The kappalab Package

September 1, 2007

Version 0.4-0

Date 2007-31-09

Title Non-additive measure and integral manipulation functions

Author Michel Grabisch, Ivan Kojadinovic, Patrick Meyer.

Maintainer Ivan Kojadinovic <ivan@stat.auckland.ac.nz>

Description Kappalab, which stands for "laboratory for capacities", is an S4 tool box for capacity (or non-additive measure, fuzzy measure) and integral manipulation on a finite setting. It contains routines for handling various types of set functions such as games or capacities. It can be used to compute several non-additive integrals: the Choquet integral, the Sugeno integral, and the symmetric and asymmetric Choquet integrals. An analysis of capacities in terms of decision behavior can be performed through the computation of various indices such as the Shapley value, the interaction index, the orness degree, etc. The well-known Möbius transform, as well as other equivalent representations of set functions can also be computed. Kappalab further contains seven capacity identification routines: three least squares based approaches, a method based on linear programming, a maximum entropy like method based on variance minimization, a minimum distance approach and an unsupervised approach grounded on parametric entropies. The functions contained in Kappalab can for instance be used in the framework of multicriteria decision making or cooperative game theory.

Depends R(>= 2.1.0), methods, lpSolve, quadprog, kernlab

Encoding latin1

License CeCILL 2 (GNU GPL 2 compatible)

URL http://www.stat.auckland.ac.nz/~ivan/kappalab

R topics documented:

`Choquet.integral-methods`
Choquet integral

Description

Computes the Choquet integral of a discrete function with respect to a game. The game can be given either under the form of an object of class `game`, `card.game` or `Mobius.game`. If the integrand is not positive, this function computes what is known as the *asymmetric Choquet integral*.

Methods

object = "Mobius.game", f = "numeric" The Choquet integral of `f` is computed from the Mobius transform of a game.

object = "game", f = "numeric" The Choquet integral of `f` is computed from a game.

object = "card.game", f = "numeric" The Choquet integral of `f` is computed from a cardinal game.

References

G. Choquet (1953), *Theory of capacities*, Annales de l'Institut Fourier 5, pages 131-295.

D. Denneberg (2000), *Non-additive measure and integral, basic concepts and their role for applications*, in: M. Grabisch, T. Murofushi, and M. Sugeno Eds, Fuzzy Measures and Integrals: Theory and Applications, Physica-Verlag, pages 42-69.

M. Grabisch, T. Murofushi, M. Sugeno Eds (2000), *Fuzzy Measures and Integrals: Theory and Applications*, Physica-Verlag.

M. Grabisch and Ch. Labreuche (2002), *The symmetric and asymmetric Choquet integrals on finite spaces for decision making*, Statistical Papers 43, pages 37-52.

M. Grabisch (2000), *A graphical interpretation of the Choquet integral*, IEEE Transactions on Fuzzy Systems 8, pages 627-631.

J.-L. Marichal (2000), *An axiomatic approach of the discrete Choquet integral as a tool to aggregate interacting criteria*, IEEE Transactions on Fuzzy Systems 8:6, pages 800-807.

Murofushi and M. Sugeno (1993), *Some quantities represented by the Choquet integral*, Fuzzy Sets and Systems 56, pages 229-235.

Murofushi and M. Sugeno (2000), *Fuzzy measures and fuzzy integrals*, in: M. Grabisch, T. Murofushi, and M. Sugeno Eds, Fuzzy Measures and Integrals: Theory and Applications, Physica-Verlag, pages 3-41.

See Also

`game-class`,
`Mobius.game-class`,
`card.game-class`.

Examples

```
## a normalized capacity
mu <- capacity(c(0:13/13,1,1))

## and its Möbius transform
a <- Mobius(mu)

## a discrete positive function f
f <- c(0.1,0.9,0.3,0.8)

## the Choquet integral of f w.r.t mu
Choquet.integral(mu,f)
Choquet.integral(a,f)

## a similar example with a cardinal capacity
mu <- uniform.capacity(4)
Choquet.integral(mu,f)
```

Mobius-methods *The Möbius transform*

Description

Computes the Mobius transform of a set function. The Mobius transform is the inverse of the zeta transform.

Methods

object = "capacity" Returns an object of class `Mobius.capacity`.

object = "card.set.func" Returns an object of class `Mobius.card.set.func`.

object = "game" Returns an object of class `Mobius.game`.

object = "set.func" Returns an object of class `Mobius.set.func`.

References

G-C. Rota (1964), *On the foundations of combinatorial theory. I. Theory of Mobius functions*, Z. Wahrscheinlichkeitstheorie und Verw. Gebiete 2, pages 340-368.

A. Chateauneuf and J-Y. Jaffray (1989), *Some characterizations of lower probabilities and other monotone capacities through the use of Mobius inversion*, Mathematical Social Sciences 17, pages 263-283.

M. Grabisch, J-L. Marichal and M. Roubens (2000), *Equivalent representations of set functions*, Mathematics of Operations Research 25:2, pages 157-178.

See Also

Mobius.capacity-class,
Mobius.card.set.func-class,
Mobius.game-class,
Mobius.set.func-class,
capacity-class,
card.set.func-class,
game-class,
set.func-class,
zeta-methods.

Examples

```
## a capacity
mu <- capacity(0:15)
mu

## its Möbius transform
a <- Mobius(mu)
a

## its zeta transform
zeta(a)

## a similar example with a game object
mu <- game(c(0,-2:12))
mu
Mobius(mu)
zeta(Mobius(mu))

## a similar example with a set.func object
mu <- set.func(-7:8)
mu
Mobius(mu)
zeta(Mobius(mu))

## a similar example with a card.set.func object
mu <- card.set.func(-3:4)
mu
Mobius(mu)
```

```
zeta(Mobius(mu))
```

`Mobius.capacity-class` *Class "Mobius.capacity"*

Description

Class representing the Mobius transform of a capacity.

Objects from the Class

Objects can be mainly created by calls to the functions `Mobius.capacity`, `mini.var.capa.ident`, `ls.sorting.capa.ident`, and `least.squares.capa.ident`.

Slots

n: Object of class `numeric` of length 1 containing the number of elements of the set on which the Mobius transform is defined.

k: Object of class `numeric` of length 1 containg the order of truncation of the Mobius transform: the value of subsets whose cardinal is superior to k is put to zero.

subsets: Object of class `numeric` containing the "`k` power set" of the underlying set in "natural" order . The subsets are encoded as integers.

data: Object of class `numeric` of length `choose(n,0) + ... + choose(n,k)` representing the coefficients of a truncated Mobius transform of a capacity in "natural" order.

Extends

Class `Mobius.game`, directly. Class `superclass.capacity`, directly.
Class `Mobius.set.func`, by class `Mobius.game`. Class `superclass.set.func`, by class `Mobius.game`.

Methods

entropy `signature(object = "Mobius.capacity")`

favor `signature(object = "Mobius.capacity")`

is.normalized `signature(object = "Mobius.capacity")`

normalize `signature(object = "Mobius.capacity")`

orness `signature(object = "Mobius.capacity")`

variance `signature(object = "Mobius.capacity")`

veto `signature(object = "Mobius.capacity")`

zeta `signature(object = "Mobius.capacity")`

See Also

capacity-class,
entropy-methods,
favor-methods,
is.normalized-methods,
orness-methods,
variance-methods,
veto-methods,
zeta-methods,
mini.var.capa.ident,
least.squares.capa.ident,
ls.sorting.capa.ident.

Examples

```
## a capacity
mu <- capacity(c(0,0,0:13))
## and its Möbius representation
a <- Mobius(mu)
a

# the attributes of object a
a@n
a@k
a@data
a@subsets

## a test
is.normalized(a)
## normalize it
normalize(a)

## a transformation
zeta(a)
## Let us check ...
Mobius(zeta(a))

## some summary indices
orness(a)
veto(a)
favor(a)
variance(a)
entropy(a)
## the same
summary(a)
```

`Mobius.card.set.func`

Creates an object representing the Möbius transform of a cardinal set function.

Description

Creates objects of class `Mobius.card.set.func`: from an object of class `numeric`.

Usage

```
Mobius.card.set.func(object)
```

Arguments

`object`	An object of class `numeric` containing the coefficients of the cardinal Mobius representation. The coefficient at position `k` corresponds to the value of the cardinal Mobius representation for subsets of size `k`.

Value

Returns an object of class `Mobius.card.set.func`.

See Also

`Mobius.card.set.func-class`.

Examples

```
Mobius.card.set.func(4:-2)
```

`Mobius.card.set.func-class`

Class "Mobius.card.set.func"

Description

Class representing the Mobius transform of a cardinal set function.

Objects from the Class

Objects can be created by calls to the function `Mobius.card.set.func`.

Slots

`n`: Object of class `numeric` of length 1 equal to the number of elements of the set on which the Mobius representation of a cardinal set function is defined.

`data`: Object of class `numeric` of length `n+1` containing the coefficients of the Mobius representation of a cardinal set function.

Extends

Class `superclass.set.func`, directly.

Methods

as.Mobius.set.func `signature(object = "Mobius.card.set.func")`

as.set.func `signature(object = "Mobius.card.set.func")`

as.card.set.func `signature(object = "Mobius.card.set.func")`

to.data.frame `signature(object = "Mobius.card.set.func")`

zeta `signature(object = "Mobius.card.set.func")`

See Also

`Mobius.set.func-class`,
`Mobius.card.set.func`,
`as.Mobius.set.func-methods`,
`as.card.set.func-methods`,
`as.set.func-methods`,
`zeta-methods`,
`to.data.frame-methods`.

Examples

```
## the Möbius representation of a cardinal set function
a <- Mobius.card.set.func(-3:2)

# the attributes of the object
a@n
a@data

## some transformations
as.set.func(a)
zeta(a)
## let us check ...
Mobius(zeta(a))
```

`Mobius.game-class` *Class "Mobius.game"*

Description

Class representing the Mobius transform of a game.

Objects from the Class

Objects can be created by calls to the function `Mobius.game`.

Slots

n: Object of class `numeric` of length 1 containing the number of elements of the set on which the Mobius transform is defined.

k: Object of class `numeric` of length 1 containg the order of truncation of the Mobius transform: subsets whose cardinal is superior to k are considered to be zero.

subsets: Object of class `numeric` containing the "`k` power set" of the underlying set in "natural" order . The subsets are encoded as integers.

data: Object of class `numeric` of length `choose(n,0) + ... + choose(n,k)` representing the coefficients of a truncated Mobius transform of a game in "natural" order.

Extends

Class `Mobius.set.func`, directly. Class `superclass.set.func`, by class `Mobius.set.func`.

Methods

Choquet.integral `signature(object = "Mobius.game", f = "numeric")`

Sipos.integral `signature(object = "Mobius.game", f = "numeric")`

Sugeno.integral `signature(object = "Mobius.game", f = "numeric")`

zeta `signature(object = "Mobius.game")`

See Also

`game-class`,
`Mobius.game`,
`Choquet.integral-methods`,
`Sipos.integral-methods`,
`Sugeno.integral-methods`,
`zeta-methods`,
`expect.Choquet.norm-methods`.

Examples

```
## a game (which is a capacity)
mu <- game(c(0,rep(1,15)))
## and its Möbius representation
a <- Mobius(mu)

# the attributes of object a
a@n
a@k
a@data
a@subsets

## a transformation
zeta(a)
## let us check ...
Mobius(zeta(a))
```

```
## integral calculations
f <- c(0.2,0.3,0.1,0.7)
Choquet.integral(a,f)
Sugeno.integral(a,f)
f <- c(0.2,-0.3,0.1,-0.7)
Sipos.integral(a,f)
```

`Mobius.set.func` *Create objects representing the Möbius transform of a set function.*

Description

These functions create objects of class `Mobius.set.func`, `Mobius.game`, or `Mobius.capacity` from an object of class `numeric`.

Usage

```
Mobius.set.func(object, n, k)
Mobius.game(object, n, k)
Mobius.capacity(object, n, k)
additive.capacity(v)
```

Arguments

`n`	An object of class `numeric` of length 1 equal to the cardinal of the set on which the Mobius representation is defined.
`k`	An object of class `numeric` of length 1 equal to the order of truncation of the Mobius representation. Coefficients whose cardinal is superior to `k` are considered to be zero.
`object`	Object of class `numeric` of length `choose(n,0) + ... + choose(n,k)` representing the coefficients of the truncated Mobius transform in "natural" order.
`v`	An object of class `numeric` containing the coefficients of the capacity on singletons.

Value

Return objects of class `Mobius.set.func`, `Mobius.game`, or `Mobius.capacity`.

See Also

`Mobius.capacity-class`,
`Mobius.game-class`,
`Mobius.set.func-class`,
`k.truncate.Mobius-methods`.

Examples

```
Mobius.set.func(c(0,1,1,2,1,2,1,2,2,3,2),4,2)
Mobius.game(c(0,1,1,2,1,2,1,2,2,3,2),4,2)
Mobius.capacity(c(0,1,1,2,1,2,1,2,2,3,2),4,2)
additive.capacity(c(1,2,3,4))
```

Mobius.set.func-class

Class "Mobius.set.func"

Description

Class representing the Mobius transform of a set function.

Objects from the Class

Objects can be created by calls to the function `Mobius.set.func`.

Slots

n: Object of class `numeric` of length 1 containing the number of elements of the set on which the Mobius transform is defined.

k: Object of class `numeric` of length 1 containg the order of truncation of the Mobius transform: subsets whose cardinal is superior to k are considered to be zero.

subsets: Object of class `numeric` containing the "`k` power set" of the underlying set in "natural" order . The subsets are encoded as integers.

data: Object of class `numeric` of length `choose(n,0) + ... + choose(n,k)` representing the coefficients of a truncated Mobius transform of a set function in "natural" order.

Extends

Class `superclass.set.func`, directly.

