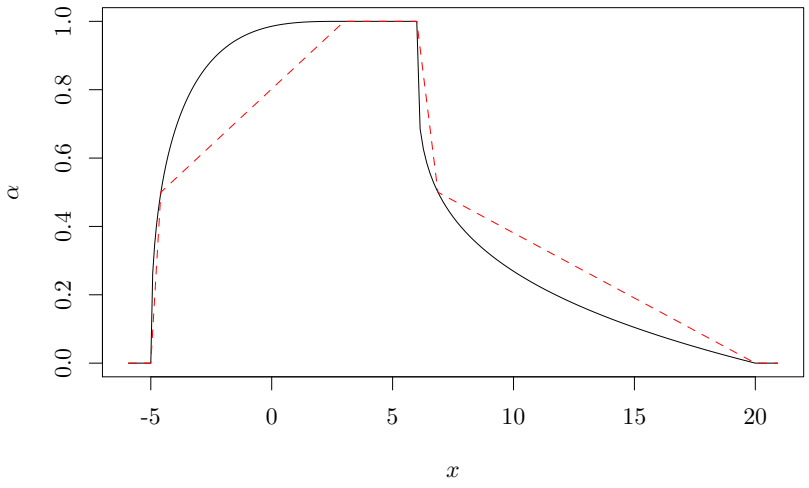


5.3 Approximation by piecewise linear fuzzy numbers

5.3.1 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 α-cuts.

```
P1 <- piecewiseLinearApproximation(A, method="Naive",
        knot.n=1, knot.alpha=0.5)
P1["allknots"]
##        alpha      left  right
## supp     0.0 -5.000000 20.000
## knot_1   0.5 -4.583591  6.875
## core     1.0  3.000000  6.000
print(distance(A, P1), 8)
## [1] 2.4753305
plot(A)
plot(P1, col="red", lty=2)
```



5.3.2 L_2-nearest approximation

Exact algorithm for fixed `knot.alpha`. For `knot.n==1` the method proposed in [4] is used.

```
system.time(P2 <- piecewiseLinearApproximation(A,
   method="NearestEuclidean", knot.n=1, knot.alpha=0.5))
```

```
##    user  system elapsed
##   0.010   0.001   0.082
```

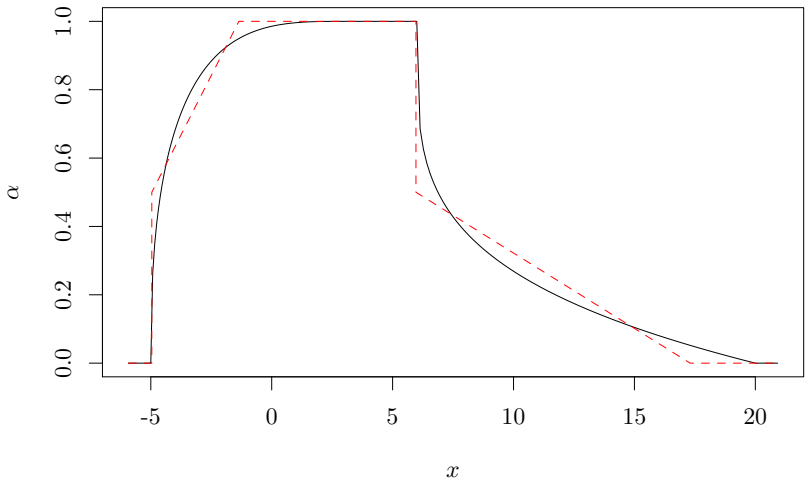
```
print(P2["allknots"], 6)
```

```
##        alpha     left  right
## supp     0.0 -4.95920 17.305
## knot_1   0.5 -4.95920  5.965
## core     1.0 -1.35769  5.965
```

```
print(distance(A, P2), 12)
```

```
## [1] 0.837361269482
```

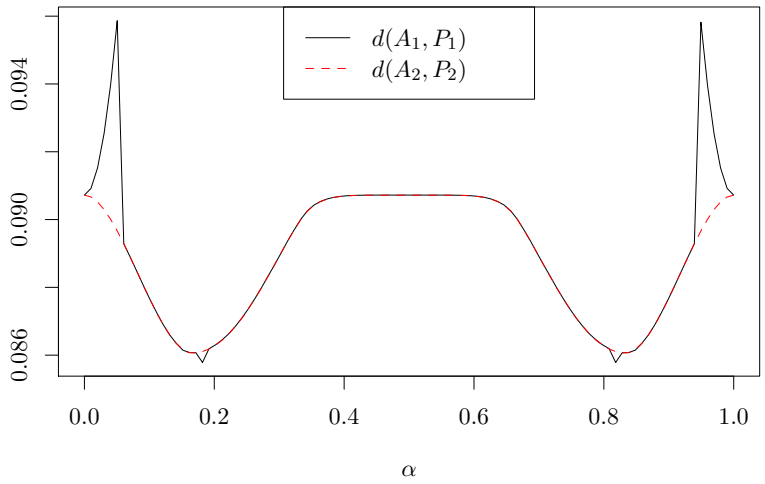
```
plot(A)
plot(P2, col="red", lty=2)
```



