Python package `pyfmtools` for handling and fitting fuzzy measures. Version 4.0 User Manual

Gleb Beliakov
`gleb@deakin.edu.au`

License agreement

Contents

Chapter 1

Introduction

This manual describes the programming library `pyfmtools` , which provides various tools for handling fuzzy measures, calculating various indices, Choquet and Sugeno integrals, as well as fitting fuzzy measures to empirical data. This package is designed for `Python` , but it also includes the C++ source files and this user manual.

Chapter 2 provides some background on fuzzy measures. A more detailed overview can be found in [4, 5, 12, 16] and references therein. Chapter 3 outlines computational methods used to fit fuzzy measures to empirical data. The description of the programming library `pyfmtools` is given in Chapter 4. Examples of its usage are provided in Section 4.6.

To cite `pyfmtools` package, use references [2–6, 21–24].

New in version 4

Random generation of fuzzy measures of different types, including k-additive, k-interactive, supermodular and submodular, sparse representation of k-additive fuzzy measures.

New in version 3

We added the concept of K-interactive fuzzy measures, and 4 methods of fitting K-interactive fuzzy measures from data based on linear programming. K-interactive fuzzy measures significantly reduce the computational complexity. We also added fitting fuzzy measures in marginal contribution representation and using maximal chains method, which fits only the values directly identifiable from the data. This method is useful for small data sets.

Fitting fuzzy measures in marginal contribution representation allows simple sub and supermodularity constraints, which can now be enforced.

See functions fittingKinteractive, fittingKinteractiveAuto, fittingKinteractiveMC, fittingKinteractiveMarginal, fittingKinteractiveMarginalMC.

We added calculation of new non-additivity and bipartition interaction indices. See functions Bipartition, BipartitionBanzhaf, NonadditivityIndex, NonadditivityIndexMob.

New in version 2

We added fitting K-maxitive and K-tolerant fuzzy measures, based on linear and mixed integer programming. See functions fittingktolerant and fittingK-maxitive.

We added a method for fitting sub-modular fuzzy measures reported in [3]. Supermodular fuzzy measure can also be fit by using duality: construct dual data set, fit a sub-modular fuzzy measure and then compute its dual. See function FuzzyMeasureFitLP.

We added an extra requirement of preservation of output ordering. See function FuzzyMeasureFitLP.

Fixed many warnings in the lpsolve code.

Chapter 2

Background on fuzzy measures

2.1 Preliminaries

Definition 1 (Aggregation function) *An aggregation function is a function of $n > 1$ arguments that maps the (n-dimensional) unit cube onto the unit interval $f : [0,1]^n \to [0,1]$, with the properties*

(i) $f(\underbrace{0, 0, \ldots, 0}_{n-times}) = 0$ *and* $f(\underbrace{1, 1, \ldots, 1}_{n-times}) = 1.$

(ii) $\mathbf{x} \leq \mathbf{y}$ *implies* $f(\mathbf{x}) \leq f(\mathbf{y})$ *for all* $\mathbf{x}, \mathbf{y} \in [0,1]^n$.

A large family of aggregation functions is based on Choquet and Sugeno integrals. The Choquet integral generalizes the Lebesgue integral, and like it, is defined with respect to a measure. We note that measures can be additive (the measure of a set is the sum of the measures of its non-intersecting subsets) or non-additive. Lengths, areas and volumes are examples of additive measures. Lebesgue integration is defined with respect to additive measures. If a measure is non-additive, then the measure of the total can be larger or smaller than the sum of the measures of its components.

Choquet and Sugeno integration are defined with respect to not necessarily additive monotone measures, called fuzzy measures (see Definition 2 below), or *capacities*. In this manual we are interested only in *discrete* fuzzy measures, which are defined on finite discrete subsets. This is because our construction of aggregation functions involves a finite set of inputs.

The main purpose of Choquet integral-based aggregation is to combine the inputs in such a way that not only the importance of individual inputs (as in weighted means), or of their magnitude (as in OWA), are taken into account, but also of their groups (or coalitions). For example, a particular input may not be important by itself, but become very important in the presence of some other inputs. In medical diagnosis, for instance, some symptoms by themselves may not be really important, but may become key factors in the presence of other signs.

A discrete fuzzy measure allows one to assign importances to all possible groups of criteria, and thus offers a much greater flexibility for modeling aggregation. It also turns out that weighted arithmetic means and OWA are special cases of Choquet integrals with respect to additive and symmetric fuzzy measures respectively. Thus we deal with a much broader class of aggregation functions. The uses of Choquet and Sugeno integrals as aggregation functions are documented in [11, 14, 15, 17, 18].

2.2 Basic definitions

Definition 2 (Fuzzy measure) *Let $\mathcal{N} = \{1, 2, \ldots, n\}$. A discrete fuzzy measure is a set function*[1] $v : 2^{\mathcal{N}} \to [0, 1]$ *which is monotonic (i.e. $v(\mathcal{A}) \leq v(\mathcal{B})$ whenever $\mathcal{A} \subset \mathcal{B}$) and satisfies $v(\emptyset) = 0$ and $v(\mathcal{N}) = 1$.*

In the context of aggregation functions, we are interested in the interpretation of the values of a fuzzy measure as the importance of a coalition. In the Definition 2, a subset $\mathcal{A} \subseteq \mathcal{N}$ can be considered as a *coalition*, so that $v(\mathcal{A})$ gives us an idea about the importance or the weight of this coalition. The monotonicity condition implies that adding new elements to a coalition does not decrease its weight.

Definition 3 (Möbius transformation) *Let v be a fuzzy measure. The Möbius transformation of v is a set function defined for every $\mathcal{A} \subseteq \mathcal{N}$ as*

$$\mathcal{M}(\mathcal{A}) = \sum_{\mathcal{B} \subseteq \mathcal{A}} (-1)^{|\mathcal{A} \setminus \mathcal{B}|} v(\mathcal{B}).$$

Möbius transformation is invertible, and one recovers v by using its inverse, called ***Zeta transform***,

$$v(\mathcal{A}) = \sum_{\mathcal{B} \subseteq \mathcal{A}} \mathcal{M}(B) \quad \forall \mathcal{A} \subseteq \mathcal{N}.$$

[1] A set function is a function whose domain consists of all possible subsets of $\mathcal{N}$. For example, for $n = 3$, a set function is specified by $2^3 = 8$ values at $v(\emptyset)$, $v(\{1\})$, $v(\{2\})$, $v(\{3\})$, $v(\{1, 2\})$, $v(\{1, 3\})$, $v(\{2, 3\})$, $v(\{1, 2, 3\})$.

Möbius transformation is helpful in expressing various quantities, like the interaction indices discussed later, in a more compact form. It also serves as an alternative representation of a fuzzy measure, called Möbius representation. That is, one can either use v or $\mathcal{M}$ to perform calculations, whichever is more convenient. The conditions of monotonicity of a fuzzy measure, and the boundary conditions $v(\emptyset) = 0, v(\mathcal{N}) = 1$ are expressed, respectively, as

$$\sum_{\mathcal{B}\subseteq A | i\in\mathcal{B}} \mathcal{M}(\mathcal{B}) \geq 0, \qquad \text{for all } \mathcal{A}\subseteq\mathcal{N} \text{ and all } i\in\mathcal{A}, \tag{2.1}$$

$$\mathcal{M}(\emptyset) = 0 \text{ and } \sum_{\mathcal{A}\subseteq\mathcal{N}} \mathcal{M}(\mathcal{A}) = 1.$$

There are various special classes of fuzzy measures, which we discuss below. We now proceed with the definition of the Choquet integral–based aggregation functions.

Definition 4 (Discrete Choquet integral) *The discrete Choquet integral with respect to a fuzzy measure v is given by*

$$C_v(\mathbf{x}) = \sum_{i=1}^{n} x_{(i)}[v(\{j | x_j \geq x_{(i)}\}) - v(\{j | x_j \geq x_{(i+1)}\})], \tag{2.2}$$

where $\mathbf{x}_{\nearrow} = (x_{(1)}, x_{(2)}, \ldots, x_{(n)})$ is a non-decreasing permutation of the input $\mathbf{x}$, and $x_{(n+1)} = \infty$ by convention.

Alternative expressions

- By rearranging the terms of the sum, (2.2) can also be written as

$$C_v(\mathbf{x}) = \sum_{i=1}^{n} \left[x_{(i)} - x_{(i-1)}\right] v(H_i). \tag{2.3}$$

  where $x_{(0)} = 0$ by convention, and $H_i = \{(i), \ldots, (n)\}$ is the subset of indices of the $n - i + 1$ largest components of $\mathbf{x}$.

- The Choquet integral can be expressed as

$$\mathcal{C}_v(\mathbf{x}) = \sum_{\mathcal{A}\subseteq\mathcal{N}} v(\mathcal{A}) g_{\mathcal{A}}(\mathbf{x}), \tag{2.4}$$

  where the basis functions are

$$g_{\mathcal{A}}(\mathbf{x}) = \max(0, \min_{i\in\mathcal{A}} x_i - \max_{i\in\mathcal{N}\setminus\mathcal{A}} x_i). \tag{2.5}$$

- The Choquet integral can be expressed with the help of the Möbius transformation as

$$C_v(\mathbf{x}) = \sum_{\mathcal{A} \subseteq \mathcal{N}} \mathcal{M}(\mathcal{A}) \min_{i \in \mathcal{A}} x_i = \sum_{\mathcal{A} \subseteq \mathcal{N}} \mathcal{M}(\mathcal{A}) h_{\mathcal{A}}(\mathbf{x}), \tag{2.6}$$

with $h_{\mathcal{A}}(\mathbf{x}) = \min_{i \in \mathcal{A}} x_i$.

Main properties

- The Choquet integral is a continuous piecewise linear idempotent aggregation function;
- An aggregation function is a Choquet integral if and only if it is homogeneous, shift-invariant and *comonotone additive*, i.e., $C_v(\mathbf{x}+\mathbf{y}) = C_v(\mathbf{x}) + C_v(\mathbf{y})$ for all comonotone [2] $\mathbf{x}, \mathbf{y}$;
- The Choquet integral is uniquely defined by its values at the vertices of the unit cube $[0,1]^n$, i.e., at the points $\mathbf{x}$, whose coordinates $x_i \in \{0,1\}$. Note that there are 2^n such points, the same as the number of values that determine the fuzzy measure v;
- The discrete Choquet integral is a linear function of the values of the fuzzy measure v.
- The class of Choquet integrals includes weighted means and OWA functions, as well as minimum, maximum and order statistics as special cases;
- A linear convex combination of Choquet integrals with respect to fuzzy measures v_1 and v_2, $\alpha C_{v_1} + (1-\alpha)C_{v_2}, \alpha \in [0,1]$, is also a Choquet integral with respect to $v = \alpha v_1 + (1-\alpha)v_2$.

Calculation

Calculation of the discrete Choquet integral is performed using Equation (2.3) using the following procedure. Consider the vector of pairs $((x_1, 1), (x_2, 2), \ldots, (x_n, n))$, where the second component of each pair is

[2] Two vectors $\mathbf{x}, \mathbf{y} \in \Re^n$ are called comonotone if there exists a common permutation P of $\{1, 2, \ldots, n\}$, such that $x_{P(1)} \leq x_{P(2)} \leq \cdots \leq x_{P(n)}$ and $y_{P(1)} \leq y_{P(2)} \leq \cdots \leq y_{P(n)}$. Equivalently, this condition is frequently expressed as $(x_i - x_j)(y_i - y_j) \geq 0$ for all $i, j \in \{1, \ldots, n\}$.

just the index i of x_i. The second component will help keeping track of all permutations.

Calculation of $C_v(\mathbf{x})$.

1. Sort the components of $((x_1, 1), (x_2, 2), \dots, (x_n, n))$ with respect to the first component of each pair in non-decreasing order. We obtain $((x_{(1)}, i_1), (x_{(2)}, i_2), \dots, (x_{(n)}, i_n))$, so that $x_{(j)} = x_{i_j}$ and $x_{(j)} \le x_{(j+1)}$ for all i. Let also $x_{(0)} = 0$.
2. Let $\mathcal{T} = \{1, \dots, n\}$, and $S = 0$.
3. For $j = 1, \dots, n$ do
   (a) $S := S + [x_{(j)} - x_{(j-1)}]v(\mathcal{T})$;
   (b) $\mathcal{T} := \mathcal{T} \setminus \{i_j\}$
4. Return S.

For computational purposes it is convenient to store the values of a fuzzy measure v in an array $\mathbf{v}$ of size 2^n, and to use the following indexing system, which provides a one-to-one mapping between the subsets $\mathcal{J} \subseteq \mathcal{N}$ and the set of integers $I = \{0, \dots, 2^n - 1\}$, which index the elements of v. Take the binary representation of each index in I, e.g. $j = 5 = 101$ (binary). Now for a given subset $\mathcal{J} \subseteq \mathcal{N} = \{1, \dots, n\}$ define its characteristic vector $\mathbf{c} \in \{0,1\}^n : c_{n-i+1} = 1$ if $i \in \mathcal{J}$ and 0 otherwise. For example, if $n = 5$, $\mathcal{J} = \{1, 3\}$, then $\mathbf{c} = (0, 0, 1, 0, 1)$. Put the value $v(\mathcal{J})$ into correspondence with v_j, so that the binary representation of j corresponds to the characteristic vector of $\mathcal{J}$. In our example $v(\{1, 3\}) = v_5$.

Such an ordering of the subsets of $\mathcal{N}$ is called binary ordering:

$$\emptyset, \{1\}, \{2\}, \{1,2\}, \{3\}, \{1,3\}, \{2,3\}, \{1,2,3\}, \{4\}, \dots, \{1,2,\dots,n\}.$$

The values of v are mapped to the elements of vector $\mathbf{v}$ as follows

v_0 $= v_{(0000)}$	v_1 $= v_{(0001)}$	v_2 $= v_{(0010)}$	v_3 $= v_{(0011)}$	v_4 $= v_{(0100)}$	v_5 $= v_{(0101)}$	...
$v(\emptyset)$	$v(\{1\})$	$v(\{2\})$	$v(\{1,2\})$	$v(\{3\})$	$v(\{1,3\})$	...

An alternative ordering of the values of v is based on set cardinality:

$$\emptyset, \underbrace{\{1\}, \{2\}, \dots, \{n\}}_{n \text{ singletons}}, \underbrace{\{1,2\}, \{1,3\}, \dots, \{1,n\}, \{2,3\}, \dots, \{n-1,n\}}_{\binom{n}{2} \text{ pairs}}, \{1,2,3\}, \dots .$$

Such an ordering is useful when dealing with k-additive fuzzy measures (see Definition 16 and Proposition 1 below), as it allows one to group non-zero values $\mathcal{M}(\mathcal{A})$ (in Möbius representation) at the beginning of the array.

2.3 Types of fuzzy measures

In this section we present the most important definitions and classes of fuzzy measures.

Definition 5 (Dual fuzzy measure) *Given a fuzzy measure v, its dual fuzzy measure v^* is defined by*

$$v^*(\mathcal{A}) = 1 - v(\mathcal{A}^c), \text{ for all } \mathcal{A} \subseteq \mathcal{N},$$

where $\mathcal{A}^c = \mathcal{N} \setminus \mathcal{A}$ is the complement of $\mathcal{A}$ in $\mathcal{N}$.