Methods

show `signature(object = "Mobius.set.func")`

as.Mobius.card.set.func `signature(object = "Mobius.set.func")`

as.card.set.func `signature(object = "Mobius.set.func")`

as.set.func `signature(object = "Mobius.set.func")`

as.Mobius.game `signature(object = "Mobius.set.func")`

as.Mobius.capacity `signature(object = "Mobius.set.func")`

interaction.indices `signature(object = "Mobius.set.func")`

is.cardinal `signature(object = "Mobius.set.func")`

is.kadditive `signature(object = "Mobius.set.func", k = "numeric")`

is.monotone `signature(object = "Mobius.set.func")`

k.truncate.Mobius `signature(object = "Mobius.set.func", k = "numeric")`

Shapley.value `signature(object = "Mobius.set.func")`

to.data.frame `signature(object = "Mobius.set.func")`

zeta `signature(object = "Mobius.set.func")`

See Also

set.func-class,
Mobius.set.func,
as.Mobius.card.set.func-methods,
as.card.set.func-methods,
as.set.func-methods,
as.Mobius.game-methods,
as.Mobius.capacity-methods,
interaction.indices-methods,
is.cardinal-methods,
is.kadditive-methods,
is.monotone-methods,
k.truncate.Mobius-methods,
Shapley.value-methods,
to.data.frame-methods,
zeta-methods.

Examples

```
## the Möbius transform of a set function directly
a <- Mobius.set.func(1:16,4,4)

## the attributes of the object
a@n
a@k
a@data
a@subsets

## a set function
mu <- set.func(7:-8)
## and its Möbius transform
a <- Mobius(mu)

## some conversions that cannot work
## as.game(a)
## as.capacity(a)
## as.card.set.func(a)

## some tests
is.cardinal(a)
is.kadditive(a,2)
is.monotone(a)
```

```
## some transformations
zeta(a)
k.truncate.Mobius(a,2)

## summary
Shapley.value(a)
interaction.indices(a)
# the same
summary(a)

## save the Möbius transform to a file
d <- to.data.frame(a)
write.table(d,"my.Mobius.set.func.csv",sep="\t")

# finally, some conversions that should work
mu <- set.func(c(0,1,1,1,2,2,2,3))
a <- Mobius(mu)
as.Mobius.game(a)
as.Mobius.capacity(a)
as.Mobius.card.set.func(a)
```

`Shapley.value-methods`
The Shapley value

Description

Computes the Shapley value (`n` indices) of a set function. The set function can be given either under the form of an object of class `set.func`, `card.set.func` or `Mobius.set.func`.

Methods

object = "Mobius.set.func" The Shapley value is computed from the Mobius transform of a set function.

object = "card.set.func" The Shapley value is computed from a cardinal set function.

object = "set.func" The Shapley value is computed from a general set function.

References

L.S. Shapley (1953), *A value for n-person games*, Contributions to the theory of games, vol. 2, Annals of Mathematics Studies, no. 28, pages 307-317, Princeton University Press, Princeton, N. J. .

See Also

`Mobius.set.func-class`,
`card.set.func-class`,
`set.func-class`,
`Mobius-methods`.

Examples

```
## a set function
mu <- set.func(c(0:13/13,1,1))

## the Shapley value
Shapley.value(mu)

## the efficiency property should be satisfied
sum(Shapley.value(mu))

## a similar example using a Mobius.set.func object
a <- Mobius(mu)
Shapley.value(a)

## a similar example using a card.set.func object
mu <- upper.capacity(6)
Shapley.value(mu)
## the efficiency property should be satisfied
Shapley.value(mu)*6
```

Sipos.integral-methods
Sipos integral

Description

Computes the Sipos integral (also called *symmetric Choquet integral*) of a real-valued function with respect to a game. The game can be given either under the form of an object of class `game`, `card.game` or `Mobius.game`.

Methods

object = "game", f = "numeric" The Sipos or symmetric Choquet integral of `f` is computed from a game.

object = "Mobius.game", f = "numeric" The Sipos or symmetric Choquet integral of `f` is computed from the Mobius transform of a game.

object = "card.game", f = "numeric" The Sipos or symmetric Choquet integral of `f` is computed from a cardinal game.

References

M. Grabisch and Ch. Labreuche (2002), The symmetric and asymmetric Choquet integrals on finite spaces for decision making, Statistical Papers 43, pages 37-52.

See Also

`game-class`,
`Mobius.game-class`,
`card.game-class`.

Examples

```
## a normalized capacity
mu <- capacity(c(0:13/13,1,1))

## and its Möbius transform
a <- Mobius(mu)

## a discrete function f
f <- c(0.1,-0.9,-0.3,0.8)

## the Sugeno integral of f w.r.t mu
Sipos.integral(mu,f)
Sipos.integral(a,f)

## a similar example with a cardinal capacity
mu <- uniform.capacity(4)
Sipos.integral(mu,f)
```

`Sugeno.integral-methods`
Sugeno integral

Description

Computes the Sugeno integral of a **non negative** function with respect to a game. Moreover, if the game is a capacity, the range of the function must be contained into the range of the capacity. The game can be given either under the form of an object of class `game`, `card.game` or `Mobius.game`.

Methods

object = "Mobius.game", f = "numeric" The Sugeno integral of `f` is computed from the Mobius transform of a game.

object = "game", f = "numeric" The Sugeno integral of `f` is computed from a game.

object = "card.game", f = "numeric" The Sugeno integral of `f` is computed from a cardinal game.

References

M. Sugeno (1974), *Theory of fuzzy integrals and its applications*, Tokyo Institute of Technology, Tokyo, Japan.

J-L. Marichal (2000), *On Sugeno integral as an aggregation function*, Fuzzy Sets and Systems 114, pages 347-365.

J-L. Marichal (2001), *An axiomatic approach of the discrete Sugeno integral as a tool to aggregate interacting criteria in a qualitative framework*, IEEE Transactions on Fuzzy Systems 9:1, pages 164-172.

T. Murofushi and M. Sugeno (2000), *Fuzzy measures and fuzzy integrals*, in: M. Grabisch, T. Murofushi, and M. Sugeno Eds, Fuzzy Measures and Integrals: Theory and Applications, Physica-Verlag, pages 3-41.

See Also

`game-class`,
`Mobius.game-class`,
`card.game-class`.

Examples

```
## a normalized capacity
mu <- capacity(c(0:13/13,1,1))

## and its Möbius transform
a <- Mobius(mu)

## a discrete function f
f <- c(0.1,0.9,0.3,0.8)

## the Sugeno integral of f w.r.t mu
Sugeno.integral(mu,f)
Sugeno.integral(a,f)

## a similar example with a cardinal capacity
mu <- uniform.capacity(4)
Sugeno.integral(mu,f)
```

---

as.Mobius.capacity-methods

Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "Mobius.set.func" Returns an object of class `Mobius.capacity`.

`as.Mobius.card.set.func-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "Mobius.set.func" Returns an object of class `Mobius.card.set.func`.

object = "set.func" Returns an object of class `Mobius.card.set.func`.

object = "card.set.func" Returns an object of class `Mobius.card.set.func`.

`as.Mobius.game-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "Mobius.set.func" Returns an object of class `Mobius.game`.

`as.Mobius.set.func-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "Mobius.card.set.func" Returns an object of class `Mobius.set.func`.

object = "set.func" Returns an object of class `Mobius.set.func`.

object = "card.set.func" Returns an object of class `Mobius.set.func`.

`as.capacity-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "set.func" Returns an object of class `capacity`.

object = "card.capacity" Returns an object of class `capacity`.

`as.card.capacity-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "card.set.func" Returns an object of class `card.capacity`.

object = "capacity" Returns an object of class `card.capacity`.

`as.card.game-methods`
Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "card.set.func" Returns an object of class `card.game`.

object = "game" Returns an object of class `card.game`.

as.card.set.func-methods

Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "set.func" Returns an object of class `card.set.func`.

object = "Mobius.set.func" Returns an object of class `card.set.func`.

object = "Mobius.card.set.func" Returns an object of class `card.set.func`.

as.game-methods *Coercion method*

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "set.func" Returns an object of class `game`.

object = "card.game" Returns an object of class `game`.

as.set.func-methods

Coercion method

Description

Creates **if possible** a new object of the desired class from the old object.

Methods

object = "card.set.func" Returns an object of class `set.func`.

object = "Mobius.set.func" Returns an object of class `set.func`.

object = "Mobius.card.set.func" Returns an object of class `set.func`.

`capacity-class` *Class "capacity"*

Description

Class representing a capacity, i.e. a monotone set function vanishing at the empty set (also called *fuzzy measure*, *non-additive measure*, *monotone measure*).

Objects from the Class

Objects can be mainly created by calls to the functions `capacity` and `entropy.capa.ident`.

Slots

`n`: Object of class `numeric` of length 1 equal to the number of elements of the set on which the capacity is defined.

`subsets`: Object of class `numeric` of length `2^n` containing the power set of the underlying set in "natural" order. The subsets are coded as integers.

`data`: Object of class `numeric` of length `2^n` containing the coefficients of the capacity in binary order. We necessarily have `data[1] = 0` plus some additional monotonicity constraints.

Extends

Class `game`, directly. Class `superclass.capacity`, directly. Class `set.func`, by class `game`. Class `superclass.set.func`, by class `game`.

Methods

as.card.capacity `signature(object = "capacity")`

conjugate `signature(object = "capacity")`

entropy `signature(object = "capacity")`

favor `signature(object = "capacity")`

is.normalized `signature(object = "capacity")`

normalize `signature(object = "capacity")`

Mobius `signature(object = "capacity")`

orness `signature(object = "capacity")`

variance `signature(object = "capacity")`

veto `signature(object = "capacity")`

See Also

capacity,
entropy.capa.ident,
as.card.capacity-methods,
conjugate-methods,
entropy-methods,
favor-methods,
is.normalized-methods,
normalize-methods,
Mobius-methods,
orness-methods,
variance-methods,
veto-methods.

Examples

```
## a capacity
mu <- capacity(c(0:13,13,13)/13)

## the attributes of the object
mu@n
mu@data
mu@subsets

## a test
is.normalized(mu)
normalize(mu)

## a conversion that should not work
## as.card.capacity(mu)

## some transformations
conjugate(mu)
Mobius(mu)
## let us check ...
zeta(Mobius(mu))

## some summary indices
orness(mu)
veto(mu)
favor(mu)
variance(mu)
entropy(mu)
## the same
summary(mu)
```

---

card.capacity-class

Class "card.capacity"

Description

Class representing a cardinal capacity, i.e. a capacity whose values depend only on the cardinality of subsets (also called *symmetric capacity*).

Objects from the Class

Objects can be created by calls to the functions `card.capacity`, `lower.capacity`, `upper.capacity`, `uniform.capacity`.

Slots

`n`: Object of class `numeric` of length 1 equal to the number of elements of the set on which the cardinal game is defined.

`data`: Object of class `numeric` of length `n+1` containing the coefficients of the cardinal capacity. We necessarily have `data[1]=0` and `data[i+1]-data[i]>0`.

Extends

Class `card.game`, directly. Class `superclass.capacity`, directly. Class `card.set.func`, by class `card.game`. Class `superclass.set.func`, by class `card.game`.

Methods

as.capacity `signature(object = "card.capacity")`

conjugate `signature(object = "card.capacity")`

entropy `signature(object = "card.capacity")`

favor `signature(object = "card.capacity")`

is.normalized `signature(object = "card.capacity")`

normalize `signature(object = "card.capacity")`

orness `signature(object = "card.capacity")`

variance `signature(object = "card.capacity")`

veto `signature(object = "card.capacity")`

See Also

`capacity-class`,
`card.capacity`,
`as.capacity-methods`,
`conjugate-methods`,
`entropy-methods`,
`favor-methods`,
`is.normalized-methods`,
`orness-methods`,
`variance-methods`,
`veto-methods`.

Examples

```
## a capacity
mu <- card.capacity(0:6/6)
## the same
mu <- uniform.capacity(6)

# the attributes of the object
mu@n
mu@data

## a test
is.normalized(mu)
normalize(mu)

## a transformation
conjugate(mu)

## some summary indices
orness(mu)
veto(mu)
favor(mu)
variance(mu)
entropy(mu)
## the same
summary(mu)
```

`card.game-class` *Class "card.game"*

Description

Class representing a cardinal game, i.e. a game whose values depend only on the cardinality of subsets.

Objects from the Class

Objects can be created by calls to the function `card.game`.

Slots

`n`: Object of class `numeric` of length 1 equal to the number of elements of the set on which the cardinal game is defined.

`data`: Object of class `numeric` of length `n+1` containing the coefficients of the cardinal game. We necessarily have `data[1]=0`.

Extends

Class `card.set.func`, directly. Class `superclass.set.func`, by class `card.set.func`.

Methods

as.game `signature(object = "card.game")`

Choquet.integral `signature(object = "card.game", f = "numeric")`

Sipos.integral `signature(object = "card.game", f = "numeric")`

Sugeno.integral `signature(object = "card.game", f = "numeric")`

See Also

`game-class`,
`card.game`,
`as.game-methods`,
`Choquet.integral-methods`,
`Sipos.integral-methods`,
`Sugeno.integral-methods`,

Examples

```
## a cardinal game (which is a capacity)
mu <- card.game(c(0,rep(1,4)))

# the attributes of the object
mu@n
mu@data

## a conversion
as.game(mu)

## integral calculations
f <- c(0.2,0.3,0.1,0.7)
Choquet.integral(mu,f)
Sugeno.integral(mu,f)
f <- c(0.2,-0.3,0.1,-0.7)
Sipos.integral(mu,f)
```

`card.set.func-class` *Class "card.set.func"*

Description

Class representing a cardinal set function, i.e. whose values depend only on the cardinality of subsets.

Objects from the Class

Objects can be created by calls to the function `card.set.func`.

Slots

n: Object of class `numeric` of length 1 equal to the number of elements of the set on which the cardinal set function is defined.

data: Object of class `numeric` of length `n+1` containing the coefficients of the cardinal set function.

Extends

Class `superclass.set.func`, directly.