Beware of numerical error in integration e.g. due to discontinuity.

```
A1 <- FuzzyNumber(0,1,1,1,
         lower=function(a) floor(3*a)/3,
         upper=function(a) 1-a
) # no info on discontinuities

A2 <- DiscontinuousFuzzyNumber(0,1,1,1,
         lower=function(a) floor(3*a)/3,
         upper=function(a) 1-a,
         discontinuities.lower=c(0, 1/3, 2/3, 1),
         discontinuities.upper=numeric(0)
) # discontinuities info included

a <- seq(1e-9, 1-1e-9, length.out=100) # many alphas from (0,1)
d1 <- numeric(length(a)) # distances #1 (to be calculated)
d2 <- numeric(length(a)) # distances #2 (to be calculated)
for (i in 1:length(a))
{
   P1 <- piecewiseLinearApproximation(A1, method="NearestEuclidean",
            knot.n=1, knot.alpha=a[i])
   P2 <- piecewiseLinearApproximation(A2, method="NearestEuclidean",
            knot.n=1, knot.alpha=a[i])

   d1[i] <- distance(A1, P1)
   d2[i] <- distance(A2, P2)
}
## Warning:  max(abs(d[K]))==3.20058e-07
## Warning:  max(abs(d[K]))==3.78765e-07
## Warning:  max(abs(d[K]))==4.5e-09
## Warning:  max(abs(d[K]))==4.5e-09
matplot(a, cbind(d1, d2), type='l')
legend("top", c("d(A1,P1)", "d(A2,P2)"), lty=c(1,2), col=c(1,2))
```

We note that in the first case the distance for $\alpha = 0$ (trapezoidal approximation) is smaller than e.g. for $\alpha \simeq 0.05$, which, theoretically, is not possible. Moreover, the distance is not continuous at some α (but it is in theory).

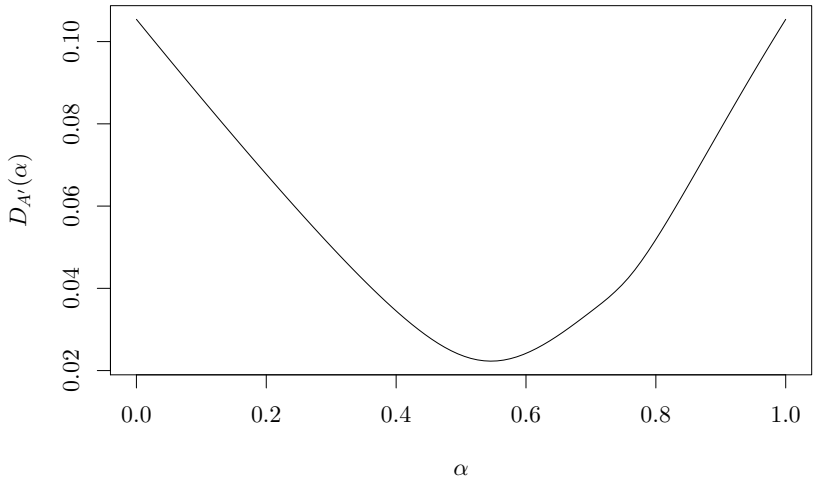


Finding best `knot.alpha` numerically. Consider the following fuzzy number A:

```
A1 <- FuzzyNumber(0,0,0,1,
   lower=function(a) a,
   upper=function(a) (1-a)^2)
```

Let us depict the error function (L_2 distance):

```
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 1:length(a))
{
   P1 <- piecewiseLinearApproximation(A1, method="NearestEuclidean",
            knot.n=1, knot.alpha=a[i])
   d[i] <- distance(A1, P1)
}
## Warning:  max(abs(d[K]))==7.14806e-08
## Warning:  max(abs(d[K]))==1.19959e-07
## Warning:  max(abs(d[K]))==3.57378e-07
## Warning:  max(abs(d[K]))==1.19209e-07
## Warning:  max(abs(d[K]))==2.38169e-07
plot(a, d, type='l', xlab=expression(alpha), ylab=expression(D[A](alpha)));
```



We may find best `knot.alpha` using numerical optimization. We only know that the distance function is continuous.

```
for (i in 1:5) # 5 iterations
{
   a0 <- runif(1,0,1); # random starting point
   optim(a0,
      function(a)
      {
         P1 <- piecewiseLinearApproximation(A1, method="NearestEuclidean",
                                            knot.n=1, knot.alpha=a)

         distance(A1, P1);
      }, method="L-BFGS-B", lower=1e-9, upper=1-1e-9) -> res;
   cat(sprintf("%.9f %6g %.9f %.9f\n", a0, res$counts[1], res$par, res$value));
}
```

```
## Warning:  max(abs(d[K]))==1.19959e-07
## Warning:  max(abs(d[K]))==3.57378e-07
## Warning:  max(abs(d[K]))==1.19209e-07
## Warning:  max(abs(d[K]))==2.38169e-07
## Warning:  max(abs(d[K]))==1.19959e-07
## Warning:  max(abs(d[K]))==3.57378e-07
## Warning:  max(abs(d[K]))==1.19209e-07
## Warning:  max(abs(d[K]))==2.38169e-07
## 0.187746551     19 0.546146113 0.022294007
## 0.687958484     56 0.545856036 0.022293594
## 0.629934967     14 0.546148523 0.022294007
## 0.789465386     60 0.545856274 0.022293594
## 0.767003964     35 0.545856348 0.022293594
```