Definition 6 (Self–dual fuzzy measure) *A fuzzy measure v is self-dual if it is equal to its dual $v*$, i.e.,*

$$v(\mathcal{A}) + v(\mathcal{A}^c) = 1, \text{holds for all } A \subseteq \mathcal{N} \ .$$

Definition 7 (Submodular and supermodular fuzzy measure) *A fuzzy measure v is called submodular if for any $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}$*

$$v(\mathcal{A} \cup \mathcal{B}) + v(\mathcal{A} \cap \mathcal{B}) \le v(\mathcal{A}) + v(\mathcal{B}). \tag{2.7}$$

It is called supermodular if

$$v(\mathcal{A} \cup \mathcal{B}) + v(\mathcal{A} \cap \mathcal{B}) \ge v(\mathcal{A}) + v(\mathcal{B}). \tag{2.8}$$

Two weaker conditions which are frequently used are called sub- and super-additivity. These are special cases of sub- and supermodularity for disjoint subsets

Definition 8 (Subadditive and superadditive fuzzy measure) *A fuzzy measure v is called subadditive if for any two nonintersecting subsets $\mathcal{A}, \mathcal{B} \subset \mathcal{N}$, $\mathcal{A} \cap \mathcal{B} = \emptyset$:*

$$v(\mathcal{A} \cup \mathcal{B}) \le v(\mathcal{A}) + v(\mathcal{B}). \tag{2.9}$$

It is called superadditive if

$$v(\mathcal{A} \cup \mathcal{B}) \ge v(\mathcal{A}) + v(\mathcal{B}). \tag{2.10}$$

Note 1 *A general fuzzy measure may be submodular only with respect to specific pairs of subsets $\mathcal{A}, \mathcal{B}$, and supermodular with respect to other pairs.*

Definition 9 (Additive (probability) measure) *A fuzzy measure v is called* ***additive*** *if for any $\mathcal{A}, \mathcal{B} \subset \mathcal{N}$, $\mathcal{A} \cap \mathcal{B} = \emptyset$:*

$$v(\mathcal{A} \cup \mathcal{B}) = v(\mathcal{A}) + v(\mathcal{B}). \tag{2.11}$$

An additive fuzzy measure is called a **probability** *measure.*

Note 2 *For an additive fuzzy measure clearly $v(\mathcal{A}) = \sum_{i \in \mathcal{A}} v(\{i\})$.*

Note 3 *Additivity implies that for any subset $\mathcal{A} \subseteq \mathcal{N} \setminus \{i, j\}$*

$$v(\mathcal{A} \cup \{i, j\}) = v(\mathcal{A} \cup \{i\}) + v(\mathcal{A} \cup \{j\}) - v(\mathcal{A}).$$

Definition 10 (Balanced measure) *A fuzzy measure v is called balanced if it holds:*

$$\mid \mathcal{A} \mid < \mid \mathcal{B} \mid \Longrightarrow v(\mathcal{A}) \leq v(\mathcal{B}), \text{ for all } \mathcal{A}, \mathcal{B} \subseteq \mathcal{N}.$$

Definition 11 (Symmetric fuzzy measure) *A fuzzy measure v is called symmetric if the value $v(\mathcal{A})$ depends only on the cardinality of the set $\mathcal{A}$, i.e., for any $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}$,*

$$\text{if } |\mathcal{A}| = |\mathcal{B}| \text{ then } v(\mathcal{A}) = v(\mathcal{B}).$$

Alternatively, one can say that a fuzzy measure v is symmetric if for any $\mathcal{A} \subseteq \mathcal{N}$ it is

$$v(\mathcal{A}) = Q\left(\frac{|\mathcal{A}|}{n}\right), \tag{2.12}$$

for some monotone non-decreasing function $Q : [0, 1] \to [0, 1], Q(0) = 0$ and $Q(1) = 1$.

Definition 12 (Possibility and necessity measures) *A fuzzy measure is called a possibility, Pos, if for all $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}$ it satisfies*

$$Pos(\mathcal{A} \cup \mathcal{B}) = \max\{Pos(\mathcal{A}), Pos(\mathcal{B})\}.$$

A fuzzy measure is called a necessity, Nec, if for all $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}$ it satisfies

$$Nec(\mathcal{A} \cap \mathcal{B}) = \min\{Nec(\mathcal{A}), Nec(\mathcal{B})\}.$$

Note 4 *Possibility and necessity measures are dual to each other in the sense of Definition 5, that is, for all* $\mathcal{A} \subseteq \mathcal{N}$

$$Nec(\mathcal{A}) = 1 - Pos(\mathcal{A}^c).$$

A possibility measure is subadditive. A necessity measure is superadditive.

Definition 13 (Belief Measure) *A belief measure* $Bel : 2^{\mathcal{N}} \to [0,1]$ *is a fuzzy measure that satisfies the following condition: for all* $m > 1$

$$Bel(\bigcup_{i=1}^{m} \mathcal{A}_i) \geq \sum_{\emptyset \neq I \subset \{1,\dots,m\}} (-1)^{|I|+1} Bel(\bigcap_{i \in I} \mathcal{A}_i),$$

where $\{\mathcal{A}_i\}_{i \in \{1,\dots,m\}}$, *is any finite family of subsets of* $\mathcal{N}$. [3]

Definition 14 (Plausibility measure) *A plausibility measure* $Pl : 2^{\mathcal{N}} \to [0,1]$ *is a fuzzy measure that satisfies the following condition: for all* $m > 1$

$$Pl(\bigcap_{i=1}^{m} \mathcal{A}_i) \leq \sum_{\emptyset \neq I \subset \{1,\dots,m\}} (-1)^{|I|+1} Pl(\bigcup_{i \in I} \mathcal{A}_i),$$

where $\{\mathcal{A}_i\}_{i \in \{1,\dots,m\}}$ *is any finite family of subsets of* $\mathcal{N}$.

Note 5 *A set function* $Pl : 2^{\mathcal{N}} \to [0,1]$ *is a plausibility measure if its dual set function is a belief measure, i.e., for all* $\mathcal{A} \subseteq \mathcal{N}$

$$Pl(\mathcal{A}) = 1 - Bel(\mathcal{A}^c).$$

Any belief measure is superadditive. Any plausibility measure is subadditive.

λ-fuzzy measures

Additive and symmetric fuzzy measures are two examples of very simple fuzzy measures, whereas general fuzzy measures are sometimes too complicated for applications. As a way of reducing the complexity of a fuzzy measure Sugeno [20] introduced λ-fuzzy measures (also called Sugeno measures).

[3] For a fixed $m \geq 1$ this condition is called m-monotonicity (simple monotonicity for $m = 1$), and if it holds for all $m \geq 1$, it is called total monotonicity. For a fixed m, condition in Definition 14 is called m-alternating monotonicity. 2-monotone fuzzy measures are called supermodular (see Definition 7), also called convex, whereas 2-alternating fuzzy measures are called submodular. If a fuzzy measure is m-monotone, its dual is m-alternating and vice versa.

Definition 15 (λ-fuzzy measure) *Given a parameter $\lambda \in]-1, \infty[$, a λ-fuzzy measure is a fuzzy measure v that for all $\mathcal{A}, \mathcal{B} \subseteq \mathcal{N}, \mathcal{A} \cap \mathcal{B} = \emptyset$ satisfies*

$$v(\mathcal{A} \cup \mathcal{B}) = v(\mathcal{A}) + v(\mathcal{B}) + \lambda v(\mathcal{A}) v(\mathcal{B}). \tag{2.13}$$

Under these conditions, all the values $v(\mathcal{A})$ are immediately computed from n independent values $v(\{i\}), i = 1, \ldots, n$, by using the explicit formula

$$v(\bigcup_{i=1}^{m} \{i\}) = \frac{1}{\lambda} \left(\prod_{i=1}^{m} (1 + \lambda v(\{i\})) - 1 \right), \quad \lambda \neq 0.$$

If $\lambda = 0$, λ-fuzzy measure becomes a probability measure. The coefficient λ is determined from the boundary condition $v(\mathcal{N}) = 1$, which gives

$$\lambda + 1 = \prod_{i=1}^{n} (1 + \lambda v(\{i\})), \tag{2.14}$$

which can be solved on $(-1, 0)$ or $(0, \infty)$ numerically (note that $\lambda = 0$ is always a solution). Thus a λ-fuzzy measure is characterized by n independent values $v(\{i\}), i = 1, \ldots, n$.

Note 6 *A λ-fuzzy measure is either sub- or supermodular, when $-1 < \lambda \leq 0$ or $\lambda \geq 0$ respectively.*

Note 7 *When $-1 < \lambda \leq 0$, a λ-fuzzy measure is a plausibility measure, and when $\lambda \geq 0$ it is a belief measure.*

k - order fuzzy measures

Another way to reduce complexity of aggregation functions based on fuzzy measures is to impose various linear constraints on their values. Such constraints acquire an interesting interpretation in terms of interaction indices discussed in the next section. One type of constraints leads to k-additive fuzzy measures.

Definition 16 (k-additive fuzzy measure) *A fuzzy measure v is called k-additive ($1 \leq k \leq n$) if its Möbius transformation verifies*

$$\mathcal{M}(\mathcal{A}) = 0$$

for any subset $\mathcal{A}$ with more than k elements, $|\mathcal{A}| > k$, and there exists a subset $\mathcal{B}$ with k elements such that $\mathcal{M}(\mathcal{B}) \neq 0$.

Definition 17 (Possibilistic Möbius transform) *The possibilistic Möbius transform of a fuzzy measure v on $\mathcal{N}$ is a mapping $m_p : \mathcal{P}(\mathcal{N}) \to [0,1]$ defined by*

$$m_p(\mathcal{A}) = \begin{cases} v(\mathcal{A}) & \text{if } v(\mathcal{A}) > \max_{\mathcal{B} \subset \mathcal{A}} v(\mathcal{B}), \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

The possibilistic Zeta transform of m_p is the mapping $Z_{m_p} : \mathcal{P}(\mathcal{N}) \to [0,1]$ defined by:

$$\mathcal{Z}_{m_p}(\mathcal{A}) = \max_{\mathcal{B} \subseteq \mathcal{A}} m_p(\mathcal{B}). \tag{2.16}$$

Definition 18 (k-maxitive fuzzy measure) *A fuzzy measure v is called k-maxitive if its possibilistic Möbius transform satisfies $m_p(\mathcal{A}) = 0$ for any $\mathcal{A}$ such that $|\mathcal{A}| > k$ and there exists at least one subset $\mathcal{A}$ of $\mathcal{N}$ of exactly k elements such that $m_p(\mathcal{A}) \neq 0$.*

Definition 19 (k-tolerant fuzzy measure) *Let $k \in \{1, 2, ..., n\} = \mathcal{N}$. A fuzzy measure on $\mathcal{N}$ is k-tolerant if $v(\mathcal{A}) = 1$ for all $\mathcal{A} \subseteq \mathcal{N}$ such that $|\mathcal{A}| \geq k$ and there exists a subset $\mathcal{B} \subseteq \mathcal{N}$, with $|\mathcal{B}| = k - 1$, such that $v(\mathcal{B}) \neq 1$. A fuzzy measure v on $\mathcal{N}$ is k-intolerant if $v(\mathcal{A}) = 0$ for all $\mathcal{A} \subseteq \mathcal{N}$ such that $|\mathcal{A}| \leq n - k$ and there exists a subset $\mathcal{B} \subseteq \mathcal{N}$, with $|\mathcal{B}| = n - k + 1$, such that $v(\mathcal{B}) \neq 0$.*

k-intolerant fuzzy measures can be obtained from k-tolerant measures by using duality. The Choquet integral with respect to a k-tolerant capacity is independent of the first $n - k$ smallest inputs.

k - interactive fuzzy measures

A recent approach to reduce both the number of variables and constraints by fixing the values of the fuzzy measure for all subsets of cardinality greater than k in some appropriate way was presented in [6]. The values of fuzzy measure $v(\mathcal{A})$ are fixed (in some way) for all $|\mathcal{A}| > k$. We need to ensure that these values are: a) consistent with the semantics of the problem, and b) consistent with the values at smaller subsets, which will be fitted to the data.

Let is fix the maximal value KC of a fuzzy measure at any subset of cardinality k, that is $v(\mathcal{A}) \leq KC$ for all $\mathcal{A}, |\mathcal{A}| = k$. Our first step is to define the values of $v(\mathcal{B}), |\mathcal{B}| > k$ in a suitable way. To do this, we fix the values $v(\mathcal{B}) = KC$ for all $\mathcal{B}, |\mathcal{B}| = k+1$, and then define the values at the larger sets by maximising the entropy. This results in

$$v(\mathcal{A}) = KC + \frac{a-k-1}{n-k-1}(1-KC), \quad \text{for all } \mathcal{A} : |\mathcal{A}| > k. \tag{2.17}$$

The Choquet integral with respect to a fuzzy measure with coefficients given by (2.17) for $|\mathcal{A}| > k$ can be written as

$$\mathcal{C}_v(\mathbf{x}) = \frac{1-K}{n-k-1} \sum_{i=1}^{n-k-1} x_{(i)} + Kx_{(n-k)} + \sum_{\mathcal{A} \subseteq \mathcal{N}, |\mathcal{A}| \leq k} v(\mathcal{A}) g_{\mathcal{A}}(\mathbf{x}). \tag{2.18}$$

We see that the contribution of the $n-k-1$ smallest inputs is averaged with the arithmetic mean while the interactions are accounted for the remaining inputs.

This type of fuzzy measures is called k-interactive in [6]. The k largest inputs exhibit unrestricted interaction (in terms of redundancy or complementarity) whereas the rest of the inputs are averaged in a symmetric way (as in OWA functions), and with our choice in (2.17), as OWA with equal weights, making it the arithmetic mean. There is still some interaction of the inputs in the larger subsets in terms of non-zero interaction indices, but these interactions are determined fully by interactions in the smaller subsets and the values KC and k.

Numerical experiments in [6] show that k-interactive fuzzy measures are much easier to fit to the data even for $n > 10$ because both the number of variables and monotonicity constraints are reduced, but also the number of non-zero coefficients in the matrix of constraints is reduced drastically. This means that the actual monotonicity constraints are much simpler than in the case of k-additive fuzzy measures.

2.4 Interaction, importance and other indices

When dealing with multiple criteria, it is often the case that these are not independent, and there is some interaction (positive or negative) among the criteria. For instance, two or more criteria may point essentially to the same concept, for example criteria such as "learnability" and "memorability" that are used to evaluate software user interface. If the criteria are combined by using, e.g., weighted means, their scores will be double counted. In other instances, contribution of one criterion to the total score by itself may be small, but sharply rise when taken in conjunction with other criteria (i.e., in a "coalition").

Thus to measure such concepts as the importance of a criterion and interaction among the criteria, we need to account for contribution of these criteria in various coalitions. To do this we will use the concepts of Shapley value, which measures the importance of a criterion i in all possible coalitions, and the interaction index, which measures the interaction of a pair of criteria i, j in all possible coalitions [12, 13].

Definition 20 (Shapley value) *Let v be a fuzzy measure. The Shapley index for every $i \in \mathcal{N}$ is*

$$\phi(i) = \sum_{\mathcal{A} \subseteq \mathcal{N} \setminus \{i\}} \frac{(n - |\mathcal{A}| - 1)!|\mathcal{A}|!}{n!} [v(\mathcal{A} \cup \{i\}) - v(\mathcal{A})].$$

The Shapley value is the vector $\boldsymbol{\phi}(v) = (\phi(1), \ldots, \phi(n))$.

The Shapley value is interpreted as a kind of average value of the contribution of each criterion alone in all coalitions.

Definition 21 (Interaction index) *Let v be a fuzzy measure. The interaction index for every pair $i, j \in \mathcal{N}$ is*

$$I_{ij} = \sum_{\mathcal{A} \subseteq \mathcal{N} \setminus \{i,j\}} \frac{(n - |\mathcal{A}| - 2)!|\mathcal{A}|!}{(n-1)!} [v(\mathcal{A} \cup \{i,j\}) - v(\mathcal{A} \cup \{i\}) - v(\mathcal{A} \cup \{j\}) + v(\mathcal{A})].$$

The interaction indices verify $I_{ij} < 0$ as soon as i, j are positively correlated (negative synergy, redundancy). Similarly $I_{ij} > 0$ for negatively correlated criteria (positive synergy, complementarity). $I_{ij} \in [-1, 1]$ for any pair i, j.

Note 8 *For a submodular fuzzy measure v, all interaction indices verify $I_{ij} \leq 0$. For a supermodular fuzzy measure, all interaction indices verify $I_{ij} \geq 0$.*

Definition 22 (Interaction index for coalitions) *Let v be a fuzzy measure. The interaction index for every set $\mathcal{A} \subseteq \mathcal{N}$ is*

$$I(\mathcal{A}) = \sum_{\mathcal{B} \subseteq \mathcal{N} \setminus \mathcal{A}} \frac{(n - |\mathcal{B}| - |\mathcal{A}|)!|\mathcal{B}|!}{(n - |\mathcal{A}| + 1)!} \sum_{\mathcal{C} \subseteq \mathcal{A}} (-1)^{|\mathcal{A} \setminus \mathcal{C}|} v(\mathcal{B} \cup \mathcal{C}).$$

Note 9 *Clearly $I(\mathcal{A})$ coincides with I_{ij} if $\mathcal{A} = \{i, j\}$, and coincides with $\phi(i)$ if $\mathcal{A} = \{i\}$.*

An alternative to the Shapley value is the Banzhaf index [1]. It measures the same concept as the Shapley index, but weights the terms $[v(\mathcal{A} \cup \{i\}) - v(\mathcal{A})]$ in the sum equally.

Definition 23 (Banzhaf Index) *Let v be a fuzzy measure. The Banzhaf index b_i for every $i \in \mathcal{N}$ is*

$$b_i = \frac{1}{2^{n-1}} \sum_{\mathcal{A} \subseteq \mathcal{N} \setminus \{i\}} [v(\mathcal{A} \cup \{i\}) - v(\mathcal{A})].$$

Definition 24 (Banzhaf interaction index for coalitions) *Let v be a fuzzy measure. The Banzhaf interaction index between the elements of $\mathcal{A} \subseteq \mathcal{N}$ is given by*

$$J(\mathcal{A}) = \frac{1}{2^{n-|\mathcal{A}|}} \sum_{\mathcal{B} \subseteq \mathcal{N} \setminus \mathcal{A}} \sum_{\mathcal{C} \subseteq \mathcal{A}} (-1)^{|\mathcal{A} \setminus \mathcal{C}|} v(\mathcal{B} \cup \mathcal{C}).$$

Note 10 *Möbius transformation help one to express the indices mentioned above in a more compact form [12, 13, 16, 18], namely*

$$\phi(i) = \sum_{\mathcal{B} |\, i \in \mathcal{B}} \frac{1}{|\mathcal{B}|} \mathcal{M}(\mathcal{B}),$$

$$I(\mathcal{A}) = \sum_{\mathcal{B} | \mathcal{A} \subseteq \mathcal{B}} \frac{1}{|\mathcal{B}| - |\mathcal{A}| + 1} \mathcal{M}(\mathcal{B}),$$

$$J(\mathcal{A}) = \sum_{\mathcal{B} | \mathcal{A} \subseteq \mathcal{B}} \frac{1}{2^{|\mathcal{B}| - |\mathcal{A}|}} \mathcal{M}(\mathcal{B}).$$

The next result due to Grabisch [12,13] establishes a fundamental property of k-additive fuzzy measures, which justifies their use in simplifying interactions between the criteria in multiple criteria decision making.

