Package ‘FuzzyNumbers.Ext.2’

September 5, 2017

Type Package
Title Apply Two Fuzzy Numbers on a Monotone Function
Version 3.2
Date 2017-09-02
Author Abbas Parchami (Department of Statistics, Faculty of Mathematics and Computer, Shahid Bahonar University of Kerman, Kerman, Iran)
Maintainer Abbas Parchami <parchami@uk.ac.ir>
Description One can easily draw the membership function of \( f(x,y) \) by package ‘FuzzyNumbers.Ext.2’ in which \( f(\ldots) \) is supposed monotone and \( x \) and \( y \) are two fuzzy numbers. This work is possible using function \texttt{f2apply()} which is an extension of function \texttt{fapply()} from Package ‘FuzzyNumbers’ for two-variable monotone functions. Moreover, this package has the ability of computing the core, support and alpha-cuts of the fuzzy-valued final result.
License LGPL (>= 3)
Imports FuzzyNumbers
NeedsCompilation no
Repository CRAN
Date/Publication 2017-09-05 07:29:09 UTC

R topics documented:

- FuzzyNumbers.Ext.2-package
- f2apply
- is.decreasing
- is.decreasing.on.x
- is.decreasing.on.y
- is.increasing
- is.increasing.on.x
- is.increasing.on.y

Index 16
**FuzzyNumbers.Ext.2-package**

*Apply Two Fuzzy Numbers on a Monotone Function*

**Description**

One can easily draw the membership function of f(x,y) by package 'FuzzyNumbers.Ext.2' in which f(...) is supposed monotone and x and y are two fuzzy numbers. This work is possible using function f2apply() which is an extension of function fapply() from Package 'FuzzyNumbers' for two-variable monotone functions. Moreover, this package has the ability of computing the core, support and alpha-cuts of the fuzzy-valued final result.

**Details**

The main goal of Package FuzzyNumbers.Ext.2 is apply two fuzzy numbers x and y into a monotone two-variable function f(x, y) which is possible by using function f2apply.

**Author(s)**

Abbas Parchami <parchami@uk.ac.ir>

**References**

Gagolewski, M., Caha, J., FuzzyNumbers Package: Tools to Deal with Fuzzy Numbers in R. R package version 0.4-1, 2015. https://cran.r-project.org/web/packages=FuzzyNumbers


**See Also**

FuzzyNumbers
**f2apply**

*Apply a two-variable function on two fuzzy numbers*

**Description**

Suppose that we are going to put two fuzzy numbers \(x\) and \(y\) into the monotonic two-variable function \(f(x, y)\). A usual approach is using Zadeh’s extension Principle which has a complex computation. Function \(f2apply\) applies easily two fuzzy numbers to a monotonic two-variable function. Although the theory of \(f2apply\) computation is based on the Zadeh’s extension Principle, but it works with the \(\alpha\)-cuts of two inputted fuzzy numbers for all \(\alpha \in (0, 1]\). It must be mentioned that the ability of computing \(\alpha\)-cuts of the result is added to the Version 2.0.

**Usage**

```r
f2apply(x, y, fun, knot.n=10, I.0.plot="TRUE", ...)
```

**Arguments**

- **x**
  - the first fuzzy number, which must be according to the format of FuzzyNumbers package
- **y**
  - the second fuzzy number, which must be according to the format of FuzzyNumbers package
- **fun**
  - a two-variable function which is monotone function on the supports of \(x\) and \(y\) fuzzy numbers
- **knot.n**
  - the number of knots; see package FuzzyNumbers
- **I.0.plot**
  - a logical argument with default \(\text{TRUE}\). If \(\text{I.0.plot}=\text{TRUE}\), then three membership functions of \(x, y\) (Inputted fuzzy numbers) and \(f(x, y)\) (Outputted fuzzy number) are drawn in a figure. If \(\text{I.0.plot}=\text{FALSE}\), then just the membership function of Outputted fuzzy number \(f(x, y)\) will be shown in figure.
- **...**
  - additional arguments passed from plot

**Value**

This function returns piecewise linear fuzzy number \(f(x, y)\) and also plot the result.