Methods

as.card.game `signature(object = "card.set.func")`

as.card.capacity `signature(object = "card.set.func")`

as.set.func `signature(object = "card.set.func")`

as.Mobius.card.set.func `signature(object = "card.set.func")`

as.Mobius.set.func `signature(object = "card.set.func")`

conjugate `signature(object = "card.set.func")`

interaction.indices `signature(object = "card.set.func")`

is.cardinal `signature(object = "card.set.func")`

is.kadditive `signature(object = "card.set.func", k = "numeric")`

is.monotone `signature(object = "card.set.func")`

Mobius `signature(object = "card.set.func")`

Shapley.value `signature(object = "card.set.func")`

to.data.frame `signature(object = "card.set.func")`

See Also

`set.func-class`,
`card.set.func`,
`as.card.game-methods`,
`as.card.capacity-methods`,
`as.set.func-methods`,
`as.Mobius.set.func-methods`,
`as.Mobius.card.set.func-methods`,
`conjugate-methods`,
`interaction.indices-methods`,
`is.cardinal-methods`,
`is.kadditive-methods`,
`is.monotone-methods`,
`Mobius-methods`,
`Shapley.value-methods`,
`to.data.frame-methods`.

Examples

```
## a cardinal set function
mu <- card.set.func(-3:2)

# the attributes of the object
mu@n
mu@data

## some conversions that cannot work
## Not run: as.card.game(mu)
## Not run: as.card.capacityfunc(mu)

## the following should work
as.set.func(mu)

## some tests
is.cardinal(mu)
is.kadditive(mu,2)
is.monotone(mu)

## some transformations
conjugate(mu)
Mobius(mu)
## let us check ...
zeta(Mobius(mu))

## summary
Shapley.value(mu)
interaction.indices(mu)
# the same
summary(mu)

## save the set function to a file
d <- to.data.frame(mu)
write.table(d,"my.card.set.func.csv",sep="\t")

# finally, some other conversions that should work
mu <- card.set.func(0:5)
as.card.game(mu)
as.card.capacity(mu)
```

`card.set.func` *Create objects representing cardinal set functions.*

Description

These functions create objects of class `card.set.func`, `card.game`, or `card.capacity` from an object of class `numeric`.

Usage

```
card.set.func(object)
card.game(object)
card.capacity(object)
lower.capacity(n)
upper.capacity(n)
uniform.capacity(n)
```

Arguments

object	An object of class numeric containing the coefficients of the cardinal set function. The coefficient at position k corresponds to the value of the cardinal set function for subsets of size k.
n	An object of class numeric of length 1 containing the cardinal of the set on which the cardinal set function is defined.

Value

Return objects of class card.set.func, card.game, or card.capacity.

See Also

card.capacity-class,
card.game-class,
card.set.func-class.

Examples

```
card.set.func(4:-2)
card.game(c(0,-2:2))
card.capacity(0:5)
lower.capacity(3)
upper.capacity(4)
uniform.capacity(5)
```

---

conjugate-methods *The conjugate (or dual) transform*

Description

Computes the conjugate (also called the *dual*) of a set function. The conjugate of the conjugate of a game gives the original game back.

Methods

object = "capacity" Returns an object of class `capacity`.

object = "card.capacity" Returns an object of class `card.capacity`.

object = "card.set.func" Returns an object of class `card.game`.

object = "set.func" Returns an object of class `game`.

References

T. Murofushi and M. Sugeno (2000), *Fuzzy measures and fuzzy integrals*, in: M. Grabisch, T. Murofushi, and M. Sugeno Eds, Fuzzy Measures and Integrals: Theory and Applications, Physica-Verlag, pages 3-41.

See Also

`capacity-class`,
`card.capacity-class`,
`set.func-class`,
`card.set.func-class`.

Examples

```
## a game
mu <- game(c(0,-7:7))
mu

## its conjugate
conjugate(mu)
## and mu again
conjugate(conjugate(mu))

## a similar example with the upper capacity
mu <- capacity(c(0,rep(1,15)))
mu
conjugate(mu)
conjugate(conjugate(mu))

## a similar example with an object of class card.capacity
mu <- upper.capacity(6)
mu
conjugate(mu)
conjugate(conjugate(mu))

## the conjugate of a set function is a game
mu <- set.func(-7:8)
mu
conjugate(mu)
mu <- card.set.func(-2:5)
conjugate(mu)
```

`entropy-methods` *Normalized entropy of a capacity*

Description

Computes the normalized entropy of a capacity. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" The normalized entropy is computed from the Mobius transform of a capacity.

object = "capacity" The normalized entropy is computed directly from a capacity.

object = "card.capacity" The normalized entropy is computed from a cardinal capacity.

References

J-L. Marichal (2002), *Entropy of discrete Choquet capacities*, European Journal of Operational Research, 3:137, 2002, pages 612-624.

I. Kojadinovic, J-L. Marichal and M. Roubens (2005), *An axiomatic approach to the definition of the entropy of a discrete Choquet capacity*, Information Sciences 172, pages 131-153.

See Also

`capacity-class`,
`Mobius.capacity-class`,
`card.capacity-class`,
`Mobius-methods`.

Examples

```
## a capacity
mu <- capacity(c(0,0,0:13))

## its Möbius transform
a <- Mobius(mu)

## their normalized entropy
entropy(mu)
entropy(a)

## similar examples with card.capacity objects
mu <- lower.capacity(4)
entropy(mu)
mu <- uniform.capacity(4)
entropy(mu)
```

`entropy.capa.ident` *Unsupervised identification of a capacity from profiles*

Description

This function estimates a capacity using as argument a set of data under the form: datum=(score on attribute 1, ..., score on attribute n). The approach roughly consists in replacing the subjective notion of *importance* of a subset of attributes by that of *information content* of a subset of attributes, which is estimated from the data by means of a parametric entropy measure. For more details, see the references hereafter.

Usage

```
entropy.capa.ident(d, entropy = "renyi", parameter = 1)
```

Arguments

`d`	An object of class `data.frame` containing the discretized data. Each column of the `data.frame` must be a `factor`. Each line corresponds to a datum.
`entropy`	An object of class `character` containg the name of the parametric entropy measure to be used for the estimation. The allowed values are "renyi" and "havrda.charvat".
`parameter`	An object of class `numeric` containing he value of the parameter of the choosen entropy. The parameter value must be a positive real number. If equal to 1, the Shannon entropy is used.

Value

Returns an object of class `capacity`.

References

I. Kojadinovic (2004), *Estimation of the weights of interacting criteria from the set of profiles by means of information-theoretic functionals*, European Journal of Operational Research 155:3, pages 741-751.

I. Kojadinovic (2005), *Unsupervised aggregation of commensurate correlated attributes by means of the Choquet integral and entropy functionals*, International Journal of Intelligent Systems, in press.

See Also

`capacity-class`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`least.squares.capa.ident`,
`heuristic.ls.capa.ident`,

ls.sorting.capa.ident,
ls.ranking.capa.ident.

Examples

```
## a set of randomly generated data
## for instance, marks on a [0,20] scale
p <- data.frame(matrix(runif(500,0,20),100,5))
names(p) <- c("Stat","Prob","Alg","Cal","Eng")

## discretization
p[p <= 5] <- 1
p[p > 5 & p <= 10] <- 2
p[p > 10 & p <= 15] <- 3
p[p > 15] <- 4

d <- data.frame(factor(p[[1]]),
                factor(p[[2]]),
                factor(p[[3]]),
                factor(p[[4]]),
                factor(p[[5]]))

## associated unsupervised capacity
mu <- entropy.capa.ident(d)
mu
```

---

expect.Choquet.unif-methods

Expectation and standard deviation of the Choquet integral in the uniform and normal cases

---

Description

Methods for computing the expectation and standard deviation of the Choquet integral in the standard uniform and standard normal cases.

Methods

object = "game" Returns the expectation or the standard deviation of the Choquet integral.

References

J-L. Marichal and I. Kojadinovic (2007), *The distribution of linear combinations of lattice polynomials from the uniform distribution*, submitted.

See Also

game-class, Mobius.game-class.

Examples

```
## a capacity
mu <- capacity(c(0,0.1,0.6,rep(0.9,4),1))

## the expectation and the standard deviation
## of the Choquet integral in the uniform case
expect.Choquet.unif(mu)
sd.Choquet.unif(mu)

## the same but empirically
m <- 10000
ch <- numeric(m)
for (i in 1:m) {
     f <- runif(3)
     ch[i] <- Choquet.integral(mu,f)
}
mean(ch)
sd(ch)

## the expectation and the standard deviation
## of the Choquet integral in the normal case
expect.Choquet.norm(mu)
sd.Choquet.norm(mu)
expect.Choquet.norm(Mobius(mu))

## the same but empirically
for (i in 1:m) {
     f <- rnorm(3)
     ch[i] <- Choquet.integral(mu,f)
}
mean(ch)
sd(ch)
```

`favor-methods` *Favor indices*

Description

Computes the favor indices of a Choquet integral from the underlying **normalized** capacity. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" The favor indices are computed from the Mobius transform of a capacity.

object = "capacity" The favor indices are computed directly from a capacity.

object = "card.capacity" The favor indices are computed from a cardinal capacity.

References

J.-L. Marichal (2000), *Behavioral analysis of aggregation in multicriteria decision aid*, in: Preferences and Decisions under Incomplete Knowledge, J. Fodor and B. De Baets and P. Perny Eds, Physica-Verlag, pages 153-178.

J.-L. Marichal (2004), *Tolerant or intolerant character of interacting criteria in aggregation by the Choquet integral*, European Journal of Operational Research 155:3, pages 771-791.

See Also

capacity-class,
Mobius.capacity-class,
card.capacity-class,
Mobius-methods.

Examples

```
## a capacity
mu <- capacity(c(0:13,13,13))

## its Möbius transform
a <- Mobius(mu)

## their favor indices
favor(mu)
favor(a)

## the same with a card.capacity object
mu <- lower.capacity(4)
favor(mu)
```

---

game-class *Class "game"*

Description

Class representing a game, i.e. a set function vanishing at the empty set (also called *non monotonic fuzzy measure*).

Objects from the Class

Objects can be created by calls to the function `game`.

Slots

`n`: Object of class `numeric` of length 1 equal to the number of elements of the set on which the game is defined.

`subsets`: Object of class `numeric` of length `2^n` containing the power set of the underlying set in "natural" order. The subsets are coded as integers.

data: Object of class `numeric` of length `2^n` containing the coefficients of the game in binary order. We necessarily have `data[1] = 0`.

Extends

Class `set.func`, directly. Class `superclass.set.func`, by class `set.func`.

Methods

as.card.game `signature(object = "game")`

Choquet.integral `signature(object = "game", f = "numeric")`

Mobius `signature(object = "game")`

Sipos.integral `signature(object = "game", f = "numeric")`

Sugeno.integral `signature(object = "game", f = "numeric")`

pdf.Choquet.unif `signature(object = "game", f = "numeric")`

cdf.Choquet.unif `signature(object = "game", f = "numeric")`

expect.Choquet.unif `signature(object = "game")`

sd.Choquet.unif `signature(object = "game")`

expect.Choquet.norm `signature(object = "game")`

sd.Choquet.norm `signature(object = "game")`

See Also

`game`,
`as.card.game-methods`,
`Choquet.integral-methods`,
`Mobius-methods`,
`Sipos.integral-methods`,
`Sugeno.integral-methods`,
`pdf.Choquet.unif-methods`,
`cdf.Choquet.unif-methods`,
`expect.Choquet.unif-methods`,
`sd.Choquet.unif-methods`,
`expect.Choquet.norm-methods`,
`sd.Choquet.norm-methods`.

Examples

```
## a game (which is a capacity)
mu <- game(c(0,rep(1,15)))

## the attributes of the object
mu@n
mu@data
mu@subsets

## a conversion
```

```
as.card.game(mu)

## a transformation
Mobius(mu)
## let us check ...
zeta(Mobius(mu))

## integral calculations
f <- c(0.2,0.3,0.1,0.7)
Choquet.integral(mu,f)
Sugeno.integral(mu,f)
f <- c(0.2,-0.3,0.1,-0.7)
Sipos.integral(mu,f)
```

`heuristic.ls.capa.ident` *Heuristic least squares capacity identification*

Description

Creates an object of class `capacity` or `game` by means of the heuristic algorithm proposed by Grabisch (1995). More precisely, given a set of data under the form: datum=(score on criterion 1, ..., score on criterion n, overall score), this function heuristically determines a capacity that should be close to minimizing the sum of squared errors between overall scores as given by the data and the output of the Choquet integral for those data. See reference hereafter for more details.

Usage

```
heuristic.ls.capa.ident(n, mu, C, g, Integral="Choquet", maxiter = 500,
                        alpha = 0.01, epsilon = 1e-6)
```

Arguments

`n`	Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined.
`mu`	Object of class `capacity` containing the initial capacity. It should be initialized to the equidistributed capacity.
`C`	Object of class `matrix` containing the `n`-column criteria matrix. Each line of this matrix corresponds to a vector of the form: (score on criterion 1, ..., score on criterion n).
`g`	Object of class `numeric` containg the global scores associated with the vectors given in the criteria matrix.
`Integral`	Object of class `character` indicating whether the model is based on the asymmetric Choquet integral (`Integral = "Choquet"`) or the symmetric Choquet integral (`Integral = "Sipos"`).
`maxiter`	Maximum number of iterations.

`alpha`	Object of class `numeric` containing the coefficient of the gradient (multiplicative coefficient of the partial derivative). Its value lies in [0,1]. The higher the value, the larger the move of the capacity at each iteration.
`epsilon`	Object of class `numeric` used for defining a stopping criterion. Iterations are stopped if one of the following cases happen: 1) itmax iterations have been performed 2) the error criterion is growing up 3) normax < epsilon, where normax is the maximum absolute normalized model error, i.e., the maximum over all data of the quantity |e|/outmax, where e=desired output - model output, and outmax is the highest value of the absolute value of the model output

Details

The algorithm is explained in details in the reference hereafter.