Proposition 1 *Let v be a k-additive fuzzy measure, $1 \leq k \leq n$. Then*

- *$I(\mathcal{A}) = 0$ for every $\mathcal{A} \subseteq \mathcal{N}$ such that $|\mathcal{A}| > k$;*
- *$I(\mathcal{A}) = J(\mathcal{A}) = \mathcal{M}(\mathcal{A})$ for every $\mathcal{A} \subseteq \mathcal{N}$ such that $|\mathcal{A}| = k$.*

Thus k-additive measures acquire an interesting interpretation. These are fuzzy measures that limit interaction among the criteria to groups of size at most k. For instance, for 2-additive fuzzy measures, there are pairwise interactions among the criteria but no interactions in groups of 3 or more. By limiting the class of fuzzy measures to k-additive measures, one reduces their complexity (the number of values) by imposing linear equality constraints. The total number of linearly independent values is reduced from $2^n - 1$ to $\sum_{i=1}^{k} \binom{n}{i} - 1$.

Orness value

The *measure of orness*, also called the *degree of orness, orness value* or *attitudinal character*, is an important numerical characteristic of averaging aggregation functions. Basically, the measure of orness measures how far a given averaging function is from the max function, which is the weakest disjunctive function. The measure of orness is computed for any averaging function [9, 10] using

Definition 25 (Measure of orness) *Let f be an averaging aggregation function. Then its measure of orness is*

$$orness(f) = \frac{\int_{[0,1]^n} f(\mathbf{x})d\mathbf{x} - \int_{[0,1]^n} \min(\mathbf{x})d\mathbf{x}}{\int_{[0,1]^n} \max(\mathbf{x})d\mathbf{x} - \int_{[0,1]^n} \min(\mathbf{x})d\mathbf{x}}. \tag{2.19}$$

Clearly, $orness(\max) = 1$ and $orness(\min) = 0$, and for any f, $orness(f) \in [0, 1]$. The calculation of the integrals of max and min functions results in simple equations

$$\int_{[0,1]^n} \max(\mathbf{x})d\mathbf{x} = \frac{n}{n+1} \text{ and } \int_{[0,1]^n} \min(\mathbf{x})d\mathbf{x} = \frac{1}{n+1}. \tag{2.20}$$

By using the Möbius transform one can calculate the orness of a Choquet integral C_v with respect to a fuzzy measure v as follows.

Proposition 2 (Orness of Choquet integral) *[19] For any fuzzy measure v the orness of the Choquet integral with respect to v is*

$$orness(C_v) = \frac{1}{n-1} \sum_{\mathcal{A} \subseteq \mathcal{N}} \frac{n - |\mathcal{A}|}{|\mathcal{A}| + 1} \mathcal{M}(\mathcal{A}),$$

where $\mathcal{M}(\mathcal{A})$ is the Möbius representation of $\mathcal{A}$. In terms of v the orness value is

$$orness(C_v) = \frac{1}{n-1} \sum_{\mathcal{A} \subseteq \mathcal{N}} \frac{(n - |\mathcal{A}|)!|\mathcal{A}|!}{n!} v(\mathcal{A}).$$

Nonadditivity and bipartition indices

The following index was proposed in [21]. It measures the degree of non-additivity in every subset.

Definition 26 (Nonadditivity index) *Let v be a capacity on $\mathcal{N}$. The nonadditivity index of subset $\mathcal{A} \subseteq \mathcal{N}$, $|\mathcal{A}| \geq 2$, with respect to v is defined as*

$$n_v(A) = \frac{1}{2^{|\mathcal{A}|-1} - 1} \sum_{\substack{(\mathcal{B}, \mathcal{A} \backslash \mathcal{B}) \\ \emptyset \neq \mathcal{B} \subset \mathcal{A}}} [v(\mathcal{A}) - v(\mathcal{B}) - v(\mathcal{A} \backslash \mathcal{B})].$$

Here $\star$-additive means "additive, subadditive, superadditive". $\overset{\star}{=} 0$ means $=, \leq, \geq$.

Theorem 1 *Let v be a capacity on $\mathcal{N}$. If v is $\star$-additive, then $n_v(A) \overset{\star}{=} 0$, $\forall \mathcal{A} \subseteq \mathcal{N}$ and $|\mathcal{A}| \geq 2$.*

The nonadditivity index can be computed by an alternative formula

$$n_v(\mathcal{A}) = v(\mathcal{A}) - \frac{1}{2^{|\mathcal{A}|-1} - 1} \sum_{\emptyset \neq B \subset A} v(B), \forall \mathcal{A} \subseteq \mathcal{N}, |\mathcal{A}| \geq 2 \tag{2.21}$$

It can be expressed in the Möbius representation as

$$n_v(\mathcal{A}) = \sum_{\mathcal{C} \subseteq \mathcal{A}} \frac{2^{|\mathcal{A}|-1} - 2^{|\mathcal{A}|-|\mathcal{C}|}}{2^{|\mathcal{A}|-1} - 1} \mathcal{M}(\mathcal{C}), \forall \mathcal{A} \subseteq \mathcal{N}.$$

The following index was proposed in [24]. The sign of this index is consistent with the sub- and super-additivity of the fuzzy measure, and it ranges in [-1,1] for all subsets.

Definition 27 (Bipartition Shapley and Banzhaf indices) *The Shapley bipartition interaction index of a subset $\mathcal{A} \subseteq \mathcal{N}$ is defined as*

$$\hat{I}^v_{\mathrm{Sh}}(\mathcal{A}) = \sum_{\mathcal{B} \subseteq \mathcal{N} \backslash \mathcal{A}} \frac{1}{|\mathcal{N}| - |\mathcal{A}| + 1} \binom{|\mathcal{N}| - |\mathcal{A}|}{|\mathcal{B}|}^{-1} \hat{\Delta}_A v(\mathcal{B}).$$

The Banzhaf bipartition interaction index of a subset $A \subseteq N$ is defined as

$$\hat{I}^v_{\mathrm{Ba}}(\mathcal{A}) = \sum_{\mathcal{B} \subseteq \mathcal{N} \backslash \mathcal{A}} \frac{1}{2^{(|\mathcal{N}| - |\mathcal{A}|)}} \hat{\Delta}_{\mathcal{A}} v(\mathcal{B}).$$

Here

$$\hat{\mathbf{\Delta}}_{\mathcal{A}}\mathbf{v}(\mathcal{B}) = \mathbf{v}(\mathcal{B}\cup\mathcal{A}) - \mathbf{v}(\mathcal{B}), |\mathcal{A}| < \mathbf{2}.$$

Both indices are extensions of the Shapley and Banzhaf indices.

As with non-additivity indices we have that

Theorem 2 *Let v be a capacity on $\mathcal{N}$. If v is $\star$-additive, then $\hat{I}^v_{\mathrm{Sh}}(\mathcal{A}) \overset{\star}{=} 0$, and $\hat{I}^v_{\mathrm{Ba}}(\mathcal{A}) \overset{\star}{=} 0$ $\forall \mathcal{A} \subseteq \mathcal{N}$ and $|\mathcal{A}| \geq 2$.*

Hence the sign of the bipartition indices reflects the kind of non-additivity of the fuzzy measure.

The nonmodularity index is based on the sum of probabilistic expectations of the second derivatives $\Delta_{ij}\mu(B)$ when $B \subset A$ [23] .

Definition 28 (Nonmodularity index) *Let μ be a fuzzy measure on N. For any $A \subseteq N$, $|A| \geq 2$, the nonmodularity index is defined as*

$$\begin{aligned} d_\mu(A) &= \sum_{\substack{B\subset A, |A\setminus B|\geq 2\\ i,j\in A\\ i,j\notin B}} w(|A|,|B|)(\mu(B\cup\{i,j\}) - \mu(B\cup\{i\}) - \mu(B\cup\{j\}) + \mu(B)) \\ &= \sum_{\substack{B\cup\{i,j\}\subseteq A\\ i,j\notin B}} w(|A|,|B|)\Delta_{ij}\mu(B), \end{aligned} \tag{2.22}$$

where $w(|A|,|B|) > 0$ is the weight given by $w(|A|,|B|) = \left(\binom{|A|}{2}\binom{|A|-2}{|B|}\right)^{-1}$.

The nonmodularity index reflects the expected interaction within a respective subset, or a measure convexity of the fuzzy measure within that subset. We list some properties of the nonmodularity index established in [23].

1. If a capacity μ on N is $\star$-modular, then $d_\mu(A) \overset{\star}{=} 0$, $\forall A \subseteq N$, $|A| \geq 2$. If a capacity μ on N is $\star$-modular within $S \subseteq N$, then $d_\mu(A) \overset{\star}{=} 0$, $\forall A \subseteq S$, $|A| \geq 2$.

2. If the fuzzy measures μ, ν are dual on N, then $d_\mu(N) = -d_\nu(N)$. For a self-dual fuzzy measure $d_\mu(N) = 0$. This holds for additive (or modular) fuzzy measures as special cases.

3. The range of $d_\mu(A)$ is $[-1, 1]$ for all $A \subseteq N$, $|A| \geq 2$.

4. For a self-dual fuzzy measure $d_\mu(N) = 0$. This holds for additive (or modular) fuzzy measures as special cases. However $d_\mu(N) = 0$ does not imply modularity, because the nonmodularity index is defined as a sum of expectations.

5. The nonmodularity index can be expressed in terms of the values of the fuzzy measure, for any $A \subseteq N$, $|A| \geq 2$,

$$d_\mu(A) = \mu(A) + \mu(\emptyset) - \frac{1}{|A|} \sum_{\{i\} \subset A} [\mu(\{i\}) + \mu(A \backslash \{i\})]. \tag{2.23}$$

6. For any $A \subseteq N$, $|A| \geq 2$,

$$d_\mu(A) = \sum_{B \subseteq A, |B| \geq 2} \frac{|B|}{|A|} m_\mu(B). \tag{2.24}$$

2.5 Sugeno Integral

Similarly to the Choquet integral, Sugeno integral is also frequently used to aggregate inputs, such as preferences in multicriteria decision making. Various important classes of aggregation functions, such as medians, weighted minimum and weighted maximum are special cases of Sugeno integral.

Definition 29 (Discrete Sugeno integral) *The Sugeno integral with respect to a fuzzy measure v is given by*

$$S_v(\mathbf{x}) = \max_{i=1,\ldots,n} \min\{x_{(i)}, v(H_i)\}, \tag{2.25}$$

where $\mathbf{x}_\nearrow = (x_{(1)}, x_{(2)}, \ldots, x_{(n)})$ is a non-decreasing permutation of the input $\mathbf{x}$, and $H_i = \{(i), \ldots, (n)\}$.

Sugeno integrals can be expressed, for arbitrary fuzzy measures, by means of the Median function in the following way:

$$S_v(\mathbf{x}) = Med(x_1, \ldots, x_n, v(H_2), v(H_3), \ldots, v(H_n)).$$

Let us denote max by $\vee$ and min by $\wedge$ for compactness. We denote by $\mathbf{x} \vee \mathbf{y} = \mathbf{z}$ the componentwise maximum of $\mathbf{x}, \mathbf{y}$ (i.e., $z_i = \max(x_i, y_i)$), and by $\mathbf{x} \wedge \mathbf{y}$ their componentwise minimum.

Main properties

- Sugeno integral is a continuous idempotent aggregation function;
- An aggregation function is a Sugeno integral if and only if it is *min-homogeneous*, i.e., $S_v(x_1 \wedge r, \ldots, x_n \wedge r) = S_v(x_1, \ldots, x_n) \wedge r$ and *max-homogeneous*, i.e., $S_v(x_1 \vee r, \ldots, x_n \vee r) = S_v(x_1, \ldots, x_n) \vee r$ for all $\mathbf{x} \in [0,1]^n, r \in [0,1]$ (See [17], Th. 4.3. There are also alternative characterizations);
- Sugeno integral is *comonotone maxitive* and *comonotone minimitive*, i.e., $S_v(\mathbf{x} \vee \mathbf{y}) = S_v(\mathbf{x}) \vee S_v(\mathbf{y})$ and $S_v(\mathbf{x} \wedge \mathbf{y}) = S_v(\mathbf{x}) \wedge S_v(\mathbf{y})$ for all comonotone[4] $\mathbf{x}, \mathbf{y} \in [0,1]^n$.

Calculation

Calculation of the discrete Sugeno integral is performed using Equation (2.25) similarly to calculating the Choquet integral on p. 11. We take the vector of pairs $((x_1, 1), (x_2, 2), \ldots, (x_n, n))$, where the second component of each pair is just the index i of x_i. The second component will help keeping track of all permutations.

Calculation of $S_v(\mathbf{x})$.

```
1. Sort the components of $((x_1,1),(x_2,2),\ldots,(x_n,n))$ with respect to
   the first component of each pair in non-decreasing order. We obtain
   $((x_{(1)},i_1),(x_{(2)},i_2),\ldots,(x_{(n)},i_n))$, so that $x_{(j)} = x_{i_j}$ and $x_{(j)} \leq x_{(j+1)}$
   for all $i$.
2. Let $\mathcal{T} = \{1,\ldots,n\}$, and $S = 0$.
3. For $j = 1,\ldots,n$ do
   (a) $S := \max(S, \min(x_{(j)}, v(\mathcal{T})))$;
   (b) $\mathcal{T} := \mathcal{T} \setminus \{i_j\}$
4. Return $S$.
```

[4]See footnote 2 on p. 10.

2.6 Constructing fuzzy measures

This section outlines the problem of fitting fuzzy measures to some sort of empirical data, the observed (or sometimes desired) pairs of input-output values. In the most typical case, the data comes in pairs $(\mathbf{x}, y)$, where $\mathbf{x} \in [0,1]^n$ is the input vector and $y \in [0,1]$ is the desired output. There are several pairs, which will be denoted by a subscript k: $(\mathbf{x}_k, y_k), k = 1, \ldots, K$.

When the data comes from an experiment, it will normally contain some errors, and therefore it is pointless to interpolate the inaccurate values y_k. In this case our aim is to stay close to the desired outputs without actually matching them.

The goal is to find a fuzzy measure v, such that the function $f = C_v$ approximates y_k, $f(\mathbf{x}_k) \approx y_k$. The satisfaction of approximate equalities $f(\mathbf{x}_k) \approx y_k$ is usually translated into the following minimization problem.

$$\begin{aligned} &\text{minimize } ||\mathbf{r}|| \\ &\text{subject to } f \text{ satisfies properties } \quad \mathcal{P}_1, \mathcal{P}_2, \ldots, \end{aligned} \tag{2.26}$$

where $||\mathbf{r}||$ is the norm of the residuals, i.e., $\mathbf{r} \in R^K$ is the vector of the differences between the predicted and observed values $r_k = f(\mathbf{x}_k) - y_k$. There are many ways to choose the norm, and the most popular are the least squares norm

$$||\mathbf{r}||_2 = \left(\sum_{k=1}^{K} r_k^2\right)^{1/2},$$

the least absolute deviation norm

$$||\mathbf{r}||_1 = \sum_{k=1}^{K} |r_k|,$$

and the Chebyshev norm

$$||\mathbf{r}||_\infty = \max_{k=1,\ldots,K} |r_k|.$$

It was also suggested that for decision making problems, the actual numerical value of the output $f(\mathbf{x}_k)$ was not as important as the ranking of the outputs. For instance, if $y_k \leq y_l$, then it should be $f(\mathbf{x}_k) \leq f(\mathbf{x}_l)$. Indeed, people are not really good at assigning consistent numerical scores to their preferences, but they are good at ranking the alternatives. Thus a suitable choice of aggregation function should be consistent with the ranking of the outputs y_k rather than their numerical values. The use of the mentioned fitting criteria does not preserve the ranking of outputs, unless they are interpolated. Preservation of ranking of outputs can be done by imposing the constraints $f(\mathbf{x}_k) \leq f(\mathbf{x}_l)$ if $y_k \leq y_l$ for all pairs k, l.