- **fun.rep**
  - describes the monotonic behavior of the considered function
- **cuts**
  - returns the \(\alpha\)-cuts of the computed fuzzy number \(f(x, y)\)
- **core**
  - returns the core of the computed fuzzy number \(f(x, y)\)
- **support**
  - returns the support of the computed fuzzy number \(f(x, y)\)

**Note**

\(f2apply\) is an extended version of \(fapply\) from package FuzzyNumbers. The duty of functions \(fapply\) and \(f2apply\) are applying one-variable and two-variable function on fuzzy numbers. Two imported fuzzy numbers into \(f2apply\) must be piecewised by PiecewiseLinearFuzzyNumber function in package FuzzyNumbers. Moreover, the considered function \(f(x, y)\) must be monotone on \(x\) and \(y\).
Author(s)
Abbas Parchami

References
Gagolewski, M., Caha, J., FuzzyNumbers Package: Tools to Deal with Fuzzy Numbers in R. R package version 0.4-1, 2015. https://cran.r-project.org/web/packages=FuzzyNumbers

See Also
See PiecewiseLinearFuzzyNumber, as.PiecewiseLinearFuzzyNumber and piecewiseLinearApproximation from package FuzzyNumbers.

Examples

library(FuzzyNumbers)  # For Loud 'FuzzyNumbers' package, after its instalation

# Example 1: Four different cases of function (in respect to increasing/decreasing on x and y)
x = TriangularFuzzyNumber(1,2,5)
y = TrapezoidalFuzzyNumber(3,4,5,6)

g1 = function(x,y) 2*x+y
f2apply(x, y, g1, knot.n=5, type="l", I.O.plot=TRUE)
f2apply(x, y, g1, knot.n=10, xlim=c(0,18), col=4, type="b", I.O.plot=FALSE)
plot(2*x+y, col=2, lty=4, lwd=3, add=TRUE)  #Compare the result from "FuzzyNumbers" package

g2 = function(x,y) -2*pnorm(x)+y
f2apply(x, y, g2, type="b")

g3 = function(x,y) 2*x-punif(y, min=1, max=8)
f2apply(x, y, g3, type="l")

g4 = function(x,y) -2*x-y^3
f2apply(x, y, g4, knot.n=20, type="b")

# Example 2:
knot.n = 10
A <- FuzzyNumber(-1, .5, 1, 3,
  lower=function(alpha) qbeta(alpha,0.4,3),
  upper=function(alpha) (1-alpha)*4
)
B = PowerFuzzyNumber(1, 2, 2.5, 4, p.left=2, p.right=0.5)
fapply(A, B, function(x, y) -2*x - y^3, knot.n=knot.n, type="l", col=2, lty=5, lwd=3, I.O.plot=FALSE)
fapply(A, B, function(x, y) -2*x - y^3, knot.n=knot.n, type="l", col=2, lty=5, lwd=3)

# As another example, change the function and work with the cuts of result:
Result <- fapply(A, B, function(x, y) abs(y*x-10), knot.n=knot.n, type="l", I.O.plot=TRUE, col=3, lwd=2)

class(Result)

# The result of alphacut for alpha=0.444:
Result$cuts["0.444",] # Or equivalently,
Result$cuts[6,]

# Upper bounds of alphacuts:
Result$cuts[,"U"] # Or equivalently,
Result$cuts[,2]

# The core of the result:
Result$core

# The support of the result:
Result$support

# Example 3:
knot.n = 10
x = PowerFuzzyNumber(0, 1, 1, 1.3, p.left=1, p.right=1)
y = PowerFuzzyNumber(3, 4, 4, 6, p.left=1, p.right=1)
f = function(x, y) 3*x - 2*y
fapply(x, y, f, knot.n=knot.n, type="l", I.O.plot=TRUE)

g = function(x, y) exp(x^2) + 3*log(sqrt(y+4))
fapply(x, y, g, knot.n=knot.n, type="l", I.O.plot=TRUE)