Value

The function returns a list structured as follows:

`solution`	Object of class `capacity` if the solution is montone or of class `game` otherwise.
`n.iter`	Number of iterations taken by the algorithm.
`residuals`	Differences between the provided global evaluations and those returned by the obtained model.
`mse`	Mean square error between the provided global evaluations and those returned by the obtained model.

References

M. Grabisch (1995), *A new algorithm for identifying fuzzy measures and its application to pattern recognition*, Int. Joint Conf. of the 4th IEEE Int. Conf. on Fuzzy Systems and the 2nd Int. Fuzzy Engineering Symposium, Yokohama, Japan, 145-150.

See Also

`capacity-class`,
`least.squares.capa.ident`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`ls.sorting.capa.ident`,
`ls.ranking.capa.ident`,
`entropy.capa.ident`.

Examples

```
## number of criteria
n <- 4
```

```
## the number of alternatives
n.a <- 1000

## a randomly generated 5-criteria matrix
C <- matrix(rnorm(n*n.a,10,2),n.a,n)

## the corresponding global scores
g <- numeric(n.a)

## generate a random capacity
x <- runif(2^n-1)
for (i in 2:(2^n-1))
    x[i] <- x[i] + x[i-1]
mu <- normalize(capacity(c(0,x)))
for (i in 1:n.a)
  g[i] <- Choquet.integral(mu,C[i,])

## the initial capacity
## here the uniform capacity
mu.in <- as.capacity(uniform.capacity(n))

## the solution
hlsc <- heuristic.ls.capa.ident(n,mu.in,C,g)
mu.sol <- hlsc$solution

## the difference between mu and mu.sol
mu@data - mu.sol@data

hlsc
```

`interaction.indices-methods`

The Shapley interaction indices

Description

Computes the Shapley interaction index for pairs of elements with respect to a set function. The set function can be given either under the form of an object of class `set.func`, `card.set.func` or `Mobius.set.func`.

Methods

object = "Mobius.set.func" The Shapley interaction indices are computed from the Mobius transform of a set function.

object = "card.set.func" The Shapley interaction indices are computed from a cardinal set function.

object = "set.func" The Shapley interaction indices are computed from a general set function.

References

G. Owen (1971/72), *Multilinear extensions of games*, Management Sci. 18, pages 64–79.

T. Murofushi and S. Soneda (1993), *Techniques for reading fuzzy measures (III): interaction index*, 9th Fuzzy System Symposium, pages 693–696, Saporo, Japan.

M. Grabisch, J-L. Marichal and M. Roubens (2000), *Equivalent representations of set functions*, Mathematics of Operations Research 25(2), pages 157–178.

See Also

`Mobius.set.func-class`,
`card.set.func-class`,
`set.func-class`,
`Mobius-methods`.

Examples

```
## a set function
mu <- set.func(c(-7:6,6,6))

## the associated interaction indices
interaction.indices(mu)

## a similar example using a Mobius.set.func object
a <- Mobius(mu)
interaction.indices(a)

## a similar example using a card.set.func object
mu <- upper.capacity(6)
interaction.indices(mu)
```

---

is.cardinal-methods

Test method

Description

Tests whether a set function is cardinal, i.e., if its values depend only on the cardinality of subsets. The set function can be given either under the form of an object of class `set.func`, `card.set.func` or `Mobius.set.func`.

Methods

object = "Mobius.set.func" Returns an object of class `logical`.

object = "card.set.func" Returns an object of class `logical`.

object = "set.func" Returns an object of class `logical`.

Examples

```
is.cardinal(set.func(-7:8))
is.cardinal(uniform.capacity(8))
is.cardinal(Mobius.game(0:10,4,2))
```

---

is.kadditive-methods

Test method

Description

Tests whether a set function is k-additive, i.e., if its Mobius function vanishes for subsets of more than k elements. The set function can be given either under the form of an object of class `set.func`, `card.set.func` or `Mobius.set.func`.

Details

In order to test whether a coefficient is equal to zero, its absolute value is compared with `epsilon` whose default value is `1e-9`.

Methods

object = "Mobius.set.func", k = "numeric", epsilon = "numeric", epsilon = "numeric" Returns an object of class `logical`.

object = "card.set.func", k = "numeric", epsilon = "numeric" Returns an object of class `logical`.

object = "set.func", k = "numeric", epsilon = "numeric" Returns an object of class `logical`.

References

M. Grabisch (1997), *k-order additive discrete fuzzy measures and their representation*, Fuzzy Sets and Systems 92(2), pages 167–189.

M. Grabisch (2000), *The interaction and Mobius representations of fuzzy measures on finites spaces, k-additive measures: a survey*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 70-93.

See Also

`Mobius.set.func-class`,
`card.set.func-class`,
`set.func-class`,
`Mobius-methods`,

`k.truncate.Mobius-methods`.

Examples

```
## a set function
mu <- set.func(c(0,1,1,1,2,2,2,3))
mu
is.kadditive(mu,2)
is.kadditive(mu,1)

## the Möbius representation of a set function, 2-additive by construction
a <- Mobius.set.func(c(0,1,2,1,3,1,2,1,2,3,1),4,2)
is.kadditive(a,2)
is.kadditive(a,1)
```

---

is.monotone-methods

Test method

Description

Tests whether a set function is monotone with respect to set inclusion. The set function can be given either under the form of an object of class `set.func`, `card.set.func` or `Mobius.set.func`.

Details

For objects of class `set.func` or `card.set.func`, the monotonicity constraints are considered to be satisfied (cf. references hereafter) if the following inequalities are satisfied

$$\mu(S \cup i) - \mu(S) \geq -epsilon$$

for all S and all i. For objects of class `Mobius.set.func`, it is required that a similar condition with respect to the Mobius representation be satisfied (cf. references hereafter).

Methods

object = "Mobius.set.func", verbose = "logical", epsilon = "numeric" Returns an object of class `logical`. If `verbose=TRUE`, displays the violated monotonicity constraints, if any.

object = "card.set.func", verbose = "logical", epsilon = "numeric" Returns an object of class `logical`. If `verbose=TRUE`, displays the violated monotonicity constraints, if any.

object = "set.func", verbose = "logical", epsilon = "numeric" Returns an object of class `logical`. If `verbose=TRUE`, displays the violated monotonicity constraints, if any.

References

A. Chateauneuf and J-Y. Jaffray (1989), *Some characterizations of lower probabilities and other monotone capacities through the use of Mobius inversion*, Mathematical Social Sciences 17:3, pages 263–283.

M. Grabisch (2000), *The interaction and Mobius representations of fuzzy measures on finites spaces, k-additive measures: a survey*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 70-93.

See Also

Mobius.set.func-class,
card.set.func-class,
set.func-class.

Examples

```
## a monotone set function
mu <- set.func(c(0,1,1,1,2,2,2,3))
mu
is.monotone(mu)

## the Möbius representation of a monotone set function
a <- Mobius.set.func(c(0,1,2,1,3,1,2,1,2,3,1),4,2)
is.monotone(a)

## non-monotone examples
mu <- set.func(c(0,-7:7))
is.monotone(mu,verbose=TRUE)
a <- Mobius(mu)
is.monotone(a,verbose=TRUE)
```

---

is.normalized-methods

Test method

Description

Tests whether a capacity is normalized, i.e., if its value on the universal set is 1. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" Returns a `logical`.

object = "capacity" Returns a `logical`.

object = "card.capacity" Returns a `logical`.

See Also

capacity-class,
Mobius.capacity-class,
card.capacity-class.

Examples

```
## a capacity
mu <- capacity(0:15)
is.normalized(mu)
normalize(mu)

## its Möbius transform
a <- Mobius(mu)
is.normalized(a)
normalize(a)

## a cardinal capacity
mu <- uniform.capacity(7)
is.normalized(mu)
```

---

k.truncate.Mobius-methods

k-order truncation of the Möbius representation of a set function.

Description

Truncates the Mobius representation of a set function by considering that the values of subsets whose cardinal is superior to `k` are zero. The result is at most `k`-additive.

Methods

object = "set.func", k = "numeric" Returns an object of class `Mobius.set.func`. The Mobius representation of the set function is first computed and then `k`-truncated.

object = "Mobius.set.func", k = "numeric" Returns an object of class `Mobius.set.func`.

References

M. Grabisch (1997), *k-order additive discrete fuzzy measures and their representation*, Fuzzy Sets and Systems 92(2), pages 167-189.

M. Grabisch (2000), *The interaction and Mobius representations of fuzzy measures on finites spaces, k-additive measures: a survey*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 70-93.

See Also

`Mobius.set.func-class`,
`set.func-class`,
`Mobius-methods`.

Examples

```
## a set function
mu <- set.func(c(0,1,1,1,2,2,2,3))
mu

## 2-truncate it
k.truncate.Mobius(mu,2)
## 2-truncate it
k.truncate.Mobius(Mobius(mu),2)
```

---

least.squares.capa.ident

Least squares capacity identification

Description

Creates an object of class `Mobius.capacity` by means of an approach grounded on least squares optimization. More precisely, given a set of data under the form: datum=(score on criterion 1, ..., score on criterion n, overall score), and possibly additional linear constraints expressing preferences, importance of criteria, etc., this function determines, if it exists, a capacity minimizing the sum of squared errors between overall scores as given by the data and the output of the Choquet integral for those data, and compatible with the additional linear constraints. The existence is ensured if no additional constraint is given. The problem is solved using quadratic programming.

Usage

```
least.squares.capa.ident(n, k, C, g, Integral="Choquet",
A.Shapley.preorder = NULL, A.Shapley.interval = NULL,
A.interaction.preorder = NULL, A.interaction.interval = NULL,
A.inter.additive.partition = NULL,  sigf = 7, maxiter = 40,
epsilon = 1e-6)
```

Arguments

`n`	Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined.
`k`	Object of class `numeric` imposing that the solution is at most a k-additive capacity (the Mobius transform of subsets whose cardinal is superior to `k` vanishes).
`C`	Object of class `matrix` containing the `n`-column criteria matrix. Each line of this matrix corresponds to a vector of the form: (score on criterion 1, ..., score on criterion n).
`g`	Object of class `numeric` containg the global scores associated with the vectors given in the criteria matrix.
`Integral`	Object of class `character` indicating whether the model is based on the asymmetric Choquet integral (`Integral = "Choquet"`) or the symmetric Choquet integral (`Integral = "Sipos"`).

A.Shapley.preorder
: Object of class matrix containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion i is greater than the Shapley importance index of criterion j with preference threshold delta.S". A line is structured as follows: the first element encodes i, the second j, and the third element contains the preference threshold delta.S.

A.Shapley.interval
: Object of class matrix containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion i lies in the interval [a,b]". The interval [a,b] has to be included in [0,1]. A line of the matrix is structured as follows: the first element encodes i, the second a, and the third b.

A.interaction.preorder
: Object of class matrix containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair ij of criteria is greater than the Shapley interaction index of the pair kl of criteria with preference threshold delta.I". A line is structured as follows: the first two elements encode ij, the second two kl, and the fifth element contains the preference threshold delta.I.

A.interaction.interval
: Object of class matrix containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair ij of criteria lies in the interval [a,b]". The interval [a,b] has to be included in [-1,1]. A line is structured as follows: the first two elements encode ij, the third element encodes a, and the fourth element encodes b.

A.inter.additive.partition
: Object of class numeric encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from {1,...,n} of length n such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition {{1,3},{2,4},{5}} can be encoded as c(1,2,1,2,3). See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

sigf
: Precision (default: 7 significant figures). Parameter to be passed to the ipop function (quadratic programming) of the **kernlab** package.

maxiter
: Maximum number of iterations. Parameter to be passed to the ipop function (quadratic programming) of the **kernlab** package.

epsilon
: Object of class numeric containing the threshold value for the monotonicity constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than epsilon.

Details

The quadratic program is solved using the ipop function of the **kernlab** package.

Value

The function returns a list structured as follows:

`solution`	Object of class `Mobius.capacity` containing the Mobius transform of the `k`-additive solution, if any.
`dual`	The dual solution of the problem.
`how`	Character string describing the type of convergence.
`residuals`	Differences between the provided global evaluations and those returned by the obtained model.

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

M. Grabisch, H.T. Nguyen and E.A. Walker (1995), *Fundamentals of uncertainty calculi with applications to fuzzy inference*, Kluwer Academic, Dordrecht.

M. Grabisch and M. Roubens (2000), *Application of the Choquet Integral in Multicriteria Decision Making*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 415-434.

P. Miranda and M. Grabisch (1999), *Optimization issues for fuzzy measures*, International Journal of Fuzziness and Knowledge-based Systems 7:6, pages 545-560.

See Also

`Mobius.capacity-class`,
`heuristic.ls.capa.ident`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`ls.sorting.capa.ident`,
`entropy.capa.ident`.