In the case when f is the Choquet integral with respect to a fuzzy measure v, C_v, our goal is to identify the values of v from the data set $(\mathbf{x}_k, y_k), k = 1, \ldots, K$. Identification of the $2^n - 2$ values from the data (two are given explicitly as $v(\emptyset) = 0, v(N) = 1$) involves the least squares or least absolute deviation problems

$$\text{minimize} \sum_{k=1}^{K} \left(C_v(x_{1k}, \ldots, x_{nk}) - y_k\right)^2, \text{ or}$$

$$\text{minimize} \sum_{k=1}^{K} \left|C_v(x_{1k}, \ldots, x_{nk}) - y_k\right|,$$

subject to the conditions of monotonicity of the fuzzy measure (they translate into a number of linear constraints, see below).

We concentrate on the least absolute deviation problem, because a) it is less sensitive to outliers, and b) it can be translated into a linear programming problem, which can be solved quickly and reliably even in the case of a very large number of parameters and constraints. Note that the main difficulty in fitting fuzzy measures is the large number of unknowns, and typically a much smaller number of data [14], for instance when $n = 15$, $2^n - 2 = 32766$.

Importance and interaction indices

The interaction indices defined in Section 2.4 are all linear functions of the values of the fuzzy measure. Conditions involving these functions can be expressed as linear equations and inequalities.

One can specify given values of importance (Shapley value) and interaction indices $\phi(i)$, I_{ij} (see p. 18) by adding linear equality constraints. Of course, these values may not be specified exactly, but as intervals, say, for Shapley value we may have $a_i \leq \phi(i) \leq b_i$. In this case we obtain a pair of linear inequalities.

k-additivity

Recall that Definition 16 specifies k-additive fuzzy measures through their Möbius transform

$$\mathcal{M}(\mathcal{A}) = 0$$

for any subset $\mathcal{A}$ with more than k elements. Since Möbius transform is a linear combination of values of v, we obtain a set of linear equalities. By using interaction indices, we can express k-additivity as (see Proposition 1) $I(\mathcal{A}) = 0$ for every $\mathcal{A} \subseteq \mathcal{N}$, $|\mathcal{A}| > k$, which is again a set of linear equalities.

However, these conditions on the fuzzy measures do not reduce the complexity of the least squares or least absolute deviation problems. They only add a number of equality and inequality constraints to these problems. However, it is possible to reduce the complexity of the problem when working in Möbius representation.

As the variables we will use $m_j = m_{\mathcal{A}} = \mathcal{M}(\mathcal{A})$ such that $|\mathcal{A}| \le k$ in some appropriate indexing system, such as the one based on cardinality ordering on p. 11. This is a much reduced set of variables ($\sum_{i=1}^{k} \binom{n}{i} - 1$ compared to $2^n - 2$). Now, monotonicity of a fuzzy measure, expressed as

$$v(\mathcal{A} \cup \{i\}) - v(\mathcal{A}) \ge 0, \quad \forall \mathcal{A} | i \notin \mathcal{A}, i = 1, \dots, n,$$

converts into (2.1), and using k-additivity, into

$$\sum_{\mathcal{B} \subseteq A | i \in \mathcal{B}, |\mathcal{B}| \le k} m_{\mathcal{B}} \ge 0, \qquad \text{for all } \mathcal{A} \subseteq \mathcal{N} \text{ and all } i \in \mathcal{A}.$$

The (non-redundant) set of non-negativity constraints $v(\{i\}) \ge 0, i = 1, \dots, n$, is a special case of the previous formula when $\mathcal{A}$ is a singleton, which simply become

$$\sum_{\mathcal{B} = \{i\}} m_{\mathcal{B}} = m_{\{i\}} \ge 0, \;\; i = 1, \dots, n.$$

Finally, condition $v(\mathcal{N}) = 1$ becomes $\sum\limits_{\mathcal{B} \subseteq \mathcal{N} | |\mathcal{B}| \le k} m_{\mathcal{B}} = 1.$

Then the least absolute deviation problem is translated into a simplified optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{K} \left| \sum_{\mathcal{A} | \; |\mathcal{A}| \le k} h_{\mathcal{A}}(\mathbf{x}_j) m_{\mathcal{A}} - y_j \right|, \\ \text{s.t.} \quad & \sum_{\mathcal{B} \subseteq A | i \in \mathcal{B}, |\mathcal{B}| \le k} m_{\mathcal{B}} \ge 0, \\ & \qquad \text{for all } \mathcal{A} \subseteq \mathcal{N}, |\mathcal{A}| > 1, \text{ and all } i \in \mathcal{A}, \\ & m_{\{i\}} \ge 0, \;\; i = 1, \dots, n, \\ & \sum_{\mathcal{B} \subseteq \mathcal{N} | |\mathcal{B}| \le k} m_{\mathcal{B}} = 1, \end{aligned} \qquad (2.27)$$

where $h_{\mathcal{A}}(\mathbf{x}) = \min_{i \in \mathcal{A}} x_i$. Note that only the specified $m_{\mathcal{B}}$ are non-negative, others are unrestricted. The number of monotonicity constraints is the same for all k-additive fuzzy measures for $k = 2, \ldots, n$.

The problem (2.27) will be subsequently converted to a linear programming problem (LP) using the following technique. Let $r_j = f(x_j) - y_j$ be the $j-$ th residual. We represent it as a difference of a positive and negative parts $r_j = r_j^+ - r_j^-$, $r_j^+, r_j^- \geq 0$. The absolute value is $|r_j| = r_j^+ + r_j^-$. Now the problem (2.27) is converted into an LP problem with respect to $\mathbf{m}, \mathbf{r}^+, \mathbf{r}^-$

$$\begin{aligned} &\text{minimize} && \sum_{j=1}^{K} (r_j^+ + r_j^-), && (2.28)\\ &\text{s.t.} \sum_{\mathcal{A}|\ |\mathcal{A}| \leq k} h_{\mathcal{A}}(\mathbf{x}_j) m_{\mathcal{A}} && -(r_j^+ - r_j^-) = y_j,\ j = 1, \ldots, K \\ &&& \text{other constraints from (2.27)}, \\ &&& r_j^+, r_j^- \geq 0. \end{aligned}$$

Similar problems are set for fitting k-maxitive and k-tolerant fuzzy measures, however fitting k-maxitive fuzzy measures requires solving a mixed integer programming problem MIP, which could be expensive computationally, hence relaxation techniques can be used here for larger n.

k-interactivity

K-interactivity simplifies significantly the fitting process, because both the number of variables and monotonicity constraints are reduces, plus the constraints become much simpler. When we fix the values of k and the values of $v(\mathcal{A})$ for subsets of cardinality $k+1$ to KC we obtain the linear programming problem

$$\begin{aligned} &\text{minimize} && \sum_{j=1}^{K} r_j^+ + r_j^-, && (2.29)\\ &\text{s.t.} && r_j^+ - r_j^- - \sum_{\mathcal{A} \subset \mathcal{N}, |\mathcal{A}| \leq k} v(\mathcal{A}) g_{\mathcal{A}}(\mathbf{x}^j) = \tfrac{1-KC}{n-k-1} \sum_{i=1}^{n-k-1} x_{(i)}^j + KC x_{(n-k)} - y_j \\ &&& v(\mathcal{A}) \geq v(\mathcal{A} \setminus \{i\}), \forall i \in \mathcal{A} \text{ for all } \mathcal{A} \subset \mathcal{N}, 0 < |\mathcal{A}| \leq k, \\ &&& v(\mathcal{A}) \leq KC, |\mathcal{A}| = k. \end{aligned}$$

While the value KC is a user parameter in this model, it can also be found automatically from the data by solving a bi-level optimisation problem, where at the inner level we optimise v for a fixed $KC \in [0,1]$, and at the outer level we optimise KC:

$$\min_{KC} \min_{\mu} \sum_{j=1}^{K} r_j^+ + r_j^-,$$

where the inner problem is the same as (2.29). Both methods are implemented in this package.

In the next Chapter we spell out the exact formulation of the LP problem, in which we also take into account the optional constraints on Shapley values, interaction indices, orness value and preservation of output ordering condition.

2.7 Random sampling in the set of fuzzy measures

Sometimes it is required to randomly generate fuzzy measures with some desired properties. They can be used for simulation studies, or as steps in probabilistic optimisation algorithms (such as evolutionary algorithms), or for sensitivity analysis. The set of fuzzy measures is a complicated polytope, called the order polytope, with an extremely large number of vertices. Special methods have been designed to sample points from such polytopes [8]. Particular types of fuzzy measures may lead to more or less complicated polytopes. For example, the set of totally monotone measures (called belief measures), which is a subset of supermodular fuzzy measures, is a simplex, and generation of such fuzzy measures is very simple. On the other hand, sampling from the whole set of supermodular or submodular fuzzy measures involves many additional linear constraints and hence far more complicated polytope.

This package uses the versions of MinimalsPlus and topological sort for order polytopes, followed by a Markov chain.

Chapter 3

Computational methods

3.1 Representations of set functions

When dealing with discrete set functions, like discrete fuzzy measures, on a computer, it is important to encode the 2^n values of such functions $v_{\mathcal{A}}$ in some array. Typically one uses either binary or cardinality orderings on p.11, and both orderings are best suited for different purposes. It is convenient to use both orderings at the same time, and have a conversion mechanism, for example a lookup table.

A discrete set is conveniently encoded as a binary string, which can be represented by an unsigned integer on a computer (e.g., in C language). The i-th bit of such integer indicates the presence or absence of the i-th element in a set. Sets of up to 32 elements can be efficiently encoded on 32-bit computers into a single integer, and of course arrays of integers can be used for sets with more elements.

Bitwise operations on integers allow one to calculate easily set union, intersection, complement, determine whether a set is a subset of another one and so on. In `pyfmtools` library there are a number of routines that perform these operations.

We have also seen in the previous Chapter that the standard and Möbius representations of set functions can be used for various calculation, and sometimes one is preferred to another because of computational efficiency. Thus conversion routines (the Möbius and Zeta transforms) are essential.

3.2 Basic manipulations and tests

Fuzzy measures can be characterized by various indices, such as interaction indices, and can belong to specific classes, such as sub or super-additive. `Rfmtool` implements a number of calculation routines and tests, in particular:

1. Calculation of Shapley values;
2. Calculation of Banzhaf indices;
3. Calculation of all interaction indices;
4. Calculation of all Banzhaf interaction indices;
5. Calculation of the dual fuzzy measure;
6. Calculation of the orness value of the Choquet integral;
7. Calculation of the entropy of the Choquet integral;
8. Tests whether a fuzzy measure is:
   - Balanced;
   - Self-dual;
   - Subadditive;
   - Superadditive;
   - Additive;
   - Submodlar;
   - Supermodular;
   - Symmetric;
   - k-maxitive.

Tests are performed with a given tolerance. For numerical efficiency reasons, certain quantities (like ordering conversion tables, tables of sets cardinalities and factorials) are pre-computed for a given n, at the initialization stage. `pyfmtools` uses the formulas presented in Sections 2.3 and 2.4 using the standard and the Möbius representations interchangeably. For Sugeno fuzzy measures it also computes the value of λ (given the values of v at singletons).

`pyfmtools` also implements an efficient calculation of the Choquet and Sugeno integrals as described on p. 10 and p. 24.

3.3 Fitting fuzzy measures to data

Throughout this section n will denote the dimensionality of the space, and K will denote the size of the data set. We are given a data set representing the values of an unknown function f For example, when $n = 4$ we have

x_1	x_2	x_3	x_4	y
x_{11}	x_{12}	x_{13}	x_{14}	y_1
x_{21}	x_{22}	x_{23}	x_{24}	y_2
x_{31}	x_{32}	x_{33}	x_{34}	y_3
$\vdots$				
x_{K1}	x_{K2}	x_{K3}	x_{K4}	y_K

The goal is to identify a fuzzy measure v, such that the corresponding Choquet integral $f = C_v$ predicts the outputs y_k as close as possible in the least absolute deviation sense. This is done by solving a linear programming problem (2.28). In addition, optional conditions on the bounds of interaction indices, Shapley values or orness value are also incorporated as linear constraints.

The problem is set in Möbius representation. To obtain a standard LP formulation, equality constraints are represented by pairs of inequality constraints, and unconstrained variables (in Möbius representation only the values corresponding to singletons are non-negative) are replaced with pairs of non-negative variables (the positive and negative parts of the unconstrained variable). The decision variables are:

$$\underbrace{r_1^+, \ldots, r_K^+, r_1^-, \ldots, r_K^-}_{\text{residuals}}, \underbrace{m_1, m_2, \ldots, m_n}_{\text{singletons}}, \underbrace{m_{12}^+, \ldots, m_{12\ldots n}^+, m_{12}^-, \ldots, m_{12\ldots n}^-}_{\substack{\text{positive and negative parts of} \\ \text{other values}}}.$$

If the fuzzy measure is assumed to be k-additive, then in Möbius representation values corresponding to subsets of cardinality greater than k are 0. These decision variables (from the third group) are explicitly excluded from the problem formulation, which is the key to reducing its complexity. For instance, for 2-additive fuzzy measures we will have the decision variables

$$\underbrace{r_1^+, \ldots, r_K^+, r_1^-, \ldots, r_K^-}_{\text{residuals}}, \underbrace{m_1, m_2, \ldots, m_n}_{\text{singletons}}, \underbrace{m_{12}^+, \ldots, m_{n-1,n}^+, m_{12}^-, \ldots, m_{n-1,n}^-}_{\substack{\text{positive and negative parts of} \\ \text{other values, up to } \{n-1, n\}}}.$$

An instance of a complete problem formulation is presented in Table.3.1. For numerical efficiency purposes, a dual of this LP problem is actually solved in `pyfmtools` , because when the fuzzy measure is k-additive, the number of variables is much less than that of constraints.

Fitting k-tolerant fuzzy measures can be done by solving the following LP in the standard representation v (not Möbius).

$$
\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{K} r_j^+ + r_j^-, & (3.1)\\
\text{s.t.} & r_j^+ - r_j^- = \sum_A v(A) g_A(\mathbf{x}_j) - y_j & \text{data fitting}\\
 & v(A) \geq v(A \setminus \{i\}), \forall i \in A \text{ for all } A \subseteq N, |A| \leq k, & \text{monotonicity}\\
 & v(A) = 1, \text{ for all } A,\ |A| \geq k. & \text{k-tolerance,}
\end{array}
$$

where the functions g_A are

$$
g_A(\mathbf{x}) = \max(0, \min_{i \in A} x_i - \max_{i \in N \setminus A} x_i). \tag{3.2}
$$

Fitting k-maxitive fuzzy measure is performed by solving MIP

$$
\begin{array}{lll}
\text{minimize} & \sum_{j=1}^{K} r_j^+ + r_j^-, & (3.3)\\
\text{s.t.} & r_j^+ - r_j^- - \sum_A v(A) g_A(\mathbf{x}_j) = -y_j & \text{data fitting}\\
 & v(A) - v(A \setminus \{i\}) \geq 0, \forall i \in A \text{ for all } A \subseteq N, & \text{monotonicity}\\
 & v(A) - v(A \setminus \{i\}) - c(A,i) \leq 0, \quad \forall i \in A \text{ and } |A| > k, & \text{k-maxitivity}\\
 & \sum_{i \in A} c(A,i) \leq |A| - 1, \text{ for all } A \subseteq N, |A| > k, & \text{at least one active}\\
 & v(N) = 1, \quad c(A,i) \in \{0,1\}. & \text{constraint.}
\end{array}
$$

The binary variables $c(A,i)$ indicate for which i we have equality $v(A) = v(A \setminus \{i\})$.