# Example 4:
knot.n = 20
A = PowerFuzzyNumber(0.1, 0.5, 0.6, p.left=2, p.right=0.5)
B <- FuzzyNumber(0.5, 0.6, 0.7, 0.9,
    lower=function(alpha) qbeta(alpha, 0.4, 3),
    upper=function(alpha) (1-alpha)^4)

fun1 <- function(x, y) qnorm(x) - qgamma(y, 2, 4)
fapply(A, B, fun1, knot.n=knot.n, type="l", I.O.plot=TRUE, col=2, lwd=2)

fun2 <- function(x, y) 0.3*sin(qnorm(x)) + tan(qgamma(y, 2, 4))
fapply(A, B, fun2, knot.n=knot.n, type="l", I.O.plot=TRUE)

# Example 5: It may be one of considered inputs are crisp.
knot.n = 10
A = 27
B = PowerFuzzyNumber(1, 2, 2.5, 4, p.left=2, p.right=0.5)
f2apply(A, B, function(x,y) -2*x+y^3, knot.n=knot.n, I.O.plot=TRUE)
f2apply(x=4, y=3, function(x,y) sqrt(x)*y^2, knot.n=knot.n, I.O.plot=TRUE)
f2apply(x=4, y=TriangularFuzzyNumber(2,3,5), function(x,y) sqrt(x)-y^2,knot.n=knot.n,I.O.plot=TRUE)
f2apply(x=TriangularFuzzyNumber(2,4,6), y=3, function(x,y) sqrt(x)-y^2,knot.n=knot.n,I.O.plot=TRUE)
f2apply(x=TriangularFuzzyNumber(2,4,6), y=TriangularFuzzyNumber(2,3,5), function(x,y) sqrt(x)-y^2,
  knot.n=knot.n, I.O.plot=TRUE)

## The function is currently defined as
function (x, y, fun, knot.n = 10, I.O.plot = "TRUE", ...)
{
  x.input <- x
  y.input <- y
  if (class(x) == "numeric") {
    x <- x.input.fuzzy <- TriangularFuzzyNumber(x, x, x)
  }
  if (class(x) == "TriangularFuzzyNumber" | class(x) == "TrapezoidalFuzzyNumber") {
    x.input.fuzzy <- x
    x <- as.PiecewiseLinearFuzzyNumber(x, knot.n)
  }
  if (class(x) == "FuzzyNumber" | class(x) == "PowerFuzzyNumber" |
      class(x) == "PiecewiseLinearFuzzyNumber" ){
    x.input.fuzzy <- x
    x <- piecewiseLinearApproximation(x, method = "Naive")
  }
  if (class(y) == "numeric") {
    y <- y.input.fuzzy <- TriangularFuzzyNumber(y, y, y)
  }
  if (class(y) == "TriangularFuzzyNumber" | class(y) == "TrapezoidalFuzzyNumber") {
    y.input.fuzzy <- y
    y <- as.PiecewiseLinearFuzzyNumber(y, knot.n)
  }
  if (class(y) == "FuzzyNumber" | class(y) == "PowerFuzzyNumber" |
      class(y) == "PiecewiseLinearFuzzyNumber" ){
    y.input.fuzzy <- y
    y <- piecewiseLinearApproximation(y, method = "Naive")
  }
  step.x = length(supp(x))/30
  step.y = length(supp(y))/30
  if (class(x.input) == "numeric") {
    is.inc.on.x <- TRUE
    is.dec.on.x <- FALSE
  }
  else {
    is.inc.on.x = is.increasing.on.x(fun, x.bound = supp(x),
      y.bound = supp(y), step.x)
    is.dec.on.x = is.decreasing.on.x(fun, x.bound = supp(x),
      y.bound = supp(y), step.x)
  }
  if (class(y.input) == "numeric") {
    is.inc.on.y <- TRUE
    is.dec.on.y <- FALSE
  }
else {
  is.inc.on.y = is.increasing.on.y(fun, x.bound = supp(x),
    y.bound = supp(y), step.y)
  is.dec.on.y = is.decreasing.on.y(fun, x.bound = supp(x),
    y.bound = supp(y), step.y)
}