Approximate algorithm for fixed `knot.alpha`. This method uses a constrained version of the Nelder-Mead algorithm. The procedure minimizes the target function numerically by calling the `optim()` function. There is thus no guarantee that it will find to the global minimum (it may fall into a neighborhood of a local minimum or even fail to converge to it). However, this

approach may be used for any number of knots.

```
system.time(P3 <- piecewiseLinearApproximation(A,
   method="ApproximateNearestEuclidean", knot.n=1, knot.alpha=0.5))
```

```
##    user  system elapsed
##   1.346   0.000   1.377
```

```
print(P3["allknots"], 6)
```

```
##        alpha     left    right
## supp     0.0 -4.95921 17.30510
## knot_1   0.5 -4.95921  5.96493
## core     1.0 -1.35761  5.96493
```

```
print(distance(A, P3), 12)
```

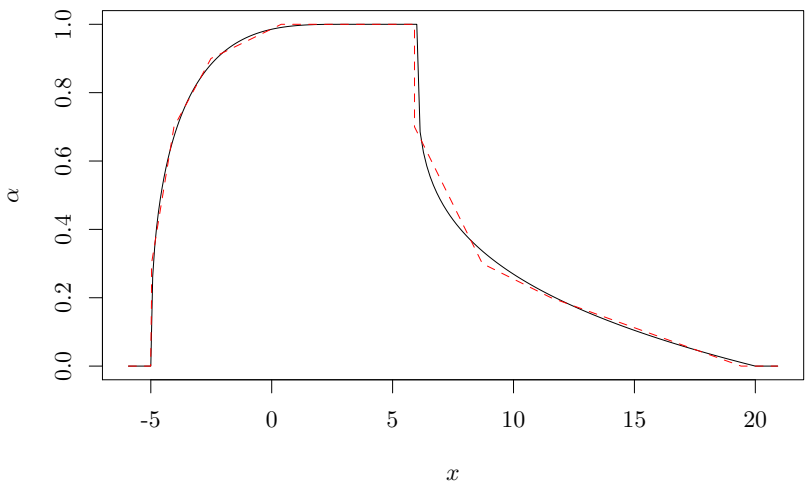
```
## [1] 0.837361274848
```

Please note that a call to this method may be time-consuming.

```
system.time(P4 <- piecewiseLinearApproximation(A,
   method="ApproximateNearestEuclidean", knot.n=4,
   knot.alpha=c(0.2, 0.3, 0.7, 0.9), verbose=TRUE))
```

```
## Pass 1a,1b,DONE.
##    user  system elapsed
##  35.515   0.014  36.227
```

```
plot(A)
plot(P4, col="red", lty=2)
```



If the method fails to converge, you may try to call it e.g. with the `optim.control=list(maxit=5000)` parameter to allow for greater number of iterations.

6 NEWS/CHANGELOG

```
FuzzyNumbers Package NEWS

*******************************************************************************
```

```
0.02 /2012-12-27/

* approx.invert(): a new function to find the numerical
  inverse of a given side/alpha-cut generating function
  (by default via Hermite monotonic spline interpolation)

* convert.side(), convert.alpha():
  new functions to convert sides and alpha cuts
  to side generating funs and alpha cut generators

* FuzzyNumber class validity check for lower, upper, left, right:
   * checks whether each function is properly vectorized
      and gives numeric results
   * does not check for the number of formal arguments,
      but just uses the first from the list

* suggests `testthat`

* each object has been documented

*******************************************************************************

0.01 /July 2012/

* initial release
```

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 Przemyslaw Grzegorzewski, Lucian Coroianu, and Pablo Villacorta Iglesias for stimulating discussion.

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] Ban A.I., On the nearest parametric approximation of a fuzzy number – Revisited, *Fuzzy Sets and Systems* **160**, 2009, pp. 3027–3047.

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

[4] Coroianu L., Gagolewski M., Grzegorzewski P., *Nearest Piecewise Linear Approximation of Fuzzy Numbers*, to appear in *Fuzzy Sets and Systems*, 2013.

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

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

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

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

[9] Grzegorzewski P., Metrics and orders in space of fuzzy numbers, *Fuzzy Sets and Systems* **97**, 1998, pp. 83–94.

[10] 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.

[11] 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.

[12] Klir G.J., Yuan B., *Fuzzy sets and fuzzy logic. Theory and applications*, Prentice Hall, New Jersey, 1995.

[13] Stefanini L., Sorini L., *Fuzzy arithmetic with parametric LR fuzzy numbers*, In: *Proc. IFSA/EUSFLAT 2009*, pp. 600–605.

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