We also implemented an alternative MIP relaxation heuristic for $n > 6$ or $n - k > 3$ as follows. We solve the problem in two steps. At Step 1 we solve a relaxation to (3.3) in which $c(A,i)$ were not required to be binary (the restriction was $c(A,i) \in [0,1]$). We then used the optimal solution for $c(A,i)$ to fix the k-maxitivity constraints for every A, $|A| > k$. Namely, for every fixed A we selected $i \in A$ which corresponded to the smallest value $c(A,i)$ in the optimal solution to the relaxed problem. We then fixed that variable $c(A,i) = 0$, meaning active constraint $v(A) = v(A \setminus \{i\})$. At Step 2 we solved LP (3.3) without requiring $c(A,i)$ to be binary, but fixing some of its values at 0 according to our selection above.

	residuals						singletons				
Variables	r_1^+	r_2^+	$\dots$	r_1^-	$\dots$	r_K^-	m_1	$\dots$	m_n	m_{12}^+	$\dots$
Obj. function	1	1	$\dots$	1	$\dots$	1					
	-1			1			$h_{\{1\}}$	$\dots$	$h_{\{n\}}$	$h_{\{12\}}$	$\dots$
i-th datum	1			-1			$-h_{\{1\}}$	$\dots$	$-h_{\{n\}}$	$-h_{\{12\}}$	$\dots$
		-1			1		$h_{\{1\}}$	$\dots$	$h_{\{n\}}$	$h_{\{12\}}$	$\dots$
		1			-1		$-h_{\{1\}}$	$\dots$	$-h_{\{n\}}$	$-h_{\{12\}}$	$\dots$
Data fitting			-1			1	$h_{\{1\}}$	$\dots$	$h_{\{n\}}$	$h_{\{12\}}$	$\dots$
			1			-1	$-h_{\{1\}}$	$\dots$	$-h_{\{n\}}$	$-h_{\{12\}}$	$\dots$
Boundary							1	$\dots$	1	1	$\dots$
cond.							-1	$\dots$	-1	-1	$\dots$
							1			1	
								1		1	
Monotonicity							1	1	$\dots$	1	1
							$\dots$				$\dots$
Interactions							$\dots$			$\dots$	
							$\dots$			$\dots$	
Orness							$\dots$			$\dots$	
Order							$\dots$			$\dots$	
preservation							$\dots$			$\dots$	

Table 3.1: Linear program formulation. All the constraints

Fitting k-interactive fuzzy measures

This method is computationally more efficient for larger n than fitting k-additive fuzzy measures. It is based on the equation (2.18), and the respective reduction of the number and complexity of monotonicity constraints. For fixed values of $k < n$ and $KC \in [0,1]$, the fitting problem translates into the following LP program.

$$\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{K} r_j^+ + r_j^-, && (3.4)\\
\text{s.t.} \quad & r_j^+ - r_j^- - \sum_{A\subset N, |A|\le k} v(A) g_A(\mathbf{x}^j) = \tfrac{1-KC}{n-k-1} \sum_{i=1}^{n-k-1} x_{(i)}^j + KC x_{(n-k)} - y_j \\
& v(A) \ge v(A \setminus \{i\}), \forall i \in A \text{ for all } A \subset N, 0 < |A| \le k, \\
& v(A) \le KC, |A| = k.
\end{aligned}$$

One parameter in this approach is the value of KC. There are two ways we can determine it from the data as well. The first way is to assign a specific value like $KC = k/n$. This will not guarantee the optimality with respect to the data fitting criterion but is a reasonable choice. Another way is to solve a bi-level optimisation problem, where at the inner level we optimise v for a fixed $KC \in [0,1]$, and at the outer level we optimise KC:

$$\min_{KC} \min_{\mu} \sum_{j=1}^{K} r_j^+ + r_j^-,$$

where the inner problem is the same as (3.4). Both methods are implemented in this package.

Maximal chains approach

The next approach to reduce the complexity of the fitting problem is based on the observation that not all the values of v can be determined from the data [6]. Notice that the Choquet integral is a piecewise linear monotone continuous function on $[0,1]^n$ with the idempotency property $C(x, x, \ldots, x) = x$ for all $x \in [0,1]$. There are exactly $n!$ linear segments in that function, the same as the number of permutations of the components of the vector $\mathbf{x}$. Thus at least $n!$ data points are needed in order for each linear segment to have one datum. Of course, the linear segments are not independent, and ultimately only $2^n - 2$ fuzzy measure values are to be determined. Each datum affects $n-1$ values, which further relaxed the requirements on K. It was shown that at least $n!/[(n/2)!]^2$ (for and even n) and $n!/[((n-1)/2)!((n+1)/2)!]$ (for an odd n) data are needed. However for sufficiently large n, K could be substantially smaller than those quantities.

The maximal chains approach is based on using only those values $v(\mathcal{A})$ that can be determined from the data as the variables of the fitting problem, and the rest to be calculated after the fitting process, but ensuring satisfaction of the monotonicity constraints. This way we can reduce the number of variables of the fitting problem.

Consider the inclusion relation $\subseteq$ over the set $P(\mathcal{N})$. It is a finite bounded lattice with the least element $\emptyset$ and the greatest element $\mathcal{N}$. There are $n!$ maximal chains in that lattice, the same as the number of sorting orders of the input vectors, as there is a one-to-one correspondence between the maximal chains and permutations of the inputs.

Now, each datum from the data set $\mathcal{D}$ corresponds to one or more (in case of equal components of $\mathbf{x}$) maximal chains. Let us construct the set $\mathcal{C} \subseteq P(\mathcal{N})$ of those values of v that belong to any of the maximal chains that correspond to the data set $\mathcal{D}$. Then $|\mathcal{C}| \leq |P(\mathcal{N})| = 2^n$. Then only the values $v(\mathcal{A})$, $\mathcal{A} \in \mathcal{C}$ can be directly identified from the data set.

The rest of the values of v still need to be consistent with those constructed from the data because of monotonicity constraints, but that problem will be solved separately. The rationale is that $\mathcal{C}$ is much smaller than $P(\mathcal{N})$ (for a large enough n), and also the number of monotonicity constraints on $\mathcal{C}$ is much smaller that those on $P(\mathcal{N})$.

Fitting in marginal contributions representation

The following values

$$\Delta_i v(B) = v(B \cup \{i\}) - v(B) \geq 0, \ \forall i \in N, B \subseteq N \setminus \{i\} \tag{3.5}$$

are called the marginal contributions (of the criterion i to the subset B).

The actual values of the fuzzy measure can be recovered from the following. Let $A = \{a_1, ..., a_{|A|}\} \subseteq N$, π be a permutation of $(1, 2, \ldots, |A|)$. Then

$$v(A) = \sum_{i=1}^{|A|} \Delta_{a_{\pi(i)}} v(A_{\pi(i-1)}), \ \forall \pi, \tag{3.6}$$

where $A_{\pi(i)} = \{a_{\pi(1)}, ..., a_{\pi(i)}\}$, $A_{\pi(0)} = \emptyset$.

It is possible to set up a linear programming fitting problem in marginal contributions representation, even though this representation has more variables and constraints than the standard representation. However the (sufficient) constraints that enforce sub- or super-modularity are much simpler (they involve only pairs of neighbouring marginal contributions on each maximal chain). Combined with k-interactivity, the resulting fitting problem becomes simple enough for efficient computational treatment. This method is implemented in `pyfmtools` package.

Chapter 4

Description of the library

4.1 Installation

Installation of `pyfmtools` package can be done from sources or using `pip install pyfmtools`. This package relies on the CFFI library for `Python` to C interface, which needs to be installed (`pip install cffi`) if not done automatically. `Numpy` is another useful library required by `pyfmtools` . The `pyfmtools.tar.gz` contains the necessary files and should be expanded into a suitable directory.

4.2 Description of the functions in package `pyfmtools`

Scripts in the `tests` folder test the correctness of installation by running tests for some functions included in the package.

Import the library:

```
import pyfmtools as fm
```

An important note: to pass `Python` arrays created with `numpy`, they need to be cast to C pointers as follows:

```
import numpy as np
a=np.array([0.2,0.1,0.6])
pa = ffi.cast("double *", a.ctypes.data)
```

In this manual it will be assumed that a) the `Python` arrays have been created (and sufficient memory allocated), and b) the pointers were found using the `ffi.cast` function.

`pyfmtools` allows the user to call the library functions bypassing `cffi` and `numpy` (the `Python` wrapper will convert all the arrays to the required format automatically, see `test_wrapper.py`), or without the wrapper (which avoids some overheads but allows more control over the code).

4.2.1 Using `Python` wrapper

Operations on fuzzy measures

The first operation should always be initialisation unless working with sparse representations and large n, it will be stated in the function description. The internal structures are precomputed for n variables and saved in the environment variable, which needs to be defined.

```
import pyfmtools as fm
fm.fm_init( n)
```

The function computes the internal structures for n variables. The output is passed to the subsequent operations. Example:

```
env=fm.fm_init(3)
```

`fm.fm_free( env)`

The function frees the memory allocated by the initialisation.

The following functions involve the parameters v (the array containing the fuzzy measure in standard representation) or Mob (in Möbius representation), env - the environment variable set to n variables. The values of the fuzzy measure almost always obey the binary ordering. it will be stated otherwise in case cardinailty ordering is used.

`fm.Choquet(x, v, env)`

The function computes and returns the value of the Choquet integral of x, wrt fuzzy measure v (eq.(2.3)). The environment variable env is precomputed in the `fm.fm_init` function. $x \in [0,1]^n$, $v \in [0,1]^m$. Example:

```
x = [0.2,0.1,0.6]
fm.fm_init( 3)
v = [0,0.3,0.5,0.6,0.4,0.8,0.7,1]
r= fm.Choquet(x,v,env)
```

`fm.ChoquetMob(x, Mob, env)`

The function computes and returns the value of the Choquet integral of x, wrt fuzzy measure given in Möbius representation Mob (eq.(2.6)). Example:

```
x=[0.2,0.5,0.4]
Mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
r = fm.ChoquetMob(x,Mob,env)
```

`fm.ChoquetKinter(x, v, kint, env)`

Calculates the value of the Choquet integral of x, wrt $k$interactive fuzzy measure v using compact representation of v (in cardinality ordering). Example:

```
kint=2
x=[0.2,0.5,0.4]
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
r=fm.ChoquetKinter(x, v, kint, env)
```

`fm.Sugeno(x, v, env)`

Calculates the value of the Sugeno integral of x, wrt fuzzy measure v (eq.(2.25)). Example:

```
x=[0.2,0.5,0.4]
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
r = fm.Sugeno(x,v,env)
```

`fm.Orness(v, env)`
Calculates the orness value of the Choquet integral wrt fuzzy measure v in general representation. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
r = fm.Orness(v,env)
```

`fm.Entropy(v, env)`
Calculates the entropy value of the Choquet integral wrt fuzzy measure v in standard representation. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
r= fm.Entropy(v,env)
```

`fm.Mobius(v, env)`
Calculates the Möbius representation of v. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
mob=fm.Mobius(v,env)
```

`fm.Zeta(mob, env)`
Calculates the inverse Möbius representation of a fuzzy measure, i.e. the standard representation. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
v= fm.Zeta(mob,env)
```

`fm.Shapley(v, env)`
Calculates the Shapley values of v in standard representation and returns it as an array of size n. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
s= fm.Shapley(v,env)
```

`fm.ShapleyMob(Mob, env)`
Calculates the Shapley values of Mob in Möbius representation and returns it as an array of size n. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
s= fm.ShapleyMob(mob,env)
```

`fm.Banzhaf(v, env)`
Calculates the Banzhaf values of v in standard representation and returns it as an array of size n. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
b= fm.Banzhaf(v,env)
```

`fm.BanzhafMob(mob, env)`
Calculates the Banzhaf values of Mob in Mobius representation and returns it as an array of size n. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
b= fm.BanzhafMob(mob, env)
```

`fm.Interaction(v, env)`
Calculates all the interaction indices of fuzzy measure v in standard representation and returns it in an array of size m. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
s=fm.Interaction(v,env)
```

`fm.InteractionMob(Mob, env)`
Calculates all the interaction indices of fuzzy measure Mob in Möbius representation and returns it in an array of size m. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
s=fm.InteractionMob(mob, env)
```

`fm.InteractionB(v, env)`
Calculates all the Banzhaf interaction indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
b= fm.InteractionB(v, env)
```

`fm.InteractionBMob(Mob, env)`
Calculates all the Banzhaf interaction indices of fuzzy measure Mob in Möbius representation and returns the result in an array of size m. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
b=fm.InteractionBMob(mob, env)
```

`fm.BipartitionShapleyIndex(v, env)`
Calculates all the Shapley bipartition indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
s=fm.BipartitionShapleyIndex(v,env)
```

`fm.BipartitionBanzhafIndex(v, env)`
Calculates all the Banzhaf bipartition indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
b= fm.BipartitionBanzhafIndex(v,env)
```

`fm.NonadditivityIndex(v, env)`
Calculates all the nonadditivity indices of fuzzy measure v given in standard representation and returns the result in an array of size $m = 2^n$. Examaple:

```
v=n[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
na= fm.NonadditivityIndex(v,env)
```

`fm.BNonadditivityIndexMob(Mob, env)`
Calculates all the nonadditivity indices of fuzzy measure v given in Möbius representation and returns the result in an array of size $m = 2^n$. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
na= fm.py_BNonadditivityIndexMob(mob,env)
```

`fm.NonmodularityIndex(pv, env)`

Calculates all the $m = 2^n$ nonmodularity indices of fuzzy measure v given in standard representation and returns the result in an array of size m.

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
nm=fm.NonmodularityIndex(v, env)
```

`fm.NonmodularityIndexMob(pmob, env)`

Calculates all the nonmodularity indices of fuzzy measure Mob given in Möbius representation and returns the result in an array of size m. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],
nm=fm.NonmodularityIndexMob(mob, env)
```

`fm.NonmodularityIndexMobkadditive(pmob, k, env)`

Calculates all the $m = 2^n$ nonmodularity indices of k-additive fuzzy measure Mob given in Möbius representation (in cardinality ordering) and returns the result in an array of size m. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
k=2
nm=fm.NonmodularityIndexMobkadditive(mob, k, env)
```

`fm.NonmodularityIndexKinteractive(pv, kint, env)`

Calculates all the $m = 2^n$ nonmodularity indices of k-interactive fuzzy measure v given in standard representation (in cardinality ordering) and returns the result in an array of size m.

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
kint=2
nm=fm.NonmodularityIndexKinteractive(v, kint, env))
```

`fm.dualm(v, env)`

Calculates the dual of fuzzy measure v in standard representation and returns it in an array of size m. Example:

```
v=[0,0.3,0.5,0.6,0.4,0.8,0.7,1]
d= fm.dualm(v,env)
```

`fm.dualmMob(Mob, env)`

Calculates the dual of fuzzy measure M in Mobius representation and returns it in an array of size m. Example as above but with Mobius input. Example:

```
mob=[0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
d=fm.dualmMob(mob, env)
```

`fm.ConstructLambdaMeasure(singletons, env)`

Given the values of a fuzzy measure at singletons, finds the value of λ, computes all other values of the fuzzy measure in standard representation, and returns the pair (λ, v), where v is an array of size m. The input array *singletons* is an array of size n of the values of fuzzy measure at singletons. Example:

```
singletons=[0, 0.3, 0.5]
lambda, v=fm.ConstructLambdaMeasure(singletons, env)
```

`fm.ConstructLambdaMeasureMob((singletons, env))`

Given the values of a fuzzy measure at singletons, finds the value of λ, computes all other values of the fuzzy measure in Möbius representation, and returns the pair (λ, Mob), where Mob is an array of size m. The input array *singletons* is an array of size n of the values of fuzzy measure at singletons. Example:

```
singletons=[0, 0.3, 0.5]
lambda,mob=fm.ConstructLambdaMeasureMob(singletons, env)
```

`fm.export_maximal_chains(n, v, mc, env)`

Returns in mc the arrays of maximal chains (there are $n!$ such arrays) of a fuzzy measure v, each array is of length n and contains the chains of discrete derivatives, which can be used to calculate the Choquet integrals for all possible orderings of the argument x, as well as serve as coefficients of the piecewise linear objective functions. That is, each maximal chain corresponds to the coefficients of a linear function on the respective simplex. All such simplices form a partition of the unit cube. Example:

```
n=3
vb=[0, 0.00224734, 0.06493396, 0.51092037, 0.00965497, 0.37405545, 0.15455057,
1]
mc=fm.export_maximal_chains(n,vb,env)
```

`fm.ConvertCard2Bit( src, env)`

Converts fuzzy measure *src* from cardinality to binary ordering. Example:

```
src=[0, 0.0340916, 0.00890771, 0.01109779, 0.07918582, 0.6305862, 0.1846705,
1]
v=fm.ConvertCard2Bit(src, env)
```

`fm.min_subset(x, n, S)`

`fm.py_max_subset(x, n, S)`

Return min or max of x_i, $i \in S$. S should be written in binary form. Example:

```
x=[0,0.3,0.5,0.6,0.7,0.8,0.85,0.9]
fm.min_subset(x,3,6)
fm.max_subset(x,3,5)
```

`fm.min_subsetC(x, n, S, env)`

Calculates the minimum of x_i, $i \in S$, in cardinality based ordering. Example:

```
x=[0,0.3,0.5]
fm.min_subsetC(x,3,3,env)
```

`fm.max_subsetNegC(x, n, S, env)`

Calculates the maximum over the complement of subset S in cardinality ordering. Example:

```
x=[0,0.3,0.5]
fm.max_subsetNegC(x,3,3,env)
```

`fm.SizeArraykinteractive(n, k, env)`

Returns the size of the array to store k-interactive fuzzy measure. Example:
`fm.SizeArraykinteractive(3,1,env)`

`fm.IsSubsetC(i, j, env)`
Returns true if j is subset of i in cardinality ordering.Example:
`fm.IsSubsetC(2,1,env)`

`fm.IsElementC(A, j, env)`
Returns true if j is an element of A in cardinality ordering. Example:
`fm.IsElementC(1,2,env)`

`fm.ExpandKinteractive2Bit( src, env, kint, arraysize)`
`fm.ExpandKinteractive2Bit_m( src, env, kint, arraysize, VVC)`
Expands k interactive fuzzy measure (from *src* to *dest*) given in more compact representation of size *arraysize* into its full representaiton in binary ordering. The function `ExpandKinteractive2Bit` allocates working memory internally, whereas the second function expects working array VVC of length m. Example:

```
src=[0.0340916, 0.01109779, 0.07918582,0.7,1]
kint=1
arraysize=fm.SizeArraykinteractive(n, kint, env)
v=fm.ExpandKinteractive2Bit(src,env,kint,arraysize)
dest,vcc=fm.ExpandKinteractive2Bit_m(src,env,kint,arraysize)
```

`fm.fm_arraysize( n, kint, env)`
Returns the length of the array of values of k-interactive fuzzy measures. Useful for reserving memory to store them. For $kint > 5$ requires initialisation of the *env* variable, otherwise no. Example:
`len=fm.fm_arraysize(n, kint, env)`

`fm.ShowCoalitions(env)`
`fm.ShowCoalitionsCard(env)`
Return the decimal expression for the subsets A, e.g. for $A = 11$ it will show 124 for printing. In binary and in cardinality ordering respectively. A is the array of integers to receive the decimal expressions for all 2^n coalitions. Example:
`fm.ShowCoalitions( env)`

The following routines are self-explanatory. They return 1 if yes, 0 if no. The input parameters are: fuzzy measure v in standard representation and fuzzy measure Mob in Möbius representation (both in binary ordering).