if ((is.inc.on.x == TRUE) & (is.inc.on.y == TRUE)) {
  fun.rep = "fun is an increasing function from x and y on introduced bounds"
  L.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "L"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "L")
  U.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "U"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "U")
  result = c(L.result, U.result[length(U.result):1])
} else {
  if ((is.dec.on.x == TRUE) & (is.inc.on.y == TRUE)) {
    fun.rep = "fun is a decreasing function on x and increasing function on y on introduced bounds"
    L.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "L"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "L")
    U.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "U"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "U")
    result = c(L.result, U.result[length(U.result):1])
  } else {
    if ((is.inc.on.x == TRUE) & (is.dec.on.y == TRUE)) {
      fun.rep = "fun is an increasing function on x and decreasing function on y on introduced bounds"
      L.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "L"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "L")
      U.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "U"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "U")
      result = c(L.result, U.result[length(U.result):1])
    } else {
      if ((is.dec.on.x == TRUE) & (is.dec.on.y == TRUE)) {
        fun.rep = "fun is a decreasing function from x and y on introduced bounds"
        L.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "L"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "L")
        U.result = fun(alphacut(x.input.fuzzy, seq(0, 1, len = knot.n))[, "U"], alphacut(y.input.fuzzy, seq(0, 1, len = knot.n))[, "U")
        result = c(L.result, U.result[length(U.result):1])
      } else {
        return(print("fun is not a monotonic function on x and y for the introduced bounds. Therefore this function is not applicable for computation."))
      }
    }
  }
}
is.decreasing

is.decreasing

Diagnosis a decreasing function

Description

is.decreasing tests if the introduced one-variable function is decreasing (or in fact, non-increasing) on the considered x.bound or not. In other words, is.decreasing returns TRUE if the introduced function is decreasing on the considered x.bound; and it returns FALSE otherwise. The goal of introducing function is.decreasing in package FuzzyNumbers.Ext.2 is using in function f2apply.
is.decreasing

Usage

is.decreasing(fun, x.bound = c(-1, 1), step = 0.01)

Arguments

fun          a one-variable R function
x.bound     a vector with two real ordered elements which determine a bound on x-axis for
            checking the monotonic of the considered function
step        a positive real-valued number which determine the increment of the considered
            sequence for checking the monotonic of the considered function. The default
            of step is 0.01. Increasing step value can cause the decreasing the time of
            computation and also cause the decreasing the precision of the calculations.

Value

TRUE for decreasing one-variable functions on the considered x.bound; otherwise FALSE

See Also

is.increasing

Examples

is.decreasing(fun=function(x) -2*x+10, x.bound=c(4,6), step=.1)

g = function(x) x^3
is.decreasing(g, x.bound=c(-24,6))

## The function is currently defined as
function (fun, x.bound = c(-1, 1), step = 0.01)
{
    x = seq(x.bound[1], x.bound[2], by = step)
    i = 1
    while (fun(x[i]) >= fun(x[i + 1])) {
        if (i < length(x) - 1) {
            i <- i + 1
        } else (return(TRUE))
    }
    return(FALSE)
}
is.decreasing.on.x  Diagnosis a decreasing two-variable function toward x

Description

is.decreasing.on.x tests for any fixed y from y.bound, if the introduced two-variable function $f(x, y)$ is decreasing toward x on the considered x.bound or not. In other words, is.decreasing.on.x returns TRUE if the introduced function $f(x, y)$ is decreasing function of $x$ on the considered x.bound (for any fixed $y$ in y.bound); and it returns FALSE otherwise. The goal of introducing function is.increasing.on.x in package FuzzyNumbers.Ext.2 is using in function f2apply.

Usage

is.decreasing.on.x(fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01)

Arguments

- **fun**: a two-variable R function
- **x.bound**: a vector with two real ordered elements which determine a bound on x-axis for checking the monotonic
- **y.bound**: a vector with two real ordered elements which determine a bound on y-axis for checking the monotonic
- **step**: a positive real-valued number which determine the increment of the considered sequence for checking the monotonic of the considered function. The default of step is 0.01. Increasing step value can cause the decreasing the time of computation and also cause the decreasing the precision of the calculations.