Examples

```
## the number of data
n.d <- 20

## a randomly generated 5-criteria matrix
C <- matrix(rnorm(5*n.d,10,2),n.d,5)

## the corresponding global scores
g <- numeric(n.d)
mu <- capacity(c(0:29,29,29)/29)
for (i in 1:n.d)
  g[i] <- Choquet.integral(mu,C[i,])
```

```
## Not run:
## the full solution
lsc <- least.squares.capa.ident(5,5,C,g)
a <- lsc$solution
a
mu.sol <- zeta(a)

## the difference between mu and mu.sol
mu@data - mu.sol@data

## the residuals
lsc$residuals

## the mean square error
mean(lsc$residuals^2)

## a 3-additive solution
lsc <- least.squares.capa.ident(5,3,C,g)
a <- lsc$solution
mu.sol <- zeta(a)
mu@data - mu.sol@data
lsc$residuals
## End(Not run)


## a similar example based on the Sipos integral

## a randomly generated 5-criteria matrix
C <- matrix(rnorm(5*n.d,0,2),n.d,5)

## the corresponding global scores
g <- numeric(n.d)
mu <- capacity(c(0:29,29,29)/29)
for (i in 1:n.d)
  g[i] <- Sipos.integral(mu,C[i,])

## Not run:
## the full solution
lsc <- least.squares.capa.ident(5,5,C,g,Integral = "Sipos")
a <- lsc$solution
mu.sol <- zeta(a)
mu@data - mu.sol@data
lsc$residuals

## a 3-additive solution
lsc <- least.squares.capa.ident(5,3,C,g,Integral = "Sipos")
a <- lsc$solution
mu.sol <- zeta(a)
mu@data - mu.sol@data
lsc$residuals
## End(Not run)
```

```
## additional constraints

## a Shapley preorder constraint matrix
## Sh(1) - Sh(2) >= -delta.S
## Sh(2) - Sh(1) >= -delta.S
## Sh(3) - Sh(4) >= -delta.S
## Sh(4) - Sh(3) >= -delta.S
## i.e. criteria 1,2 and criteria 3,4
## should have the same global importances
delta.S <- 0.01
Asp <- rbind(c(1,2,-delta.S),
             c(2,1,-delta.S),
             c(3,4,-delta.S),
             c(4,3,-delta.S)
            )

## a Shapley interval constraint matrix
## 0.3 <= Sh(1) <= 0.9
Asi <- rbind(c(1,0.3,0.9))

## an interaction preorder constraint matrix
## such that I(12) = I(45)
delta.I <- 0.01
Aip <- rbind(c(1,2,4,5,-delta.I),
             c(4,5,1,2,-delta.I))

## an interaction interval constraint matrix
## i.e. -0.20 <= I(12) <= -0.15
delta.I <- 0.01
Aii <- rbind(c(1,2,-0.2,-0.15))

## an inter-additive partition constraint
## criteria 1,2,3 and criteria 4,5 are independent
Aiap <- c(1,1,1,2,2)



## a more constrained solution

lsc <- least.squares.capa.ident(5,5,C,g,Integral = "Sipos",
                                A.Shapley.preorder = Asp,
                                A.Shapley.interval = Asi,
                                A.interaction.preorder = Aip,
                                A.interaction.interval = Aii,
                                A.inter.additive.partition = Aiap,
                                sigf = 5)

a <- lsc$solution
mu.sol <- zeta(a)
mu@data - mu.sol@data
lsc$residuals
summary(a)
```

`lin.prog.capa.ident` *Capacity identification based on linear programming*

Description

Creates an object of class `Mobius.capacity` using the linear programming approach proposed by Marichal and Roubens (see reference hereafter). Roughly speaking, this function determines, if it exists, the capacity compatible with a set of linear constraints that "separates" the most the provided alternatives. The problem is solved using the **lpSolve** package.

Usage

```
lin.prog.capa.ident(n, k, A.Choquet.preorder = NULL,
A.Shapley.preorder = NULL, A.Shapley.interval = NULL,
A.interaction.preorder = NULL, A.interaction.interval = NULL,
A.inter.additive.partition = NULL, epsilon = 1e-6)
```

Arguments

`n` Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined.

`k` Object of class `numeric` imposing that the solution is at most a k-additive capacity (the Mobius transform of subsets whose cardinal is superior to `k` vanishes).

`A.Choquet.preorder` Object of class `matrix` containing the constraints relative to the preorder of the alternatives. Each line of the matrix corresponds to one constraint of the type "alternative `a` is preferred to alternative `b` with preference threshold `delta.C`". A line is structured as follows: the first `n` elements encode alternative `a`, the next `n` elements encode alternative `b`, and the last element contains the preference threshold `delta.C`.

`A.Shapley.preorder` Object of class `matrix` containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` is greater than the Shapley importance index of criterion `j` with preference threshold `delta.S`". A line is structured as follows: the first element encodes `i`, the second `j`, and the third element contains the preference threshold `delta.S`.

`A.Shapley.interval` Object of class `matrix` containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[0,1]`. A line of the matrix is structured as follows: the first element encodes `i`, the second `a`, and the third `b`.

`A.interaction.preorder`
: Object of class `matrix` containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria is greater than the Shapley interaction index of the pair `kl` of criteria with preference threshold `delta.I`". A line is structured as follows: the first two elements encode `ij`, the second two `kl`, and the fifth element contains the preference threshold `delta.I`.

`A.interaction.interval`
: Object of class `matrix` containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[-1,1]`. A line is structured as follows: the first two elements encode `ij`, the third element encodes `a`, and the fourth element encodes `b`.

`A.inter.additive.partition`
: Object of class `numeric` encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from `{1,...,n}` of length `n` such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition `{{1,3},{2,4},{5}}` can be encoded as `c(1,2,1,2,3)`. See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

`epsilon`
: Object of class `numeric` containing the thresold value for the monotonicity constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than `epsilon`.

Details

The linear program is solved using the `lp` function of the **lpSolve**.

Value

The function returns a list structured as follows:

`solution`
: Object of class `Mobius.capacity` containing the Mobius transform of the `k`-additive solution, if any.

`value`
: Value of the objective function.

`lp.object`
: Object of class `lp.object` returned by `lpSolve`.

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

J-L. Marichal and M. Roubens (2000), *Determination of weights of interacting criteria from a reference set*, European Journal of Operational Research 124, pages 641-650.

See Also

Mobius.capacity-class,
mini.var.capa.ident,
mini.dist.capa.ident,
least.squares.capa.ident,
heuristic.ls.capa.ident,
ls.sorting.capa.ident,
entropy.capa.ident.

Examples

```
## some alternatives
a <- c(18,11,18,11,11)
b <- c(18,18,11,11,11)
c <- c(11,11,18,18,11)
d <- c(18,11,11,11,18)
e <- c(11,11,18,11,18)

## preference threshold relative
## to the preorder of the alternatives
delta.C <- 1

## corresponding Choquet preorder constraint matrix
Acp <- rbind(c(d,a,delta.C),
             c(a,e,delta.C),
             c(e,b,delta.C),
             c(b,c,delta.C)
            )

## a Shapley preorder constraint matrix
## Sh(1) - Sh(2) >= -delta.S
## Sh(2) - Sh(1) >= -delta.S
## Sh(3) - Sh(4) >= -delta.S
## Sh(4) - Sh(3) >= -delta.S
## i.e. criteria 1,2 and criteria 3,4
## should have the same global importances
delta.S <- 0.01
Asp <- rbind(c(1,2,-delta.S),
             c(2,1,-delta.S),
             c(3,4,-delta.S),
             c(4,3,-delta.S)
            )

## a Shapley interval constraint matrix
## 0.3 <= Sh(1) <= 0.9
Asi <- rbind(c(1,0.3,0.9))

## an interaction preorder constraint matrix
## such that I(12) = I(34)
delta.I <- 0.01
Aip <- rbind(c(1,2,3,4,-delta.I),
```

```
             c(3,4,1,2,-delta.I))

## an interaction interval constraint matrix
## i.e. -0.20 <= I(12) <= -0.15
Aii <- rbind(c(1,2,-0.2,-0.15))

## Not run:
## a LP 2-additive solution
lin.prog <- lin.prog.capa.ident(5,2,A.Choquet.preorder = Acp)
m <- lin.prog$solution
m

## the resulting global evaluations
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))

## the Shapley value
Shapley.value(m)

## a LP 3-additive more constrained solution
lin.prog2 <- lin.prog.capa.ident(5,3,
                                 A.Choquet.preorder = Acp,
                                 A.Shapley.preorder = Asp)
m <- lin.prog2$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
Shapley.value(m)

## a LP 5-additive more constrained solution
lin.prog3 <- lin.prog.capa.ident(5,5,
                                 A.Choquet.preorder = Acp,
                                 A.Shapley.preorder = Asp,
                                 A.Shapley.interval = Asi,
                                 A.interaction.preorder = Aip,
                                 A.interaction.interval = Aii)

m <- lin.prog3$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
summary(m)
## End(Not run)
```

`ls.ranking.capa.ident` *Least squares capacity identification in the framework of a ranking procedure*

Description

Ranking alternatives means ordering them from the best to the worst alternative. The aim of the implemented method is to model a given ranking by means of a Choquet integral. The result of the function is an object of class `Mobius.capacity`. This function is an implementation of the TOMASO method (see Meyer and Roubens (2005)) in the particular ranking framework. The input data are given under the form of a set of alternatives and a partial weak order, each alternative being described according to a set of criteria. These well-known alternatives are called "prototypes". They represent alternatives for which the decision maker has an a priori knowledge and for which he/she is able to build a ranking. If the provided ranking (partial weak order) of the prototypes cannot be described by a Choquet integral, an approximative solution, which minimizes the "gap" between the given ranking and the one derived from the Choquet integral, is proposed. The problem is solved by quadratic programming.

Usage

```
ls.ranking.capa.ident(n, k, C, rk, d, A.Shapley.preorder = NULL,
A.Shapley.interval = NULL, A.interaction.preorder = NULL,
A.interaction.interval = NULL, A.inter.additive.partition = NULL,
sigf = 5, maxiter = 20, epsilon = 1e-6)
```

Arguments

`n`	Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined (in short, the number of criteria).
`k`	Object of class `numeric` imposing that the solution is at most a k-additive capacity (the Moebius transform of subsets whose cardinal is superior to `k` vanishes).
`C`	Object of class `matrix` containing the `n`-column criteria matrix. Each line of this matrix corresponds to a prototype.
`rk`	Object of class `matrix` containing the constraints relative to the preorder of the prototypes. Each line of this 2-column matrix corresponds to one constraint of the type "the alternative `i` is prefered to the alternative `j`". A line is structured as follows: the first element encodes `i`, the second `j`.
`d`	Object of class `numeric` containing the threshold value, i.e. the minimal "distance" between two neighbor alternatives in the given ranking (e.g. the difference in terms of the Choquet integral of the a prototype with rank 3 and a prototype with rank 4 should be at least `d`).

`A.Shapley.preorder`
: Object of class `matrix` containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` is greater than the Shapley importance index of criterion `j` with preference threshold `delta.S`". A line is structured as follows: the first element encodes `i`, the second `j`, and the third element contains the preference threshold `delta.S`.

`A.Shapley.interval`
: Object of class `matrix` containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[0,1]`. A line of the matrix is structured as follows: the first element encodes `i`, the second `a`, and the third `b`.

`A.interaction.preorder`
: Object of class `matrix` containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria is greater than the Shapley interaction index of the pair `kl` of criteria with preference threshold `delta.I`". A line is structured as follows: the first two elements encode `ij`, the second two `kl`, and the fifth element contains the preference threshold `delta.I`.

`A.interaction.interval`
: Object of class `matrix` containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[-1,1]`. A line is structured as follows: the first two elements encode `ij`, the third element encodes `a`, and the fourth element encodes `b`.

`A.inter.additive.partition`
: Object of class `numeric` encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from `{1,...,n}` of length `n` such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition `{{1,3},{2,4},{5}}` can be encoded as `c(1,2,1,2,3)`. See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

`sigf`
: Precision (default: 5 significant figures). Parameter to be passed to the `ipop` function (quadratic programming) of the **kernlab** package.

`maxiter`
: Maximum number of iterations. Parameter to be passed to the `ipop` function (quadratic programming) of the **kernlab** package.

`epsilon`
: Object of class `numeric` containing the threshold value for the monotonicity constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than `epsilon`.

Details

The quadratic program is solved using the `ipop` function of the **kernlab** package.

Value

The function returns a list structured as follows:

`solution`	Object of class `Mobius.capacity` containing the Moebius transform of the `k`-additive solution.
`glob.eval`	The global evaluations satisfying the given ranking.
`how`	Information returned by `ipop` (cf. **kernlab**) on the convergence of the solver.
`rk.C`	The ranks of the prototypes
`Choquet.C`	The Choquet integral of the prototypes

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

P. Meyer, M. Roubens (2005), *Choice, Ranking and Sorting in Fuzzy Multiple Criteria Decision Aid*, in: J. Figueira, S. Greco, and M. Ehrgott, Eds, Multiple Criteria Decision Analysis: State of the Art Surveys, volume 78 of International Series in Operations Research and Management Science, chapter 12, pages 471-506. Springer Science + Business Media, Inc., New York.

See Also

`Mobius.capacity-class`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`least.squares.capa.ident`,
`heuristic.ls.capa.ident`,
`entropy.capa.ident`.

Examples

```
arthur <- c(1, 1, 0.75, 0.25)
lancelot <- c(0.75, 0.75, 0.75, 0.75)
yvain <- c(1, 0.625, 0.5, 1)
perceval <- c(0.25, 0.5, 0.75, 0.75)
erec <- c(0.375, 1, 0.5 , 0.75)

P <- rbind(arthur, lancelot, yvain, perceval, erec)

# lancelot > erec; yvain > erec, erec > perceval, erec > arthur
rk.proto <- rbind(c(2,5), c(3,5), c(5,4), c(5,1))

n<-4
k<-2
d<-0.1

## search for a capacity which satisfies the constraints
```

```
lrc <- ls.ranking.capa.ident(n ,k, P, rk.proto, d)

lrc
```

`ls.sorting.capa.ident` *Least squares capacity identification in the framework of a sorting procedure*

Description

Sorting alternatives means assigning each alternative to a predefined ordered class. The aim of the implemented method is to model a given classification (sorting) of the alternatives by means of a Choquet integral. The result of the function is an object of class `Mobius.capacity`. This function (in combination with `ls.sorting.treatment`) is an implementation of the TOMASO method; see Meyer and Roubens (2005). The input data are given under the form of a set of alternatives and associated classes, each alternative being described according to a set of criteria. These well-known alternatives are called "prototypes". They represent alternatives for which the decision maker has an a priori knowledge and that he/she is able to assign to one of the ordered classes. If the provided classification of the prototypes cannot be described by a Choquet integral, an approximative solution, which minimizes the "gap" between the given classification and the one derived from the Choquet integral, is proposed. The problem is solved by quadratic programming. This function should be used in combination with `ls.sorting.treatment` which allows to evaluate the model which has been built and to assign other alternatives to the ordered classes.