```
fm.IsMeasureAdditive(v, env)
fm.IsMeasureAdditiveMob(Mob, env)
fm.IsMeasureBalanced(v, env)
fm.IsMeasureBalancedMob(Mob, env)
fm.IsMeasureSelfdual(v, env)
fm.IsMeasureSelfdualMob(Mob, env)
fm.IsMeasureSubadditive(v, env)
```

`fm.IsMeasureSubadditiveMob(v, env)`
`fm.IsMeasureSubmodular(v, env)`
`fm.IsMeasureSubmodularMob(Mob, env)`
`fm.IsMeasureSuperadditive(v, env)`
`fm.IsMeasureSuperadditiveMob(Mob, env)`
`fm.IsMeasureSupermodular(v, env)`
`fm.IsMeasureSupermodularMob(Mob, env)`
`fm.IsMeasureSymmetric(v, env)`
`fm.IsMeasureSymmetricMob(Mob, env)`
`fm.IsMeasureKMaxitive(v, env)` (returns k);
`fm.IsMeasureKMaxitiveMob(Mob, env)` (returns k).

4.2.2 Without `Python` wrapper, using `numpy` and `cffi`

Operations on fuzzy measures

The first operation should always be initialisation unless working with sparse representations and large n, it will be stated in the function description. The internal structures are precomputed for n variables and saved in the environment variable, which needs to be defined.

`fm.py_fm_init( n, env)`
The function computes the internal structures for n variables. The output is passed to the subsequent operations. Example:

```
env=ffi.new("struct fm_env *")
fm.py_fm_init(3, env)
```

`fm.py_fm_free( env)`
The function frees the memory allocated by the initialisation.

The following functions involve the parameters v (the array containing the fuzzy measure in standard representation) or Mob (in Möbius representation), *env* - the environment variable set to n variables. The values of the fuzzy measure almost always obey the binary ordering. it will be stated otherwise in case cardinailty ordering is used.

`fm.py_Choquet(x, v, env)`
The function computes and returns the value of the Choquet integral of x, wrt fuzzy measure v (eq.(2.3)). The environment variable *env* is precomputed in the `fm.py_fm_init` function. $x \in [0,1]^n$, $v \in [0,1]^m$. Example:

```
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r= fm.py_Choquet(px,pv,env)
```

`fm.py_ChoquetMob(x, Mob, env)`
The function computes and returns the value of the Choquet integral of x, wrt fuzzy measure given in Möbius representation Mob (eq.(2.6)). Example:

```
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
Mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pMob = ffi.cast("double *", Mob.ctypes.data)
r = fm.py_ChoquetMob(px,pMob,env)
```

`fm.py_ChoquetKinter(x, v, kint, env)`

Calculates the value of the Choquet integral of x, wrt $k$interactive fuzzy measure v using compact representation of v (in cardinality ordering). Example:

```
kint=2
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r=lib.py_ChoquetKinter(px, pv, kint, env)
```

`fm.py_Sugeno(x, v, env)`

Calculates the value of the Sugeno integral of x, wrt fuzzy measure v (eq.(2.25)). Example:

```
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r = fm.py_Sugeno(px,pv,env)
```

`fm.py_Orness(v, env)`

Calculates the orness value of the Choquet integral wrt fuzzy measure v in general representation. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r = fm.py_Orness(pv,env)
```

`fm.py_Entropy(v, env)`

Calculates the entropy value of the Choquet integral wrt fuzzy measure v in standard representation. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r= fm.py_Entropy(pv,env)
```

`fm.py_Mobius(v, mob, env)`

Calculates the Möbius representation of v. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
mob=np.zeros(env.m, np.float) # note env.m=2**n
pmob = ffi.cast("double *", mob.ctypes.data)
fm.py_Mobius(pv,pmob,env)
```

`fm.py_Zeta(pmob, pv, env)`

Calculates the inverse Möbius representation of a fuzzy measure, i.e. the standard representation. Example:

```
v=np.zeros(env.m, np.float)
pv = ffi.cast("double *", v.ctypes.data)
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
fm.py_Zeta(pmob,pv,env)
```

`fm.py_Shapley(v, s, env)`

Calculates the Shapley values of v in standard representation and returns it as an array of size n. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
s=np.zeros(env.n,np.float)
ps = ffi.cast("double *", s.ctypes.data)
fm.py_Shapley(pv,ps,env)
```

`fm.py_ShapleyMob(Mob, S, env)`

Calculates the Shapley values of Mob in Möbius representation and returns it as an array of size n. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
s=np.zeros(env.n, np.float)
ps = ffi.cast("double *", s.ctypes.data)
fm.py_ShapleyMob(pmob,ps,env)
```

`fm.py_Banzhaf(pv, B, env)`

Calculates the Banzhaf values of v in standard representation and returns it as an array of size n. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
B=np.zeros(env.n, np.float)
pB = ffi.cast("double *", B.ctypes.data)
fm.py_Banzhaf(pv,pB,env)
```

`fm.py_BanzhafMob(pmob, B, env)`

Calculates the Banzhaf values of Mob in Mobius representation and returns it as an array of size n. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
B=np.zeros(env.n, np.float)
pB = ffi.cast("double *", B.ctypes.data)
fm.py_BanzhafMob(pmob, pB, env)
```

`fm.py_Interaction(v, s, env)`

Calculates all the interaction indices of fuzzy measure v in standard representation and returns it in an array of size m. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
s=np.zeros(env.m, np.float) # size of the array is 2**n
ps = ffi.cast("double *", s.ctypes.data)
fm.py_Interaction(pv,ps,env)
```

`fm.py_InteractionMob(Mob, s, env)`

Calculates all the interaction indices of fuzzy measure Mob in Möbius representation and returns it in an array of size m. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
s=np.zeros(env.m, np.float)
ps = ffi.cast("double *", s.ctypes.data)
fm.py_InteractionMob(pmob, ps, env)
```

`fm.py_InteractionB(v, b, env)`

Calculates all the Banzhaf interaction indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
b=np.zeros(env.m, np.float)
pb = ffi.cast("double *", b.ctypes.data)
fm.py_InteractionB(pv, pb, env)
```

`fm.py_InteractionBMob(Mob, b, env)`

Calculates all the Banzhaf interaction indices of fuzzy measure Mob in Möbius representation and returns the result in an array of size m. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
b=np.zeros(env.m, np.float)
pb = ffi.cast("double *", b.ctypes.data)
fm.py_InteractionBMob(pmob, pb, env)
```

`fm.py_BipartitionShapleyIndex(v, s, env)`

Calculates all the Shapley bipartition indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
s=np.zeros(env.m, np.float)
ps = ffi.cast("double *", s.ctypes.data)
fm.py_BipartitionShapleyIndex(pv,ps,env)
```

`fm.py_BipartitionBanzhafIndex(v, b, env)`

Calculates all the Banzhaf bipartition indices of fuzzy measure v given in standard representation and returns the result in an array of size m. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
```

```
b=np.zeros(env.m, np.float)
pb = ffi.cast("double *", b.ctypes.data)
fm.py_BipartitionBanzhafIndex(pv,pb,env)
```

`fm.py_NonadditivityIndex(v, na, env)`

Calculates all the nonadditivity indices of fuzzy measure v given in standard representation and returns the result in an array of size $m = 2^n$. Examaple:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1], np.float)
pv = ffi.cast("double *", v.ctypes.data)
na=np.zeros(env.m, np.float)
pna = ffi.cast("double *", na.ctypes.data)
fm.py_NonadditivityIndex(pv,pna,env)
```

`fm.py_BNonadditivityIndexMob(Mob, na, env)`

Calculates all the nonadditivity indices of fuzzy measure v given in Möbius representation and returns the result in an array of size $m = 2^n$. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
na=np.zeros(env.m, np.float)
pna = ffi.cast("double *", na.ctypes.data)
fm.py_BNonadditivityIndexMob(pmob,pna,env)
```

`fm.py_NonmodularityIndex(pv, pnm, env)`

Calculates all the $m = 2^n$ nonmodularity indices of fuzzy measure v given in standard representation and returns the result in an array of size m.

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
nm=np.zeros(env.m, np.float)
pnm = ffi.cast("double *", nm.ctypes.data)
fm.py_NonmodularityIndex(pv, pnm, env)
```

`fm.py_NonmodularityIndexMob(pmob, pnm, env)`

Calculates all the nonmodularity indices of fuzzy measure Mob given in Möbius representation and returns the result in an array of size m. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1], np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
nm=np.zeros(env.m, np.float)
pnm = ffi.cast("double *", nm.ctypes.data)
fm.py_NonmodularityIndexMob(pmob, pnm, env)
```

`fm.py_NonmodularityIndexMobkadditive(pmob, pnm, k, env)`

Calculates all the $m = 2^n$ nonmodularity indices of k-additive fuzzy measure Mob given in Möbius representation (in cardinality ordering) and returns the result in an array of size m. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
nm=np.zeros(env.m, np.float)
```

```
pnm = ffi.cast("double *", nm.ctypes.data)
k=2
fm.py_NonmodularityIndexMobkadditive(pmob, pnm, k, env)
```

`fm.py_NonmodularityIndexKinteractive(pv, pnm, kint, env)`

Calculates all the $m = 2^n$ nonmodularity indices of k-interactive fuzzy measure v given in standard representation (in cardinality ordering) and returns the result in an array of size m.

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
nm=np.zeros(env.m, np.float)
pnm = ffi.cast("double *", nm.ctypes.data)
kint=2
fm.py_NonmodularityIndexKinteractive(pv, pnm,kint, env))
```

`fm.py_dualm(v, d, env)`

Calculates the dual of fuzzy measure v in standard representation and returns it in an array of size m. Example:

```
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
d=np.zeros(env.m, np.float) # note env.m=2**n
pd = ffi.cast("double *", d.ctypes.data)
fm.py_dualm(pv,pd,env)
```

`fm.py_dualmMob(Mob, d, env)`

Calculates the dual of fuzzy measure M in Mobius representation and returns it in an array of size m. Example as above but with Mobius input. Example:

```
mob=np.array([0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
d=np.zeros(env.m, np.float) # note env.m=2**n
pd = ffi.cast("double *", d.ctypes.data)
fm.py_dualmMob(pmob, pd, env)
```

`fm.py_ConstructLambdaMeasure(singletons, lambda, v, env)`

Given the values of a fuzzy measure at singletons, finds the value of λ, computes all other values of the fuzzy measure in standard representation, and returns the pair (λ, v), where v is an array of size m. The input array *singletons* is an array of size n of the values of fuzzy measure at singletons. Example:

```
lambd=np.array([0.0],np.float)
plambd=ffi.cast("double *", lambd.ctypes.data)
singletons=np.array(singletons=np.array([0, 0.3, 0.5],np.float)
psingletons = ffi.cast("double *", singletons.ctypes.data)
v=np.zeros(env.m, np.float) # note env.m=2**n
pv = ffi.cast("double *", v.ctypes.data)
fm.py_ConstructLambdaMeasure(psingletons, plambd, pv, env)
```

`fm.py_ConstructLambdaMeasureMob((singletons, lambda, Mob, env))`

Given the values of a fuzzy measure at singletons, finds the value of λ, computes all other values of the fuzzy measure in Möbius representation, and returns the pair (λ, Mob), where Mob is an array of size m. The input array *singletons* is an array of size n of the values of fuzzy measure at singletons. Example:

```
lambd=np.array([0.0],np.float)
plambd=ffi.cast("double *", lambd.ctypes.data)
singletons=np.array([0, 0.3, 0.5],np.float)
psingletons = ffi.cast("double *", singletons.ctypes.data)
Mob=np.zeros(env.m, np.float) # note env.m=2**n
pMob = ffi.cast("double *", Mob.ctypes.data)
fm.py_ConstructLambdaMeasureMob(psingletons, plambd, pMob, env)
```

`fm.py_export_maximal_chains(n, v, mc, env)`

Returns in *mc* the arrays of maximal chains (there are $n!$ such arrays) of a fuzzy measure v, each array is of length n and contains the chains of discrete derivatives, which can be used to calculate the Choquet integrals for all possible orderings of the argument x, as well as serve as coefficients of the piecewise linear objective functions. That is, each maximal chain corresponds to the coefficients of a linear function on the respective simplex. All such simplices form a partition of the unit cube. Example:

```
n=3
mc=np.zeros(math.factorial(n)*n,np.float)
pmc = ffi.cast("double *", mc.ctypes.data)
vb=np.array([0, 0.00224734, 0.06493396, 0.51092037, 0.00965497, 0.37405545,
0.15455057, 1],np.float)
pvb = ffi.cast("double *", vb.ctypes.data)
fm.py_export_maximal_chains(n,pvb,pmc,env)
```

`fm.py_ConvertCard2Bit(dest, src, env)`

Converts fuzzy measure *src* from cardinality to binary ordering. Example:

```
src=np.array([0, 0.0340916, 0.00890771, 0.01109779, 0.07918582, 0.6305862,
0.1846705, 1],np.float)
psrc = ffi.cast("double *", src.ctypes.data)
dest=np.zeros(env.m,np.float)
pdest = ffi.cast("double *", dest.ctypes.data)
fm.py_ConvertCard2Bit(pdest,psrc, env)
```

`fm.py_min_subset(x, n, S)`

`fm.py_max_subset(x, n, S)`

Return min or max of x_i, $i \in S$. S should be written 5 in binary form. Example:

```
x=np.array([0,0.3,0.5,0.6,0.7,0.8,0.85,0.9],np.float)
px = ffi.cast("double *", x.ctypes.data)
fm.py_min_subset(px,3,6)
fm.py_max_subset(px,3,5)
```

`fm.py_min_subsetC(x, n, S, env)`

Calculates the minimum of x_i, $i \in S$, in cardinality based ordering. Example:

```
x=np.array([0,0.3,0.5],np.float)
px = ffi.cast("double *", x.ctypes.data)
fm.py_min_subsetC(px,3,3,env)
```

`fm.py_max_subsetNegC(x, n, S, env)`
: Calculates the maximum over the complement of subset S in cardinality ordering. Example:

```
x=np.array([0,0.3,0.5],np.float)
px = ffi.cast("double *", x.ctypes.data)
fm.py_max_subsetNegC(px,3,3,env)
```

`fm.py_SizeArraykinteractive(n, k, env)`
: Returns the size of the array to store k-interactive fuzzy measure. Example:

```
fm.py_SizeArraykinteractive(3,1,env)
```

`fm.py_IsSubsetC(i, j, env)`
: Returns true if j is subset of i in cardinality ordering.Example:

```
fm.py_IsSubsetC(2,1,env)
```

`fm.py_IsElementC(A, j, env)`
: Returns true if j is an element of A in cardinality ordering. Example:

```
fm.py_IsElementC(1,2,env)
```

`fm.py_ExpandKinteractive2Bit(dest, src, env, kint, arraysize)`
`fm.py_ExpandKinteractive2Bit_m(dest, src, env, kint, arraysize, VVC)`
: Expands k interactive fuzzy measure (from *src* to *dest*) given in more compact representation of size *arraysize* into its full representaiton in binary ordering. The function `ExpandKinteractive2Bit` allocates working memory internally, whereas the second function expects working array VVC of length m. Example:

```
src=np.array([0.0340916, 0.01109779, 0.07918582,0.7,1],np.float)
psrc = ffi.cast("double *", src.ctypes.data)
dest=np.zeros(env.m, np.float)
pdest = ffi.cast("double *", dest.ctypes.data)
vcc=np.zeros(env.m, np.float)
pvcc = ffi.cast("double *", vcc.ctypes.data)
kint=1
arraysize=fm.py_SizeArraykinteractive(n, kint, env)
fm.py_ExpandKinteractive2Bit(pdest,psrc,env,kint,arraysize)
fm.py_ExpandKinteractive2Bit_m(pdest,psrc,env,kint,arraysize,pvcc)
```

`fm.py_fm_arraysize( n, kint, env)`
: Returns the length of the array of values of k-interactive fuzzy measures. Useful for reserving memory to store them. For $kint > 5$ requires initialisation of the *env* variable, otherwise no. Example:

```
len=fm.py_fm_arraysize(n, kint, env)
v=np.zeros( len, float)
```

`fm.py_ShowCoalitions(A, env)`

```
fm.py_ShowCoalitionsCard(A, env)
```

Return the decimal expression for the subsets A, e.g. for $A = 11$ it will show 124 for printing. In binary and in cardinality ordering respectively. A is the array of integers to receive the decimal expressions for all 2^n coalitions. Example:

```
A=np.zeros( 2**n, np.intc)
pA = ffi.cast("int *", A.ctypes.data)
fm.py_ShowCoalitions(pA, env)
```

The following routines are self-explanatory. They return 1 if yes, 0 if no. The input parameters are: fuzzy measure v in standard representation and fuzzy measure Mob in Möbius representation (both in binary ordering).