Value

TRUE for two-variable function $f(x,y)$ which is decreasing toward x on x.bound (for any fixed y from y.bound); and otherwise FALSE

See Also

is.decreasing, is.decreasing.on.y

Examples

```r
is.decreasing.on.x(fun=function(x,y) 2*x+y, x.bound=c(0,2), y.bound=c(1,2), step=.2)

f = function(x,y) -x^2+y
is.decreasing.on.x(f, x.bound=c(0,2), y.bound=c(0,2))
is.decreasing.on.x(f, x.bound=c(-2,2), y.bound=c(0,2))
```

## The function is currently defined as

```r
function (fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01)
{
```
is.decreasing.on.y

```r
y = seq(y.bounds[1], y.bounds[2], by = step)
for (i in 1:length(y)) {
  g = function(x) fun(x, y[i])
  if (is.decreasing(g, x.bounds, step) == FALSE) {
    return(FALSE)
  }
}
return(TRUE)
```

---

**Description**

`is.decreasing.on.y` tests for any fixed `x` from `x.bounds`, if the introduced two-variable function `f(x, y)` is decreasing toward `y` on the considered `y.bounds` or not. In other words, `is.decreasing.on.y` returns `TRUE` if the introduced function `f(x, y)` is decreasing function of `y` on the considered `y.bounds` (for any fixed `x` in `x.bounds`); and it returns `FALSE` otherwise. The goal of introducing function `is.decreasing.on.y` in package `FuzzyNumbers.Ext.2` is using in function `f2apply`.

**Usage**

```r
is.decreasing.on.y(fun, x.bounds = c(-1, 1), y.bounds = c(-1, 1), step = 0.01)
```

**Arguments**

- **fun** a two-variable R function
- **x.bounds** a vector with two real ordered elements which determine a bound on x-axis for checking the monotonic
- **y.bounds** a vector with two real ordered elements which determine a bound on y-axis for checking the monotonic
- **step** a positive real-valued number which determine the increment of the considered sequence for checking the monotonic of the considered function. The default of `step` is 0.01. Increasing `step` value can cause the decreasing the time of computation and also cause the decreasing the precision of the calculations.

**Value**

`TRUE` for two-variable function `f(x, y)` which is decreasing toward `y` on `y.bounds` (for any fixed `x` from `x.bounds`); and otherwise `FALSE`

**See Also**

`is.decreasing, is.decreasing.on.x`
Examples

```r
is.decreasing.on.y(fun=function(x,y) 2*x-y, x.bound=c(0,2), y.bound=c(1,2), step=.2)
```

```r
H = function(x,y) pnorm(x)-pnorm(y)
is.decreasing.on.x(H)
is.decreasing.on.y(H)
```

```r
## The function is currently defined as
function (fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01)
{
  x = seq(x.bound[1], x.bound[2], by = step)
  for (i in 1:length(x)) {
    g = function(y) fun(x[i], y)
    if (is.decreasing(g, y.bound, step) == FALSE) {
      return(FALSE)
    }
  }
  return(TRUE)
}
```

---

**is.increasing**

**Diagnosis an increasing function**

**Description**

`is.increasing` tests if the introduced one-variable function is increasing (or in fact, non-decreasing) on the considered `x.bound` or not. In other words, `is.increasing` returns `TRUE` if the introduced function is increasing on the considered `x.bound`; and it returns `FALSE` otherwise. The goal of introducing function `is.increasing` in package `FuzzyNumbers.Ext.2` is using in function `f2apply`.

**Usage**

```r
is.increasing(fun, x.bound = c(-1, 1), step = 0.01)
```

**Arguments**

- `fun`: a one-variable R function
- `x.bound`: a vector with two real ordered elements which determine a bound on x-axis for checking the monotonic of the considered function
- `step`: a positive real-valued number which determine the increment of the considered sequence for checking the monotonic of the considered function. The default of step is 0.01. Increasing step value can cause the decreasing the time of computation and also couse the decreasing the precision of the calculations.