Usage

```
ls.sorting.capa.ident(n, k, C, cl, d, A.Shapley.preorder = NULL,
A.Shapley.interval = NULL, A.interaction.preorder = NULL,
A.interaction.interval = NULL, A.inter.additive.partition = NULL,
sigf = 5, maxiter = 20, epsilon = 1e-6)
```

Arguments

`n`	Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined (in short, the number of criteria).
`k`	Object of class `numeric` imposing that the solution is at most a k-additive capacity (the Mobius transform of subsets whose cardinal is superior to `k` vanishes).
`C`	Object of class `matrix` containing the `n`-column criteria matrix. Each line of this matrix corresponds to a prototype.
`cl`	Object of class `numeric` containing the indexes of the classes the alternatives are belonging to (the greater the class index, the better the prototype is considered by the decision maker). Each class index between min(cl) and max(cl) must be present.

`d` Object of class `numeric` containing the threshold value for the classes, i.e. the minimal "distance" between two neighbor classes (e.g. the difference in terms of the Choquet integral of the worst prototype of class 3 and the best prototype of class 2 should be at least `d`).

`A.Shapley.preorder`
Object of class `matrix` containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` is greater than the Shapley importance index of criterion `j` with preference threshold `delta.S`". A line is structured as follows: the first element encodes `i`, the second `j`, and the third element contains the preference threshold `delta.S`.

`A.Shapley.interval`
Object of class `matrix` containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[0,1]`. A line of the matrix is structured as follows: the first element encodes `i`, the second `a`, and the third `b`.

`A.interaction.preorder`
Object of class `matrix` containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria is greater than the Shapley interaction index of the pair `kl` of criteria with preference threshold `delta.I`". A line is structured as follows: the first two elements encode `ij`, the second two `kl`, and the fifth element contains the preference threshold `delta.I`.

`A.interaction.interval`
Object of class `matrix` containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[-1,1]`. A line is structured as follows: the first two elements encode `ij`, the third element encodes `a`, and the fourth element encodes `b`.

`A.inter.additive.partition`
Object of class `numeric` encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from `{1,...,n}` of length `n` such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition `{{1,3},{2,4},{5}}` can be encoded as `c(1,2,1,2,3)`. See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

`sigf` Precision (default: 5 significant figures). Parameter to be passed to the `ipop` function (quadratic programming) of the **kernlab** package.

`maxiter` Maximum number of iterations. Parameter to be passed to the `ipop` function (quadratic programming) of the **kernlab** package.

`epsilon` Object of class `numeric` containing the threshold value for the monotonicity

constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than `epsilon`.

Details

The quadratic program is solved using the `ipop` function of the **kernlab** package.

Value

The function returns a list structured as follows:

`solution`	Object of class `Mobius.capacity` containing the Mobius transform of the `k`-additive solution.
`glob.eval`	The global evaluations satisfying the given classification.
`how`	Information returned by `ipop` (cf. **kernlab**) on the convergence of the solver.

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

P. Meyer, M. Roubens (2005), *Choice, Ranking and Sorting in Fuzzy Multiple Criteria Decision Aid*, in: J. Figueira, S. Greco, and M. Ehrgott, Eds, Multiple Criteria Decision Analysis: State of the Art Surveys, volume 78 of International Series in Operations Research and Management Science, chapter 12, pages 471-506. Springer Science + Business Media, Inc., New York.

See Also

`Mobius.capacity-class`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`least.squares.capa.ident`,
`heuristic.ls.capa.ident`,
`ls.sorting.treatment`,
`entropy.capa.ident`.

Examples

```
## generate a random problem with 10 prototypes and 4 criteria
n.proto <- 10 ## prototypes
n <- 4  ## criteria
k <- 4
d <- 0.1

## generating random data for the prototypes
C <- matrix(runif(n.proto*n,0,1),n.proto,n)
cl <- numeric(n.proto)
```

```
## the corresponding global evaluations
glob.eval <- numeric(n.proto)
a <- capacity(c(0:(2^n-3),(2^n-3),(2^n-3))/(2^n-3))
for (i in 1:n.proto)
  glob.eval[i] <- Choquet.integral(a,C[i,])

## and the classes for the prototypes
cl[glob.eval <= 0.33] <- 1
cl[glob.eval > 0.33 & glob.eval <= 0.66] <-2
cl[glob.eval > 0.66] <- 3

cl

## Not run:
# starting the calculations
# search for a capacity which satisfies the constraints
lsc <- ls.sorting.capa.ident(n ,k, C, cl, d)

## output of the quadratic program (ipop, package kernlab)
lsc$how

## the capacity satisfying the constraints
lsc$solution
summary(lsc$solution)
## the global evaluations satisfying the constraints
lsc$glob.eval
## End(Not run)

## let us now add some constraints

## a Shapley preorder constraint matrix
## Sh(1) > Sh(2)
## Sh(3) > Sh(4)
delta.S <-0.01
Asp <- rbind(c(1,2,delta.S), c(3,4,delta.S))

## a Shapley interval constraint matrix
## 0.1 <= Sh(1) <= 0.2
Asi <- rbind(c(1,0.1,0.2))

## an interaction preorder constraint matrix
## such that I(12) > I(34)
delta.I <- 0.01
Aip <- rbind(c(1,2,3,4,delta.I))

## an interaction interval constraint matrix
## i.e. 0.2 <= I(12) <= 0.4
## and 0 < I(34) <= 1
Aii <- rbind(c(1,2,0.2,0.4), c(3,4,delta.I,1))

## an inter-additive partition constraint
## criteria 1,2 and criteria 3,4 are independent
Aiap <- c(1,1,2,2)
```

```
## starting the calculations
## search for a capacity which satisfies the constraints
lsc <- ls.sorting.capa.ident(n ,k, C, cl, d,
                                  A.Shapley.preorder = Asp,
                                  A.Shapley.interval = Asi,
                                  A.interaction.preorder = Aip,
                                  A.interaction.interval = Aii,
                                  A.inter.additive.partition = Aiap)

## output of ipop
lsc$how
## the capacity satisfying the constraints
lsc$solution
summary(lsc$solution)
## the global evaluations satisfying the constraints
lsc$glob.eval
```

`ls.sorting.treatment`

Least squares capacity identification in the framework of a sorting procedure: evaluation of the determined capacity

---

Description

This function assigns alternatives to classes and optionally compares the obtained classification to a given one. The classes are described by a set of prototypes (well-known alternatives for the decision maker) and a capacity (which can, for instance, be determined by `ls.sorting.capa.ident`). This function (in combination with `ls.sorting.capa.ident`) is an implementation of the TOMASO method; see Meyer and Roubens (2005).

Usage

```
ls.sorting.treatment(P, cl.proto, a, A, cl.orig.A = NULL)
```

Arguments

`P`	Object of class `matrix` containing the `n`-column criteria matrix. Each line of this matrix corresponds to a prototype. A prototype is an alternative for which the class is known beforehand.
`cl.proto`	Object of class `numeric` containing the indexes of the classes the prototypes `P` are belonging to (the greater the class index, the better the prototype is considered by the decision maker).
`a`	Object of class `Mobius.capacity` used to model the classes determined by `P` by a Choquet integral.
`A`	Object of class `matrix` containing the `n`-column criteria matrix representing the alternatives which have to be classified.

`cl.orig.A` Object of class `numeric` containing the "true" classes of the alternatives of `A`. This can be used for the evaluation of the quality of the model.

Details

Value

The function returns a list structured as follows:

`correct.A` Object of class `matrix` which contains the different types of assignments of the elements of `A`. In columns the alternatives. First line: correct assignment to a single class (assignment of degree 0). Second line: correct ambiguous assignment to two classes (assignment of degree 1), etc. Last line: bad assignment. In case no `orig.class.A` is given, `correct.A` is `NULL`.

`class.A` Object of class `matrix` which contains the assignments of the elements of `A`. In columns, the alternatives. First line, lower class. Second line, upper class.

`eval.correct` Object of class `numeric` which contains the ratio assignments over number of elements of `A`(for each type of assignment; the first element is the ratio of correct assignments). In case no `orig.class.A` is given, `eval.correct` is `NULL`.

`minmax.P` Object of class `matrix` which contains the min and the max of each class, according to the prototypes. In columns, the classes, first line: the minimum, second line: the maximum.

`Choquet.A` Object of class `numeric` which contains the Choquet integral of the evaluations of the alternatives of `A`.

References

P. Meyer, M. Roubens (2005), *Choice, Ranking and Sorting in Fuzzy Multiple Criteria Decision Aid*, in: J. Figueira, S. Greco, and M. Ehrgott, Eds, Multiple Criteria Decision Analysis: State of the Art Surveys, volume 78 of International Series in Operations Research and Management Science, chapter 12, pages 471-506. Springer Science + Business Media, Inc., New York.

See Also

`Mobius.capacity-class`,
`mini.var.capa.ident`,
`mini.dist.capa.ident`,
`ls.sorting.capa.ident`,
`least.squares.capa.ident`,
`heuristic.ls.capa.ident`,
`entropy.capa.ident`.

Examples

```
## generate a random problem with 10 prototypes and 4 criteria
n.proto <- 10 ## prototypes
n <- 4  ## criteria
P <- matrix(runif(n.proto*n,0,1),n.proto,n)

## the corresponding global scores, based on a randomly generated
## capacity a
glob.eval <- numeric(n.proto)
a <- capacity(c(0:(2^n-3),(2^n-3),(2^n-3))/(2^n-3))
for (i in 1:n.proto)
  glob.eval[i] <- Choquet.integral(a,P[i,])

## based on these global scores, let us create a classification (3 classes)
cl.proto<-numeric(n.proto)
cl.proto[glob.eval <= 0.33] <- 1
cl.proto[glob.eval > 0.33 & glob.eval<=0.66] <-2
cl.proto[glob.eval > 0.66] <- 3

## search for a capacity which satisfies the constraints
lsc <- ls.sorting.capa.ident(n ,4, P, cl.proto, 0.1)

## output of the QP
lsc$how

## analyse the quality of the model (classify the prototypes by the
## model and compare both assignments)
lst <- ls.sorting.treatment(P,cl.proto,lsc$solution,P,cl.proto)

## assignments of the prototypes
lst$class.A
## assignment types
lst$correct.A
## evaluation
lst$eval.correct

## generate a second set of random alternatives (A)
## their "correct" class is determined as beforehand with the
## randomly generated capacity a
## the goal is to see if we can reproduce this classification
## by the capacity learnt from the prototypes

## a randomly generated criteria matrix of 10 alternatives
A <- matrix(runif(10*n,0,1),10,n)
cl.orig.A <-numeric(10)

## the corresponding global scores
glob.eval.A <- numeric(10)
for (i in 1:10)
  glob.eval.A[i] <- Choquet.integral(a,A[i,])
```

```
## based on these global scores, let us determine a classification
cl.orig.A[glob.eval.A <= 0.33] <- 1
cl.orig.A[glob.eval.A>0.33 & glob.eval.A<=0.66] <-2
cl.orig.A[glob.eval.A > 0.66] <- 3

## let us now classify the alternatives of A according to the model
## built on P

lst <- ls.sorting.treatment(P,cl.proto,lsc$solution,A,cl.orig.A)

## assignment of the alternatives of A
lst$class.A
## type of assignments
lst$correct.A
## evaluation
lst$eval.correct

## show the learnt capacity
## x11()
## barplot(Shapley.value(lsc$solution), main="Learnt capacity", sub="Shapley")
## summary of the learnt capacity
lsc$solution
summary(lsc$solution)
```

`mini.dist.capa.ident`

Minimum distance capacity identification

Description

Creates an object of class `Mobius.capacity` using a minimum distance principle. More precisely, this function determines, if it exists, the closest capacity to a user-given game compatible with a set of linear constraints. The distance can be chosen among three quadratic distances (see help and references hereafter). The problem is solved using strictly convex quadratic programming.

Usage

```
mini.dist.capa.ident(a, k, distance = "Choquet.coefficients",
A.Choquet.preorder = NULL, A.Shapley.preorder = NULL,
A.Shapley.interval = NULL, A.interaction.preorder = NULL,
A.interaction.interval = NULL, A.inter.additive.partition = NULL,
epsilon = 1e-6)
```

Arguments

`a`	Object of class `Mobius.game` containing the Mobius transform of the game to be approached.

k — Object of class numeric imposing that the solution is at most a k-additive capacity (the Mobius transform of subsets whose cardinal is superior to k vanishes).

distance — Object of class character indicating which quadratic distance is to be used in the objective function. The choice distance = "Choquet.coefficients" will result in the minimization of the average distance between Choquet integral coefficients, the choice distance = "binary.alternatives" will result in the minimization of the average distance between binary alternatives, and the choice distance = "global.scores" will result in the minimization of the average distance between global scores.

A.Choquet.preorder — Object of class matrix containing the constraints relative to the preorder of the alternatives. Each line of the matrix corresponds to one constraint of the type "alternative a is preferred to alternative b with preference threshold delta.C". A line is structured as follows: the first n elements encode alternative a, the next n elements encode alternative b, and the last element contains the preference threshold delta.C.

A.Shapley.preorder — Object of class matrix containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion i is greater than the Shapley importance index of criterion j with preference threshold delta.S". A line is structured as follows: the first element encodes i, the second j, and the third element contains the preference threshold delta.S.

A.Shapley.interval — Object of class matrix containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion i lies in the interval [a,b]". The interval [a,b] has to be included in [0,1]. A line of the matrix is structured as follows: the first element encodes i, the second a, and the third b.

A.interaction.preorder — Object of class matrix containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair ij of criteria is greater than the Shapley interaction index of the pair kl of criteria with preference threshold delta.I". A line is structured as follows: the first two elements encode ij, the second two kl, and the fifth element contains the preference threshold delta.I.