`fm.py_IsMeasureAdditive(v, env)`
`fm.py_IsMeasureAdditiveMob(Mob, env)`
`fm.py_IsMeasureBalanced(v, env)`
`fm.py_IsMeasureBalancedMob(Mob, env)`
`fm.py_IsMeasureSelfdual(v, env)`
`fm.py_IsMeasureSelfdualMob(Mob, env)`
`fm.py_IsMeasureSubadditive(v, env)`
`fm.py_IsMeasureSubadditiveMob(v, env)`
`fm.py_IsMeasureSubmodular(v, env)`
`fm.py_IsMeasureSubmodularMob(Mob, env)`
`fm.py_IsMeasureSuperadditive(v, env)`
`fm.py_IsMeasureSuperadditiveMob(Mob, env)`
`fm.py_IsMeasureSupermodular(v, env)`
`fm.py_IsMeasureSupermodularMob(Mob, env)`
`fm.py_IsMeasureSymmetric(v, env)`
`fm.py_IsMeasureSymmetricMob(Mob, env)`
`fm.py_IsMeasureKMaxitive(v, env)` (returns k);
`fm.py_IsMeasureKMaxitiveMob(Mob, env)` (returns k).

4.3 Fuzzy measures in compact and sparse representations

These routines do not require initialisation of the fuzzy measure with `fm.py_fm_init` and can hence be used with large n (limited by the available memory). Note that 2-additive fuzzy measures require $O(n^2)$ parameters. They are stored in cardinality ordering.

Sparse structure definition

Sparse representation is in the form of singletons, pairs and tuples with nonzero values, stored and indexed in the respective arrays,which are part of the structure `fm_env_sparse`.

The following routines require defining a structure *envsp* of type `fm_env_sparse` to store the relevant values, but due to sparse representation only the indicated k-tuples are stored.

4.3.1 Using `Python` wrapper

`envsp=fm.prepare_fm_sparse(n, tupsize)`
This function initialises this structure, if the number of tuples is known, otherwise can be initialised with 0 tuples (i.e. set tup=0). Given the list of cardinalities of the nonzero tuples (cardinality, tuple composition) like this: 2 pairs 4-tuple and a triple: (2,1,2, 2, 3,1, 4, 1,2,3,4, 3,3,2,1...) It is used to allocate storage and later populate these values.

```
n=3
tup=0
envsp= fm.prepare_fm_sparse(n, tup)
```

`fm.free_fm_sparse(envsp)`
Frees the memory previously allocated in *envsp*.

`fm.tuple_cardinality_sparse(i, envsp)`
Returns the cardinality of the tuple numbered i in the list of tuples.

`fm.get_num_tuples(envsp)`
Returns the number of tuples.

`fm.get_sizearray_tuples(envsp)`
Returns the length of the array of tuples.

`fm.is_inset_sparse(A, card, i, envsp)`
Checks if element i (1-based) belongs to the tuple indexed A (whose cardinality can be 1,2, other (automatically determined))

`fm.is_subset_sparse(A, cardA, B, cardB, envsp)`
Checks if tuple B is a subset of tuple A, The cardinalities of both tuples need to be supplied.

`fm.min_subset_sparse(x, n, S, cardS, envsp)`
`fm.max_subset_sparse(x, n, S, cardS, envsp)`
calculate minimum (maximum) of x_i with the indices belonging to tuple indexed as S (its cardinality cardS can be 1,2, other (put 3, will be determined automatically)). Note that x is 0-based, tuples are 1-based. Example.

```
x=[0.1,0.05,0.2]
fm.min_subset_sparse(x, 3, 0, 3, envsp)
fm.max_subset_sparse(x, 3, 0, 3, envsp)
```

`fm.ChoquetMob_sparse(x, envsp)`
Calculates the Choquet integral in Mobius sparse representation. Example.

```
x=[0.1,0.05,0.2]
fm.ChoquetMob_sparse(x, envsp)
```

`fm.ShapleyMob_sparse(n, envsp)`
`fm.BanzhafMob_sparse( n, envsp)`

Calculate Shapley and Banzhaf values vectors (of size n) of a sparse fuzzy measure, returned in v. Example.

```
n=3
s=fm.ShapleyMob_sparse( n, envsp)
b=fm.BanzhafMob_sparse( n, envsp)
```

`fm.Nonmodularityindex_sparse(w, n, envsp)`

Calculates all 2^n nonmodularity indices (returned in w) using Mobius transform of a fuzzy measure (of length $2^n = m$), using sparse representation. Example.

```
n=3
nm=fm.Nonmodularityindex_sparse( n, envsp)
```

`fm.populate_fm_2add_sparse(singletons, numpairs, pairs, indicesp1, indicesp2, envsp)`

Populates 2-additive sparse capacity with nonzero values using the singletons and two arrays of indices (of size *numpairs*). Indices need to be 1-based. Example.

```
singletons=[0.1,0.2,0.3]
numpairs=3
pairs = [0.4,0.5,0.6]
indicesp1 = [1,1,2]
indicesp2 = [2,3,3]

fm.populate_fm_2add_sparse(singletons, numpairs, pairs, indicesp1, indicesp2,envsp)
```

`fm.add_pair_sparse(i, j, v, envsp)`

For populating capacities. Add pair (v_{ij}) to the structure. Indices are 1-based. Example.

```
fm.add_pair_sparse(1, 2, 0.4, envsp)
fm.add_pair_sparse(1, 3, 0.5, envsp)
fm.add_pair_sparse(2, 3, 0.6, envsp)
```

`fm.add_tuple_sparse(tupsize, tuple, v, envsp)`

For populating capacities, adds a tuple of size *tupsize* whose 1-based indices are in *tuple* Example:

```
tup=[1,3,4]
v=0.2
fm.add_tuple_sparse(3, tup, v, envsp)
fm.add_pair_sparse(2,5, v, envsp)
v=[0.1,0.2,0.3]
fm.add_singletons_sparse(v,envsp)
```

`fm.populate_fm_2add_sparse_from2add(n, v, envsp)`

Given 2-additive capacity (singletons+pairs in one array v) , selects nonzero pairs and populates sparse capacity *envsp*. Example:

```
n=3
y,v,length=fm.py_generate_fm_2additive_convex_withsomeindependent(1, n)
```

```
envsp=fm.prepare_fm_sparse(n, 0 )
fm.py_populate_fm_2add_sparse_from2add(n, v, envsp)
```

`fm.expand_2add_full( envsp)`

From sparse to full representation of 2-additive capacity (singletons and pairs, augmented with 0s). Vector v has to be allocated, its size is $n + n(n-1)/2$. Example:

```
v=fm.py_expand_2add_full(envsp)
```

`fm.expand_sparse_full(envsp)`

Exports from sparse to full capacity (vector v, size 2^n has to be preallocated). Example:

```
v=fm.expand_sparse_full(envsp)
```

`fm.sparse_get_singletons(n, envsp)`
`fm.sparse_get_pairs(pairs, envsp)`
`fm.sparse_get_tuples(tuples, envsp)`

Export the internal arrays of the sparse capacity as arrays of singletons, pairs and tuples. Return the numbers of pairs and tuples. Example:

```
sin=fm.sparse_get_singletons(3, envsp)

pairs, v = fm.sparse_get_pairs(envsp)

tuples, v = fm.py_sparse_get_tuples(envsp)
```

The following functions do not use sparse representation but they do not require initialisation by `fm_init` either, they work with 2-additive fuzzy measures stored on an array of Möbius values of singletons and pairs.

`fm.dualMobKadd(n, length, k, src, env)`

Calculate the dual of a k-additive fuzzy measures for n inputs. Generally requires initialisation but not for 2-additive fuzzy measures. Parameters: *length* is the size of the array in cardinality based ordering (can be calculated by function `py_fm_arraysize`) , k as in k-additivity, *src* is the source measure, *dest* is its dual, *env* must be initialised for $k > 2$ only. Example:

```
k=2
n=20
Mob = [0.0,0.3,0.5,-0.2,0.4,0.1,-0.2,0.1]
dual,length= fm.dualMobKadd(k, v, env)
```

`fm.Shapley2addMob(v, n)`
`fm.Banzhaf2addMob(v, n)`

Calculate the Shapley and Banzhaf values of a 2-additive fuzzy measure for n inputs given in Mobius representation. The results are in arrays s or b. Example:

```
v = [0.0, 0.3, 0.5, -0.2, 0.4, 0.1]
s=fm.Shapley2addMob(v,3)
b=fm.Banzhaf2addMob(v,3)
```

`fm.Choquet2addMob(x, v, n)`

Calculates the Choquet integral value of x for a 2-additive fuzzy measure for n inputs given in Mobius representation. Example:

```
x=[0.2,0.5,0.4]
v=[0.2, 0.3, 0.5, -0.2, 0.4, 0.1]
r=fm.Choquet2addMob(x, v, 3)
```

`fm.OWA( x, w, n)`

Returns the OWA function of x wrt weights w. Example:

```
n=3
x=[0.2,0.5,0.4]
w=[0.3,0.3,0.4]
r=fm.OWA( x, w, n)
```

`fm.WAM( x, w, n)`

returns the WAM function of x wrt weights w. Example:

```
n=3
x=[0.2,0.5,0.4]
w=[0.3,0.3,0.4]
r=fm.WAM( px, pw, n)
```

4.3.2 Without `Python` wrapper, using `numpy` and `cffi`

The sparse capacity structure needs to be defined as follows:

```
envsp=ffi.new("struct fm_env_sparse *")
```

`fm.py_prepare_fm_sparse(n, tupsize, tuples, envsp)`

This function initialises this structure, if the number of tuples is known, otherwise can be initialised with 0 tuples (i.e. set tup=0). Given the list of cardinalities of the nonzero tuples (cardinality, tuple composition) like this: 2 pairs 4-tuple and a triple: (2,1,2, 2, 3,1, 4, 1,2,3,4, 3,3,2,1...) It is used to allocate storage and later populate these values.

```
n=3
tup=0
tuples=np.zeros( 0, np.intc)
ptuples = ffi.cast("int *", tuples.ctypes.data)
fm.py_prepare_fm_sparse(n, tup, ptuples, envsp)
```

`fm.py_free_fm_sparse(envsp)`

Frees the memory previously allocated in *envsp*.

`fm.py_tuple_cardinality_sparse(i, envsp)`

Returns the cardinality of the tuple numbered i in the list of tuples.

`fm.py_get_num_tuples(envsp)`

Returns the number of tuples.

`fm.py_get_sizearray_tuples(envsp)`

Returns the length of the array of tuples.

`fm.py_is_inset_sparse(A, card, i, envsp)`
: Checks if element i (1-based) belongs to the tuple indexed A (whose cardinality can be 1,2, other (automatically determined))

`fm.py_is_subset_sparse(A, cardA, B, cardB, envsp)`
: Checks if tuple B is a subset of tuple A, The cardinalities of both tuples need to be supplied.

`fm.py_min_subset_sparse(x, n, S, cardS, envsp)`
`fm.py_max_subset_sparse(x, n, S, cardS, envsp)`
: calculate minimum (maximum) of x_i with the indices belonging to tuple indexed as S (its cardinality cardS can be 1,2, other (put 3, will be determined automatically)). Note that x is 0-based, tuples are 1-based. Example.

```
x=np.array([0.1,0.05,0.2],np.float)
px = ffi.cast("double *", x.ctypes.data)
fm.py_min_subset_sparse(px, 3, 0, 3, envsp)
fm.py_max_subset_sparse(px, 3, 0, 3, envsp)
```

`fm.py_ChoquetMob_sparse(x, n, envsp)`
: Calculates the Choquet integral in Mobius sparse representation. Example.

```
x=np.array([0.1,0.05,0.2],np.float)
px = ffi.cast("double *", x.ctypes.data)
n=3
fm.py_ChoquetMob_sparse(px, n, envsp)
```

`fm.py_ShapleyMob_sparse(v, n, envsp)`
`fm.py_BanzhafMob_sparse(v, n, envsp)`
: Calculate Shapley and Banzhaf values vectors (of size n) of a sparse fuzzy measure, returned in v. Example.

```
n=3
v=np.zeros(3, np.float)
pv = ffi.cast("double *", v.ctypes.data)
fm.py_ShapleyMob_sparse(pv, n, envsp)
v_1=np.zeros(3,np.float)
pv_1 = ffi.cast("double *", v_1.ctypes.data)
fm.py_BanzhafMob_sparse(pv_1, n, envsp)
```

`fm.py_Nonmodularityindex_sparse(w, n, envsp)`
: Calculates all 2^n nonmodularity indices (returned in w) using Mobius transform of a fuzzy measure (of length $2^n = m$), using sparse representation. Example.

```
n=3
w=np.zeros(2**n, np.float)
pw = ffi.cast("double *", w.ctypes.data)
fm.py_Nonmodularityindex_sparse(pw, n, envsp)
```

`fm.py_populate_fm_2add_sparse(singletons, numpairs, pairs, indicesp1, indicesp2, envsp)`

Populates 2-additive sparse capacity with nonzero values using the singletons and two arrays of indices (of size *numpairs*). Indices need to be 1-based. Singletons 0-based. Example.

```
singletons=np.array([0.1,0.2,0.3],np.float)
psingletons = ffi.cast("double *", singletons.ctypes.data)
numpairs=3
pairs = np.array([0.4,0.5,0.6],np.float)
ppairs = ffi.cast("double *", pairs.ctypes.data)
indicesp1 = np.array([1,1,2], np.intc)
pindicesp1 = ffi.cast("int *", indicesp1.ctypes.data)
indicesp2 = np.array([2,3,3], np.intc)
pindicesp2 = ffi.cast("int *", indicesp2.ctypes.data)
fm.py_populate_fm_2add_sparse(psingletons, numpairs, ppairs, pindicesp1,
pindicesp2,envsp)
```

`fm.py_add_pair_sparse(i, j, v, envsp)`

For populating capacities. Add pair (v_{ij}) to the structure. Indices are 1-based. Example.

```
fm.py_add_pair_sparse(1, 2, 0.4, envsp)
fm.py_add_pair_sparse(1, 3, 0.5, envsp)
fm.py_add_pair_sparse(2, 3, 0.6, envsp)
```

`fm.py_add_tuple_sparse(tupsize, tuple, v, envsp)`

For populating capacities, adds a tuple of size *tupsize* whose 1-based indices are in *tuple* Example:

```
tup=np.array([1,3,4],np.intc)
ptup = ffi.cast("int *", tup.ctypes.data)
v=0.2
fm.py_add_tuple_sparse(3, ptup, v, envsp)
fm.py_add_pair_sparse(2,5, v, envsp)
v=np.array([0.1,0.2,0.3],np.float)
pv = ffi.cast("double *", v.ctypes.data)
fm.py_add_singletons_sparse(pv,envsp)
```

`fm.py_populate_fm_2add_sparse_from2add(n, v, envsp)`

Given 2-additive capacity (singletons+pairs in one array v) , selects nonzero pairs and populates sparse capacity *envsp*. Example:

```
n=3
v=np.zeros(8,np.float)
pv = ffi.cast("double *", v.ctypes.data)
fm.py_generate_fm_2additive_convex_withsomeindependent(1, n, pv)
tuples=np.zeros( 0, np.intc)
ptuples = ffi.cast("int *", tuples.ctypes.data)
fm.py_prepare_fm_sparse(n, 0 , ptuples , envsp)
fm.py_populate_fm_2add_sparse_from2add(n, pv, envsp)
```

`fm.py_expand_2add_full(v, envsp)`

From sparse to full representation of 2-additive capacity (singletons and pairs, augmented with 0s). Vector v has to be allocated, its size is $n + n(n-1)/2$. Example:

```
v1=np.array([0.0,0.0,0.0,0.0,0.0,0.0],np.float)
pv1 = ffi.cast("double *", v1.ctypes.data)
fm.py_expand_2add_full(pv1, envsp)
```

`fm.py_expand_sparse_full(v, envsp)`

Exports from sparse to full capacity (vector v, size 2^n has to be preallocated). Example:

```
v=np.zeros(8,np.float)
pv = ffi.cast("double *", v.ctypes.data)
fm.py_expand_sparse_full(pv, envsp)
```

`fm.py_sparse_get_singletons(n, v, envsp)`
`fm.py_sparse_get_pairs(pairs, v, envsp)`
`fm.py_sparse_get_tuples(tuples, v, envsp)`

Export the internal arrays of the sparse capacity as arrays of singletons, pairs and tuples. Return the numbers of pairs and tuples. Example:

```
s=np.zeros(n,np.float)
ps = ffi.cast("double *", s.ctypes.data)
fm.py_sparse_get_singletons(n, ps,envsp)

pairs=np.zeros(32,np.intc)
ppairs = ffi.cast("int *", pairs.ctypes.data)
vals=np.zeros(32,np.float)
pvals = ffi.cast("double *", vals.ctypes.data)
sizepairs= fm.py_sparse_get_pairs(ppairs, pvals,envsp)

indextuples=np.zeros(32,np.intc)
pindextuples = ffi.cast("int *", indextuples.ctypes.data)
vvt=np.zeros(32,np.float)
pvvt = ffi.cast("double *", vvt.ctypes.data)
sizetuples = fm.py_sparse_get_tuples(pindextuples, pvvt,envsp)
```

The following functions do not use sparse representation but they do not require initialisation by `py_fm_init` either, they work with 2-additive fuzzy measures stored on an array of Möbius values of singletons and pairs.