**Value**

TRUE for increasing one-variable functions on the considered `x.bound`; otherwise FALSE
is.increasing.on.x

See Also

is.decreasing

Examples

is.increasing(fun=function(x) 2*x, x.bound=c(4,6), step=.1)

g = function(x) x^2
is.increasing(g, x.bound=c(-24,6), step=.01)

h = function(x) x^5
is.increasing(h, c(-24,6), .01)
curve(h(x), xlim=c(-2,2))

## The function is currently defined as
function (fun, x.bound = c(-1, 1), step = 0.01)
{
  x = seq(x.bound[1], x.bound[2], by = step)
  i = 1
  while (fun(x[i]) <= fun(x[i+1])) {
    if (i < length(x) - 1) {
      i <- i + 1
    } else (return(TRUE))
  }
  return(FALSE)
}

is.increasing.on.x  Diagnosis an increasing two-variable function toward x

Description

is.increasing.on.x tests for any fixed y from y.bound, if the introduced two-variable function
f(x, y) is increasing toward x on the considered x.bound or not. In other words, is.increasing.on.x
returns TRUE if the introduced function f(x, y) is increasing function of x on the considered x.bound
(for any fixed y in y.bound); and it returns FALSE otherwise. The goal of introducing function
is.increasing.on.x in package FuzzyNumbers.Ext.2 is using in function f2apply.

Usage

is.increasing.on.x(fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01)

Arguments

  fun  a two-variable R function
  x.bound  a vector with two real ordered elements which determine a bound on x-axis for checking the monotonic
is.increasing.on.y

Description

is.increasing.on.y tests for any fixed x from x.bound, if the introduced two-variable function f(x, y) is increasing toward y on the considered y.bound or not. In other words, is.increasing.on.y returns TRUE if the introduced function f(x, y) is increasing function of y on the considered y.bound (for any fixed x in x.bound); and it returns FALSE otherwise. The goal of introducing function is.increasing.on.y in package FuzzyNumbers.Ext.2 is using in function f2apply.
Usage

is.increasing.on.y(fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01)

Arguments

fun
a two-variable R function
x.bound
a vector with two real ordered elements which determine a bound on x-axis for checking the monotonic
y.bound
a vector with two real ordered elements which determine a bound on y-axis for checking the monotonic
step
a positive real-valued number which determine the increment of the considered sequence for checking the monotonic of the considered function. The default of step is 0.01. Increasing step value can cause the decreasing the time of computation and also cause the decreasing the precision of the calculations.

Value

TRUE for two-variable function f(x,y) which is increasing toward y on y.bound (for any fixed x from x.bound); and otherwise FALSE

See Also

is.increasing, is.increasing.on.x

Examples

is.increasing.on.y(fun=function(x,y) 2*x+y, x.bound=c(0,2), y.bound=c(1,2), step=.2)

f = function(x,y) 5*x+y^2
is.increasing.on.y(f, x.bound=c(0,2), y.bound=c(0,2))
is.increasing.on.y(f, x.bound=c(-2,2), y.bound=c(0,2))
is.increasing.on.y(f, x.bound=c(0,2), y.bound=c(-2,2))

H = function(x,y) pnorm(x)+y^2
is.increasing.on.x(H)
is.increasing.on.y(H)
is.increasing.on.y(H, x.bound=c(-3,3), y.bound=c(0,3))

## The function is currently defined as
function (fun, x.bound = c(-1, 1), y.bound = c(-1, 1), step = 0.01) {
    x = seq(x.bound[1], x.bound[2], by = step)
    for (i in 1:length(x)) {
        g = function(y) fun(x[i], y)
        if (is.increasing(g, y.bound, step) == FALSE) {
            return(FALSE)
        }
    }
    return(TRUE)
}
Index

*Topic **FuzzyNumbers**
  f2apply, 3
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **f2apply**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **is.decreasing.on.x**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **is.decreasing.on.y**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **is.decreasing**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **is.increasing.on.x**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10

*Topic **is.increasing**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14

*Topic **monoton function**
  f2apply, 3
  FuzzyNumbers.Ext.2-package, 2
  is.decreasing, 8
  is.decreasing.on.x, 10
  is.decreasing.on.y, 11
  is.increasing, 12
  is.increasing.on.x, 13
  is.increasing.on.y, 14
is.decreasing.on.y, 11
is.increasing, 12
is.increasing.on.x, 13
is.increasing.on.y, 14