A.interaction.interval — Object of class matrix containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair ij of criteria lies in the interval [a,b]". The interval [a,b] has to be included in [-1,1]. A line is structured as follows: the first two elements encode ij, the third element encodes a, and the fourth element encodes b.

`A.inter.additive.partition`
: Object of class `numeric` encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from `{1,...,n}` of length `n` such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition `{{1,3},{2,4},{5}}` can be encoded as `c(1,2,1,2,3)`. See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

`epsilon`
: Object of class `numeric` containing the thresold value for the monotonicity constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than `epsilon`.

Details

The quadratic program is solved using the `solve.QP` function of the **quadprog** package.

Value

The function returns a list structured as follows:

`solution`
: Object of class `Mobius.capacity` containing the Mobius transform of the `k`-additive solution, if any.

`value`
: Value of the objective function.

`iterations`
: Information returned by `solve.QP`.

`iact`
: Information returned by `solve.QP`.

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

I. Kojadinovic (2006), *Quadratic objective functions for capacity and bi-capacity identification and approximation*, A Quarterly Journal of Operations Research (40R), in press.

See Also

`Mobius.capacity-class`,
`lin.prog.capa.ident`,
`mini.var.capa.ident`,
`least.squares.capa.ident`,
`heuristic.ls.capa.ident`,
`ls.sorting.capa.ident`,
`entropy.capa.ident`.

Examples

```
## some alternatives
a <- c(18,11,18,11,11)
b <- c(18,18,11,11,11)
c <- c(11,11,18,18,11)
d <- c(18,11,11,11,18)
e <- c(11,11,18,11,18)

## preference threshold relative
## to the preorder of the alternatives
delta.C <- 1

## corresponding Choquet preorder constraint matrix
Acp <- rbind(c(d,a,delta.C),
             c(a,e,delta.C),
             c(e,b,delta.C),
             c(b,c,delta.C)
            )

## a Shapley preorder constraint matrix
## Sh(1) - Sh(2) >= -delta.S
## Sh(2) - Sh(1) >= -delta.S
## Sh(3) - Sh(4) >= -delta.S
## Sh(4) - Sh(3) >= -delta.S
## i.e. criteria 1,2 and criteria 3,4
## should have the same global importances
delta.S <- 0.01
Asp <- rbind(c(1,2,-delta.S),
             c(2,1,-delta.S),
             c(3,4,-delta.S),
             c(4,3,-delta.S)
            )

## a Shapley interval constraint matrix
## 0.3 <= Sh(1) <= 0.9
Asi <- rbind(c(1,0.3,0.9))

## an interaction preorder constraint matrix
## such that I(12) = I(34)
delta.I <- 0.01
Aip <- rbind(c(1,2,3,4,-delta.I),
             c(3,4,1,2,-delta.I))

## an interaction interval constraint matrix
## i.e. -0.20 <= I(12) <= -0.15
Aii <- rbind(c(1,2,-0.2,-0.15))

## the capacity that we want to approach
x <- runif(31)
for (i in 2:31)
    x[i] <- x[i] + x[i-1]
mu <- normalize(capacity(c(0,x)))
```

```
## and its Mobius transform
a.mu <- Mobius(mu)

## some basic checks
## Not run:
mini.dist.capa.ident(a.mu,5)
mini.dist.capa.ident(a.mu,5,"binary.alternatives")
mini.dist.capa.ident(a.mu,5,"global.scores")
mini.dist.capa.ident(a.mu,3)
mini.dist.capa.ident(a.mu,3,"binary.alternatives")
mini.dist.capa.ident(a.mu,3,"global.scores")
## End(Not run)

## a minimum distance 2-additive solution
min.dist <- mini.dist.capa.ident(a.mu,2,"binary.alternatives",A.Choquet.preorder = Acp)
m <- min.dist$solution
m

## a minimum distance 3-additive more constrained solution
min.dist2 <- mini.dist.capa.ident(a.mu,3,"global.scores",
                                  A.Choquet.preorder = Acp,
                                  A.Shapley.preorder = Asp)
m <- min.dist2$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
Shapley.value(m)

## Not run:
## a minimum distance 5-additive more constrained solution
min.dist3 <- mini.dist.capa.ident(a.mu,5,
                                  A.Choquet.preorder = Acp,
                                  A.Shapley.preorder = Asp,
                                  A.Shapley.interval = Asi,
                                  A.interaction.preorder = Aip,
                                  A.interaction.interval = Aii)

m <- min.dist3$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
summary(m)
## End(Not run)
```

`mini.var.capa.ident` *Minimum variance capacity identification*

Description

Creates an object of class `Mobius.capacity` using a maximum like quadratic entropy principle, which is equivalent to the minimization of the variance. More precisely, this function determines, if it exists, the minimum variance capacity compatible with a set of linear constraints. The problem is solved using strictly convex quadratic programming.

Usage

```
mini.var.capa.ident(n, k, A.Choquet.preorder = NULL,
A.Shapley.preorder = NULL, A.Shapley.interval = NULL,
A.interaction.preorder = NULL, A.interaction.interval = NULL,
A.inter.additive.partition = NULL, epsilon = 1e-6)
```

Arguments

`n`	Object of class `numeric` containing the number of elements of the set on which the object of class `Mobius.capacity` is to be defined.
`k`	Object of class `numeric` imposing that the solution is at most a k-additive capacity (the Mobius transform of subsets whose cardinal is superior to `k` vanishes).
`A.Choquet.preorder`	Object of class `matrix` containing the constraints relative to the preorder of the alternatives. Each line of the matrix corresponds to one constraint of the type "alternative `a` is preferred to alternative `b` with preference threshold `delta.C`". A line is structured as follows: the first `n` elements encode alternative `a`, the next `n` elements encode alternative `b`, and the last element contains the preference threshold `delta.C`.
`A.Shapley.preorder`	Object of class `matrix` containing the constraints relative to the preorder of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` is greater than the Shapley importance index of criterion `j` with preference threshold `delta.S`". A line is structured as follows: the first element encodes `i`, the second `j`, and the third element contains the preference threshold `delta.S`.
`A.Shapley.interval`	Object of class `matrix` containing the constraints relative to the quantitative importance of the criteria. Each line of this 3-column matrix corresponds to one constraint of the type "the Shapley importance index of criterion `i` lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[0,1]`. A line of the matrix is structured as follows: the first element encodes `i`, the second `a`, and the third `b`.

`A.interaction.preorder`
: Object of class `matrix` containing the constraints relative to the preorder of the pairs of criteria in terms of the Shapley interaction index. Each line of this 5-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria is greater than the Shapley interaction index of the pair `kl` of criteria with preference threshold `delta.I`". A line is structured as follows: the first two elements encode `ij`, the second two `kl`, and the fifth element contains the preference threshold `delta.I`.

`A.interaction.interval`
: Object of class `matrix` containing the constraints relative to the type and the magnitude of the Shapley interaction index for pairs of criteria. Each line of this 4-column matrix corresponds to one constraint of the type "the Shapley interaction index of the pair `ij` of criteria lies in the interval `[a,b]`". The interval `[a,b]` has to be included in `[-1,1]`. A line is structured as follows: the first two elements encode `ij`, the third element encodes `a`, and the fourth element encodes `b`.

`A.inter.additive.partition`
: Object of class `numeric` encoding a partition of the set of criteria imposing that there be no interactions among criteria belonging to different classes of the partition. The partition is to be given under the form of a vector of integers from `{1,...,n}` of length `n` such that two criteria belonging to the same class are "marked" by the same integer. For instance, the partition `{{1,3},{2,4},{5}}` can be encoded as `c(1,2,1,2,3)`. See Fujimoto and Murofushi (2000) for more details on the concept of mu-inter-additive partition.

`epsilon`
: Object of class `numeric` containing the thresold value for the monotonicity constraints, i.e. the difference between the "weights" of two subsets whose cardinals differ exactly by 1 must be greater than `epsilon`.

Details

The quadratic program is solved using the `solve.QP` function of the **quadprog** package.

Value

The function returns a list structured as follows:

`solution`
: Object of class `Mobius.capacity` containing the Mobius transform of the `k`-additive solution, if any.

`value`
: Value of the objective function.

`iterations`
: Information returned by `solve.QP`.

`iact`
: Information returned by `solve.QP`.

References

K. Fujimoto and T. Murofushi (2000) *Hierarchical decomposition of the Choquet integral*, in: Fuzzy Measures and Integrals: Theory and Applications, M. Grabisch, T. Murofushi, and M. Sugeno Eds, Physica Verlag, pages 95-103.

I. Kojadinovic (2005), *Minimum variance capacity identification*, European Journal of Operational Research, in press.

See Also

Mobius.capacity-class,
lin.prog.capa.ident,
mini.dist.capa.ident,
least.squares.capa.ident,
heuristic.ls.capa.ident,
ls.sorting.capa.ident,
entropy.capa.ident.

Examples

```
## some alternatives
a <- c(18,11,18,11,11)
b <- c(18,18,11,11,11)
c <- c(11,11,18,18,11)
d <- c(18,11,11,11,18)
e <- c(11,11,18,11,18)

## preference threshold relative
## to the preorder of the alternatives
delta.C <- 1

## corresponding Choquet preorder constraint matrix
Acp <- rbind(c(d,a,delta.C),
             c(a,e,delta.C),
             c(e,b,delta.C),
             c(b,c,delta.C)
            )

## a Shapley preorder constraint matrix
## Sh(1) - Sh(2) >= -delta.S
## Sh(2) - Sh(1) >= -delta.S
## Sh(3) - Sh(4) >= -delta.S
## Sh(4) - Sh(3) >= -delta.S
## i.e. criteria 1,2 and criteria 3,4
## should have the same global importances
delta.S <- 0.01
Asp <- rbind(c(1,2,-delta.S),
             c(2,1,-delta.S),
             c(3,4,-delta.S),
             c(4,3,-delta.S)
            )

## a Shapley interval constraint matrix
## 0.3 <= Sh(1) <= 0.9
Asi <- rbind(c(1,0.3,0.9))

## an interaction preorder constraint matrix
## such that I(12) = I(34)
delta.I <- 0.01
Aip <- rbind(c(1,2,3,4,-delta.I),
```

```
              c(3,4,1,2,-delta.I))

## an interaction interval constraint matrix
## i.e. -0.20 <= I(12) <= -0.15
Aii <- rbind(c(1,2,-0.2,-0.15))

## a minimum variance 2-additive solution
min.var <- mini.var.capa.ident(5,2,A.Choquet.preorder = Acp)
m <- min.var$solution
m

## the resulting global evaluations
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))

## the Shapley value
Shapley.value(m)

## a minimum variance 3-additive more constrained solution
min.var2 <- mini.var.capa.ident(5,3,
                                A.Choquet.preorder = Acp,
                                A.Shapley.preorder = Asp)
m <- min.var2$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
Shapley.value(m)

## a minimum variance 5-additive more constrained solution
min.var3 <- mini.var.capa.ident(5,5,
                                A.Choquet.preorder = Acp,
                                A.Shapley.preorder = Asp,
                                A.Shapley.interval = Asi,
                                A.interaction.preorder = Aip,
                                A.interaction.interval = Aii)

m <- min.var3$solution
m
rbind(c(a,mean(a),Choquet.integral(m,a)),
      c(b,mean(b),Choquet.integral(m,b)),
      c(c,mean(c),Choquet.integral(m,c)),
      c(d,mean(d),Choquet.integral(m,d)),
      c(e,mean(e),Choquet.integral(m,e)))
summary(m)
```

normalize-methods *Normalizes a capacity.*

Description

Returns the normalized version of a capacity. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`. In the case of objects of class `capacity` and `card.capacity`, the normalization is performed by dividing each coefficient of the capacity by the value of the capacity on the universe. In the case of objects of class `Mobius.capacity`, the normalization is performed by dividing each coefficient of the Mobius transform by the sum of all the coefficients.

Methods

object = "Mobius.capacity" Returns an object of class `Mobius.capacity`.

object = "capacity" Returns an object of class `capacity`.

object = "card.capacity" Returns an object of class `card.capacity`.

See Also

`capacity-class`,
`Mobius.capacity-class`,
`card.capacity-class`,
`Mobius-methods`,
`is.normalized-methods`.

Examples

```
## a capacity
mu <- capacity(0:15)

## normalize it
is.normalized(mu)
normalize(mu)

## a similar example with a Mobius.capacity object
a <- Mobius(mu)
is.normalized(a)
normalize(a)
zeta(normalize(a))

## a similar example with a card.capacity object
mu <- card.capacity(0:6)
is.normalized(mu)
normalize(mu)
Mobius(normalize(mu))
```

orness-methods *Orness degree*

Description

Computes the orness degree of a Choquet integral from the underlying **normalized** capacity. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" The orness degree is computed from the Mobius transform of a capacity.

object = "capacity" The orness degree is computed directly from a capacity.

object = "card.capacity" The orness degree is computed from a cardinal capacity.

References

J.-L. Marichal (2000), *Behavioral analysis of aggregation in multicriteria decision aid*, in Preferences and Decisions under Incomplete Knowledge, J. Fodor and B. De Baets and P. Perny Eds, Physica-Verlag, pages 153-178, 2000.

See Also

`capacity-class`,
`Mobius.capacity-class`,
`card.capacity-class`,
`Mobius-methods`.

Examples

```
## the upper capacity
mu <- capacity(c(0,rep(1,15)))

## the Choquet integral w.r.t mu behaves like the maximum operator
f <- c(0.1,0.1,0,0.9)
Choquet.integral(mu,f)

## its orness is 1
orness(mu)

## the same example with a Mobius.capacity object
a <- Mobius(mu)
Choquet.integral(a,f)
orness(a)

## the same example with a card.capacity object
mu <- upper.capacity(4)
```

```
Choquet.integral(mu,f)
orness(mu)
```

`pdf.Choquet.unif-methods` *Distribution of the Choquet integral for evaluations uniformly distributed on the unit hypercube*

Description

Methods for computing the probability density and cumulative distribution functions of the Choquet integral with respect to a game for evaluations uniformly distributed on the unit hypercube.

Methods

object = "game", y = "numeric" Returns the value of the p.d.f. or the c.d.f. at `y`.

References

J-L. Marichal and I. Kojadinovic (2007), *The distribution of linear combinations of lattice polynomials from the uniform distribution*, submitted.

See Also

`game-class`.

Examples

```
## a capacity
mu <- capacity(c(0,0.1,0.6,rep(0.9,4),1))
## the cdf of the Choquet integral at 0.7
cdf.Choquet.unif(mu,0.7)

## the same but empirically
m <- 10000
ch <- numeric(m)
for (i in 1:m) {
     f <- runif(3)
     ch[i] <- Choquet.integral(mu,f)
}
sum(ifelse(ch<=0.7,1,0))/m
```

round-methods *Rounding of set function coefficients*

Description

Rounds the coefficients of a set function to the specified number of decimal places (default 0).

Methods

x = "superclass.set.func", digits = "numeric" Returns an object of the same class as `x` whose `data` attribute is rounded.

Examples

```
## a capacity
mu <- capacity(0:15/15)
mu
round(mu,2)

## a similar example with a Mobius.capacity object
a <- Mobius(mu)
a
round(a,1)

## a similar example with a card.capacity object
mu <- uniform.capacity(6)
mu
round(mu)
```

set.func-class *Class "set.func"*

Description

Class representing a set function.

Objects from the Class

Objects can be created by calls to the function `set.func`.

Slots

n: Object of class `numeric` of length 1 equal to the number of elements of the set on which the set function is defined.

subsets: Object of class `numeric` of length `2^n` containing the power set of the underlying set in "natural" order. The subsets are coded as integers.

data: Object of class `numeric` of length `2^n` containing the coefficients of the set function in binary order.

Extends

Class `superclass.set.func`, directly.

Methods

show `signature(object = "set.func")`
as.game `signature(object = "set.func")`
as.capacity `signature(object = "set.func")`
as.card.set.func `signature(object = "set.func")`
as.Mobius.set.func `signature(object = "set.func")`
as.Mobius.card.set.func `signature(object = "set.func")`
conjugate `signature(object = "set.func")`
interaction.indices `signature(object = "set.func")`
is.cardinal `signature(object = "set.func")`
is.kadditive `signature(object = "set.func", k = "numeric")`
is.monotone `signature(object = "set.func")`
k.truncate.Mobius `signature(object = "set.func", k = "numeric")`
Mobius `signature(object = "set.func")`
Shapley.value `signature(object = "set.func")`
to.data.frame `signature(object = "set.func")`

See Also

`set.func`,
`as.game-methods`,
`as.capacity-methods`,
`as.card.set.func-methods`,
`as.Mobius.set.func-methods`,
`as.Mobius.card.set.func-methods`,
`conjugate-methods`,
`interaction.indices-methods`,
`is.cardinal-methods`,
`is.kadditive-methods`,
`is.monotone-methods`,
`k.truncate.Mobius-methods`,
`Mobius-methods`,
`Shapley.value-methods`,
`to.data.frame-methods`.

Examples

```
## a set function
mu <- set.func(c(1:8,8:1)/8)

## the attributes of the object
```

```
mu@n
mu@data
mu@subsets

## some conversions that cannot work
## Not run: as.game(mu)
## Not run: as.capacity(mu)
## Not run: as.card.set.func(mu)

## some tests
is.cardinal(mu)
is.kadditive(mu,2)
is.monotone(mu)

## some transformations
conjugate(mu)
Mobius(mu)
k.truncate.Mobius(mu,2)

## summary
Shapley.value(mu)
interaction.indices(mu)
# the same
summary(mu)

## save the set function to a file
d <- to.data.frame(mu)
write.table(d,"my.set.func.csv",sep="\t")

# finally, some conversions that should work
mu <- set.func(c(0,1,1,1,2,2,2,3))
as.game(mu)
as.capacity(mu)
as.card.set.func(mu)
```

`set.func` *Create objects of class "set.func", "game", or "capacity".*

Description

These functions create objects of class `set.func`, `game`, or `capacity` from objects of class `numeric`.

Usage

```
set.func(object)
game(object)
capacity(object)
```

Arguments

object	An object of class numeric containing the coefficients of the set function in "natural" order. Its length has to be a power of 2.

Value

Return objects of class set.func, game, or capacity.

See Also

capacity-class,
game-class,
set.func-class.

Examples

```
set.func(-12:3)
game(c(0,rep(-1,15)))
capacity(0:15)
```

---

show-methods *Methods for Function show in Package 'kappalab'*

Description

Displays the contents of an object.

Methods

object = "superclass.set.func" Generic show method for superclass.set.func.

object = "set.func" Displays a set.func object.

object = "Mobius.set.func" Displays a Mobius.set.func object.

object = "summary.superclass.set.func" Displays a summary.superclass.set.func object.

object = "summary.superclass.capacity" Displays a summary.superclass.capacity object.

See Also

superclass.set.func-class,
set.func-class,
Mobius.set.func-class,
summary.superclass.set.func-class,
summary.superclass.capacity-class.

Examples

```
## a set function
mu <- set.func(0:15/15)
show(mu)
## the same
mu

## a Möbius transform
a <- Mobius.set.func(0:10,4,2)
show(a)
a

## a cardinal capacity
mu <- lower.capacity(5)
show(mu)
mu
```

---

summary-methods *Summary method*

Description

Computes indices that can be used to summarize a set function.

Methods

object = "superclass.capacity" Returns an object of class `summary.superclass.capacity`.

object = "superclass.set.func" Returns an object of class `summary.superclass.set.func`.

See Also

`summary.superclass.capacity-class`,
`summary.superclass.set.func-class`.

---

summary.superclass.capacity-class *Class "summary.superclass.capacity"*

Description

Class containing most of the indices that can be used to summarize an object of class `superclass.capacity`.

Objects from the Class

Objects are created by calls of the form `summary(...)` on objects of class `superclass.capacity`.

Slots

Shapley.value: Object of class `numeric` containing the Shapley value of a set function.

interaction.indices: Object of class `matrix` containing the Shapley interaction indices of a set function.

orness: Object of class `numeric` containg the orness degree of the Choquet integral corresponding to the underlying capacity.

veto: Object of class `numeric` containing the veto indices of the Choquet integral corresponding to the underlying capacity.

favor: Object of class `numeric` containing the favor indices of the Choquet integral corresponding to the underlying capacity.

variance: Object of class `numeric` containing the variance of the underlying capacity.

entropy: Object of class `numeric` containing the variance of the underlying capacity.

Methods

show `signature(object = "summary.superclass.capacity")`: ...

See Also

`superclass.capacity-class`,
`summary-methods`,
`Shapley.value-methods`,
`interaction.indices-methods`,
`orness-methods`,
`veto-methods`,
`favor-methods`,
`variance-methods`,
`entropy-methods`.

Examples

```
## a capacity
mu <- capacity(c(0:13,13,13)/13)
a <- Mobius(mu)

## its summary
summary(mu)
summary(a)
```

---

summary.superclass.set.func-class

Class "summary.superclass.set.func"

Description

Class containing most of the indices that can be used to summarize an object of class `superclass.set.func` and not of class `superclass.capacity`.

Objects from the Class

Objects are created by calls of the form `summary(...)` on objects of class `superclass.set.func` and not of class `superclass.capacity`.

Slots

`Shapley.value`: Object of class `numeric` containing the Shapley value of a set function.

`interaction.indices`: Object of class `matrix` containing the Shapley interaction indices of a set function.

Methods

show `signature(object = "summary.superclass.set.func")`

See Also

`superclass.set.func-class`,
`superclass.capacity-class`,
`summary-methods`,
`show-methods`,
`Shapley.value-methods`,
`interaction.indices-methods`.

Examples

```
## a capacity
mu <- set.func(c(0:13,13,13)/13)
a <- Mobius(mu)

## its summary
summary(mu)
summary(a)
```

`superclass.capacity-class`

Class "superclass.capacity"

Description

Virtual class, superclass of all `*.capacity` classes. Used to define a common `summary` method for `*.capacity` classes. Contains no slots.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

summary `signature(object = "superclass.capacity")`

See Also

summary-methods,
capacity-class,
Mobius.capacity-class,
card.capacity-class.

Examples

```
## three capacities
mu1 <- uniform.capacity(5)
mu2 <- capacity(c(0,0,0:13))
a <- Mobius(mu2)

## compute indices to summarize them
summary(mu1)
summary(mu2)
summary(a)
```

---

superclass.set.func-class

Class "superclass.set.func"

Description

Virtual class, superclass of all `*.set.func` classes. Used to define common `show` and `summary` methods for `*.set.func` classes.

Objects from the Class

A virtual Class: No objects may be created from it.

Slots

n: Object of class `numeric` of length 1 equal to the number of elements of the set on which the set function is defined.

data: Object of class `numeric` containing the coefficients of the set function.

Methods

show `signature(object = "superclass.set.func")`

round `signature(x = "superclass.set.func", digits = "numeric")`

summary `signature(object = "superclass.set.func")`

See Also

show-methods,
round-methods,
summary-methods,
set.func-class,
Mobius.set.func-class,
card.set.func-class,
Mobius.card.set.func-class.

Examples

```
## three set functions
mu1 <- card.set.func(-2:4)
mu2 <- set.func(c(-2,-2,-2:11/11))
mu3 <- Mobius.set.func(c(-7:6,6,6),4,4)

## print mu1
show(mu1)
## the same
mu1
## the others
mu2
mu3

## round mu2
round(mu2,2)

## compute indices to summarize them
summary(mu1)
summary(mu2)
summary(mu3)
```

---

to.data.frame-methods

Puts a set function under the form of a data.frame

Description

Puts a set function under the form of a `data.frame`. This function can be used to write a set function to a file.

Methods

object = "Mobius.card.set.func" Returns a `data.frame`.

object = "Mobius.set.func" Returns a `data.frame`.

object = "card.set.func" Returns a `data.frame`.

object = "set.func" Returns a `data.frame`.

Examples

```
## the Möbius representation of set function
a <- Mobius.set.func(-1:-16,4,4)

## to data.frame
d <- to.data.frame(a)
write.table(d,"my.set.func.csv",sep="\t")
```

`variance-methods` *Normalized variance of a capacity*

Description

Computes the normalized variance of a capacity. The capacity can be given either under the form of an object of class `capacity`, a `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" The normalized variance is computed from the Mobius transform of a capacity.

object = "capacity" The normalized variance is computed directly from a capacity.

object = "card.capacity" The normalized variance is computed from a cardinal capacity.

References

I. Kojadinovic (2005), *Minimum variance capacity identification*, European Journal of Operational Research, in press.

See Also

`capacity-class`,
`Mobius.capacity-class`,
`card.capacity-class`,
`Mobius-methods`.

Examples

```
## a capacity
mu <- capacity(c(0,0,0,0:12)/12)

## its Möbius transform
a <- Mobius(mu)

## their normalized variance
variance(mu)
variance(a)

## similar examples with card.capacity objects
```

```
mu <- lower.capacity(4)
variance(mu)
mu <- uniform.capacity(4)
variance(mu)
```

`veto-methods` *Veto indices*

Description

Computes the veto indices of a Choquet integral from the underlying **normalized** capacity. The capacity can be given either under the form of an object of class `capacity`, `card.capacity` or `Mobius.capacity`.

Methods

object = "Mobius.capacity" The veto indices are computed from the Mobius transform of a capacity.

object = "capacity" The veto indices are computed directly from a capacity.

object = "card.capacity" The veto indices are computed from a cardinal capacity.

References

J.-L. Marichal (2000), *Behavioral analysis of aggregation in multicriteria decision aid*, in: Preferences and Decisions under Incomplete Knowledge, J. Fodor and B. De Baets and P. Perny Eds, Physica-Verlag, pages 153-178.

J.-L. Marichal (2004), *Tolerant or intolerant character of interacting criteria in aggregation by the Choquet integral*, European Journal of Operational Research 155:3, pages 771-791.

See Also

`capacity-class`,
`Mobius.capacity-class`,
`card.capacity-class`,
`Mobius-methods`.

Examples

```
## a capacity
mu <- capacity(c(0:13,13,13)/13)

## its Möbius transform
a <- Mobius(mu)

## their veto indices
veto(mu)
veto(a)
```

```
## the same with a card.capacity object
mu <- lower.capacity(4)
veto(mu)
```

`zeta-methods` *The zeta transform*

Description

Computes the zeta transform of a set function given under the form of its Mobius transform. The zeta transform is the inverse of the Mobius transform.

Methods

object = "Mobius.capacity" Returns an object of class `capacity`.

object = "Mobius.card.set.func" Returns an object of class `card.set.func`.

object = "Mobius.game" Returns an object of class `game`.

object = "Mobius.set.func" Returns an object of class `set.func`.

References

G-C. Rota (1964), *On the foundations of combinatorial theory. I. Theory of Mobius functions*, Z. Wahrscheinlichkeitstheorie und Verw. Gebiete 2, pages 340-368.

A. Chateauneuf and J-Y. Jaffray (1989), *Some characterizations of lower probabilities and other monotone capacities through the use of Mobius inversion*, Mathematical Social Sciences 17, pages 263-283.

M. Grabisch, J-L. Marichal and M. Roubens (2000), *Equivalent representations of set functions*, Mathematics of Operations Research 25:2, pages 157-178.

See Also

`capacity-class`,
`card.set.func-class`,
`game-class`,
`set.func-class`,
`Mobius.capacity-class`,
`Mobius.card.set.func-class`,
`Mobius.game-class`,
`Mobius.set.func-class`,
`Mobius-methods`.

Examples

```
## the Möbius transform of a capacity
a <- Mobius.capacity(c(rep(0,15),1),4,4)
a

## its zeta transform
zeta(a)

## let us check
Mobius(zeta(a))

## a similar example with a Mobius.card.set.func object
mu <- card.set.func(-3:4)
a <- Mobius(mu)
zeta(a)
```

Index