`fm.py_dualMobKadd(n, length, k, src, dest, env)`

Calculate the dual of a k-additive fuzzy measures for n inputs. Generally requires initialisation but not for 2-additive fuzzy measures. Parameters: *length* is the size of the array in cardinality based ordering (can be calculated by function `py_fm_arraysize`) , k as in k-additivity, *src* is the source measure, *dest* is its dual, *env* must be initialised for $k > 2$ only. Example:

```
k=2
n=20
env=ffi.new("struct fm_env *") # no need to initialise, not used
```

```
sz=fm.py_fm_arraysize(n,k,env)
v=np.zeros( n*n, np.float)
pv = ffi.cast("double *", v.ctypes.data)
d=np.zeros( n*n, np.float)
pd = ffi.cast("double *", d.ctypes.data)
lengthv=fm.py_generate_fm_2additive_convex(1, n, pv)
fm.py_dualMobKadd(n, lengthv, k, pv, pd, env)
```

`fm.py_Shapley2addMob(v, s, n)`
`fm.py_Banzhaf2addMob(v, b, n)`

Calculate the Shapley and Banzhaf values of a 2-additive fuzzy measure for n inputs given in Mobius representation. The results are in arrays s or b. Example:

```
n=20
s=np.zeros( n, np.float)
ps = ffi.cast("double *", s.ctypes.data)
b=np.zeros( n, np.float)
pb = ffi.cast("double *", b.ctypes.data)
fm.py_Shapley2addMob(pv,ps,n)
fm.py_Banzhaf2addMob(pv,pb,n)
```

`fm.py_Choquet2addMob(x, v, n)`

Calculates the Choquet integral value of x for a 2-additive fuzzy measure for n inputs given in Mobius representation. Example:

```
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
v=np.array([0,0.3,0.5,0.6,0.4,0.8,0.7,1],np.float)
pv = ffi.cast("double *", v.ctypes.data)
r=fm.py_Choquet2addMob(px, pv, 3)
```

`fm.py_OWA( x, w, n)`

Returns the OWA function of x wrt weights w. Example:

```
n=3
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
w=np.array([0.3,0.3,0.4],np.float)
pw = ffi.cast("double *", w.ctypes.data)
r=fm.py_OWA( px, pw, n)
```

`fm.py_WAM( x, w, n)`

returns the WAM function of x wrt weights w. Example:

```
n=3
x=np.array([0.2,0.5,0.4],np.float)
px = ffi.cast("double *", x.ctypes.data)
w=np.array([0.3,0.3,0.4],np.float)
pw = ffi.cast("double *", w.ctypes.data)
r=fm.py_WAM( px, pw, n)
```

4.4 Fuzzy measure fitting routines

The package contains six functions taking taking the empirical data, some options, and returning the (k-additive) fuzzy measure.

4.4.1 Using `Python` wrapper

Important note: after fitting, the user must re-initialise the internal structures with `fm_init`, because these functions create these structures internally and then free the memory.

`fm.FuzzyMeasureFit(datanum, kadditive, env, dataset)`

Returns (k-additive) fuzzy measure fitted to data in standard representation. Example:

```
n=3
datanum=10
kadditive=2
env=fm.fm_init(n);
dataset=np.random.rand(datanum,n+1)
v= fm.py_FuzzyMeasureFit(datanum, kadditive, env, dataset)
```

`fm.FuzzyMeasureFitMob(datanum, kadditive, env, dataset)`

Returns (k-additive) fuzzy measure fitted to data in Möbius representation (binary ordering). Example:

```
datanum=10
kadditive=3
dataset=np.random.rand(datanum,n+1)
mob=fm.FuzzyMeasureFitMob(datanum, kadditive, env, pdataset)
```

`fm.FuzzyMeasureFitLP(datanum, additive, env, v, dataset, options, indexlow, indexhihg, option1, orness)`

Returns fuzzy measure fitted to data in standard representation, with optional constraints on the orness measure and the upper and lower values of Shapley values and interaction indices. Example:

```
datanum=10
additive=2
dataset=np.random.rand(datanum,n+1)
options=0
indexlow = [0,0,0]
indexhihg = [1,1,1]
option1=0
orness = 0.5
v= fm.py_FuzzyMeasureFitLP(datanum, additive, env,dataset, options, indexlow,
indexhihg, option1, orness)
```

`fm.FuzzyMeasureFitLPMob(datanum, additive, env, dataset, options, indexlow, indexhihg, option1, orness)`

Returns fuzzy measure in Möbius representation fitted to data in standard representation, with optional constraints on the orness measure and the upper and lower values of Shapley values and interaction indices. Example:

```
datanum=10
additive=2
dataset=np.random.rand(datanum,n+1)
options=0
indexlow = [0,0,0]
indexhihg = [1,1,1]
option1=0
orness = 0.5
mob= fm.FuzzyMeasureFitLPMob(datanum, additive, env, mob, dataset, options,
indexlow, indexhihg, option1, orness)
```

`fm.FuzzyMeasureFitKtolerant(datanum, additive, env, dataset)`

Returns k-tolerant fuzzy measure fitted to data in standard representation, with no optional constraints. Example:

```
datanum=10
additive=2
dataset=np.random.rand(datanum,n+1)
v=fm.py_FuzzyMeasureFitKtolerant(datanum, additive, env, dataset)
```

`fm.FuzzyMeasureFitLPKmaxitive(datanum, additive, env, dataset)`

Returns k-maxitive fuzzy measure fitted to data in standard representation, with no optional constraints. Example:

```
datanum=10
additive=2
dataset=np.random.rand(datanum,n+1)
v=fm.FuzzyMeasureFitLPKmaxitive(datanum, additive, env, dataset)
```

`fm.fittingOWA(datanum, env, v, dataset)`

Returns symmetric fuzzy measure fitted to data. The vector of OWA weights is returned as in this case the Choquet integral becomes the OWA function. Example:

```
datanum=10
dataset=np.random.rand(datanum,n+1)
v=fm.fittingOWA(datanum, env, dataset)
```

`fm.fittingWAM(datanum, env, v, dataset)`

Returns additive fuzzy measure fitted to data. The vector of WAM weights is returned as in this case the Choquet integral becomes the WAM function. Example:

```
datanum=10
dataset=np.random.rand(datanum,n+1)
v=fm.fittingWAM(datanum, env, dataset)
```

List of parameters and options of the `fm.FuzzyMeasureFitLP` function.

`fm.FuzzyMeasureFitLP(datanum, additive, env, v, dataset, options, indexlow, indexhihg, option1, orness)`

data - an array of size $K \times (n+1)$, where each row is the pair $(\mathbf{x}_k, y_k)$, $\mathbf{x}_k \in [0,1]^n$, $y_k \in [0,1]$, K data altogether.
$Kadd$ - k in k-additive fuzzy measures, $1 < Kadd < n+1$, if $Kdd = n$ - f.m. is unrestricted.

options (default value is 0)

- 1 - lower bounds on Shapley values supplied in *indexlow*
- 2 - upper bounds on Shapley values supplied in *indexhigh*
- 3 - lower and upper bounds on Shapley values supplied in *indexlow* and *indexhigh*
- 4 - lower bounds on all interaction indices supplied in *indexlow*
- 5 - upper bounds on all interaction indices supplied in *indexhigh*
- 6 - lower and upper bounds on all interaction indices supplied in *indexlow* and *indexhigh*.

all these value will be treated as additional constraints in the LP.

indexlow, *indexhigh* - array of size n (options =1,2,3) or m (options=4,5,6) containing the lower and upper bounds on the Shapley values or interaction indices

*options*1 (default 0) is a flag whose bits indicate which additional properties are needed. If the first bit is set then the desired interval of orness values specified in array *orness* will be used. If the second bit is set, then the f.m. will be forced to be balanced (currently not implemented). If the third bit is set, then in addition to fitting the data, the order of output values will be preserved.

If the fourth bit is set, the fuzzy measure will be forced to be submodular.
Note that this constraint may lead to inconsistent set of conditions, in which case the problem will have no solution (and no output vector returned).

orness - array of size 2 which contains the lower and the upper bounds on the orness value. These values should be from [0,1], and could coincide if an exact orness value is needed. If the bounds are 0 and 1 respectively they are ignored. Only used if the first bit of *options*1 is set.

Notes: 1. arrays *indexlow* and *indexhigh* are 0-based if they contain Shapley values (i.e., the bound on Shapley value of the first input is in indexlow[0], etc. but when these arrays contain interaction indices, these are 1-based (since there is a non-zero value of the interaction index corresponding to empty set). In this case the bounds are arranged in cardinality order, i.e., the bounds correspond to the sets in this ordering (for n=3) $\emptyset\{1\}\{2\}\{3\}\{12\}\{13\}\{23\}\{123\}$.
2. If an exact value of some index or Shapley value is needed, use $indexlow[i] = indexhigh[i] = thisvalue$ if no value for some index is required, use $indexlow[i] = -1, indexhigh[i] = 1$.
3. Note that Shapley values have range [0,1], whereas interaction indices have range [-1,1].

Note that the options as listed above are not implemented for the following methods.

`fm.FuzzyMeasureFitLPKinteractive(datanum, additive, env, v, dataset, K)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). Example:

```
datanum=10
additive = 2
dataset=np.random.rand(datanum,n+1)
K=0.5
v = fm.FuzzyMeasureFitLPKinteractive(datanum, additive, env, dataset,
K)
```

`fm.FuzzyMeasureFitLPKinteractiveAutoK(datanum, additive, env, v, dataset, K, maxiters)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with the parameter KC automatically determined from the data. Example:

```
datanum=10
additive = 2
dataset=np.random.rand(datanum,n+1)
K=0.5
maxiters=100
v=fm.FuzzyMeasureFitLPKinteractiveAutoK(datanum, additive, env, dataset,
K, maxiters)
```

`fm.FuzzyMeasureFitLPKinteractiveMaxChains(datanum, additive, env, dataset, K)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses maximal chain approach and is more efficient for smaller data sets. Example:

```
datanum=10
additive = 2
dataset=np.random.rand(datanum,n+1)
K=0.5
v=fm.FuzzyMeasureFitLPKinteractiveMaxChains(datanum, additive, env, dataset,
K)
```

`fm.FuzzyMeasureFitLPKinteractiveMarginal(datanum, additive, env, dataset, K, submod)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses marginal contributions representation and may involve more variables and constraints than the other methods. It is useful when imposing sub- or supermodularity constraints, in which case parameter *submod* should be set to -1 (supermodularity), +1 (submodularity), or 0 (none). Example:

```
datanum=10
additive = 2
dataset=np.random.rand(datanum,n+1)
```

```
K=0.5
submod=1
v=fm.FuzzyMeasureFitLPKinteractiveMarginal(datanum, additive, env, dataset,
K, submod)
```

`fm.FuzzyMeasureFitLPKinteractiveMarginalMaxChain(datanum, additive, env, dataset, K, maxiters, submod)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses marginal contributions representation and maximal chains approach, and may involve more variables and constraints than the other methods. It is useful when imposing sub- or super-modularity constraints, in which case parameter *submod* should be set to -1 (supermodularity), +1 (submodularity), or 0 (none). It is more efficient than $fittingKinteractiveMarginal$ in case of a small data set.

Note: at the moment only supermodular option works in this method. If a submodular measure is needed, fit its dual supermodular to dual data. Example:

```
datanum=10
additive = 2
dataset=np.random.rand(datanum,n+1)
K=0.5
maxiters=100
submod=1
v=fm.FuzzyMeasureFitLPKinteractiveMarginalMaxChain(datanum, additive,
env, dataset, K, maxiters, submod)
```

4.4.2 Without Python wrapper, using numpy and cffi

Important note: after fitting, the user must re-initialise the internal structures with `py_fm_init`, because these functions create these structures internally and then free the memory.

`fm.py_FuzzyMeasureFit(datanum, kadditive, env, v, dataset)`

Returns (k-additive) fuzzy measure fitted to data in standard representation. Example:

```
n=3
env=ffi.new("struct fm_env *")
datanum=10
kadditive=2
fm.py_fm_init(n, env);
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
fm.py_FuzzyMeasureFit(datanum, kadditive, env, pv, pdataset)
```

`fm.py_FuzzyMeasureFitMob(datanum, kadditive, env, Mob, dataset)`

Returns (k-additive) fuzzy measure fitted to data in Möbius representation (binary ordering). Example:

```
datanum=10
kadditive=3
mob=np.array([0,0,0,0,0,0,0,0],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
fm.py_FuzzyMeasureFitMob(datanum, kadditive, env, pmob, pdataset)
```

`fm.py_FuzzyMeasureFitLP(datanum, additive, env, v, dataset, options, indexlow, indexhihg, option1, orness)`

Returns fuzzy measure fitted to data in standard representation, with optional constraints on the orness measure and the upper and lower values of Shapley values and interaction indices. Example:

```
datanum=10
additive=2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
options=np.array([0],np.intc)
poptions= ffi.cast("int *", options.ctypes.data)
indexlow = np.array([0,0,0],np.float)
pindexlow = ffi.cast("double *", indexlow.ctypes.data)
indexhihg = np.array([1,1,1],np.float)
pindexhihg = ffi.cast("double *", indexhihg.ctypes.data)
option1=np.array([0],np.intc)
poption1= ffi.cast("int *", option1.ctypes.data)
orness = np.array([0.5],np.float)
porness = ffi.cast("double *", orness.ctypes.data)
fm.py_FuzzyMeasureFitLP(datanum, additive, env, pv, pdataset, poptions,
pindexlow, pindexhihg, poption1, porness)
```

`fm.py_FuzzyMeasureFitLPMob(datanum, additive, env, Mob, dataset, options, indexlow, indexhihg, option1, orness)`

Returns fuzzy measure in Möbius representation fitted to data in standard representation, with optional constraints on the orness measure and the upper and lower values of Shapley values and interaction indices. Example:

```
datanum=10
additive=2
mob=np.array([0,0,0,0,0,0,0,0],np.float)
pmob = ffi.cast("double *", mob.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
options=np.array([0],np.intc)
```

```
poptions= ffi.cast("int *", options.ctypes.data)
indexlow = np.array([0,0,0],np.float)
pindexlow = ffi.cast("double *", indexlow.ctypes.data)
indexhihg = np.array([1,1,1],np.float)
pindexhihg = ffi.cast("double *", indexhihg.ctypes.data)
option1=np.array([0],np.intc)
poption1= ffi.cast("int *", option1.ctypes.data)
orness = np.array([0.5],np.float)
porness = ffi.cast("double *", orness.ctypes.data)
fm.py_FuzzyMeasureFitLPMob(datanum, additive, env, pmob, pdataset, poptions,
pindexlow, pindexhihg, poption1, porness)
```

`fm.py_FuzzyMeasureFitKtolerant(datanum, additive, env, v, dataset)`

Returns k-tolerant fuzzy measure fitted to data in standard representation, with no optional constraints. Example:

```
datanum=10
additive=2
v=np.array([0,0,0,0,0,0,0,0],float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
fm.py_FuzzyMeasureFitKtolerant(datanum, additive, env, pv, pdataset)
```

`fm.py_FuzzyMeasureFitLPKmaxitive(datanum, additive, env, v, dataset)`

Returns k-maxitive fuzzy measure fitted to data in standard representation, with no optional constraints. Example:

```
datanum=10
additive=2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
fm.py_FuzzyMeasureFitLPKmaxitive(datanum, additive, env, pv, pdataset)
```

`fm.py_fittingOWA(datanum, env, v, dataset)`

Returns symmetric fuzzy measure fitted to data. The vector of OWA weights is returned as in this case the Choquet integral becomes the OWA function. Example:

```
datanum=10
v=np.array([0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
fm.py_fittingOWA(datanum, env, pv, pdataset)
```

`fm.py_fittingWAM(datanum, env, v, dataset)`

Returns additive fuzzy measure fitted to data. The vector of WAM weights is returned as in this case the Choquet integral becomes the WAM function. Example:

```
datanum=10
v=np.array([0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
fm.py_fittingWAM(datanum, env, pv, pdataset)
```

List of parameters and options of the `fm.py_FuzzyMeasureFitLP` function.

`fm.py_FuzzyMeasureFitLP(datanum, additive, env, v, dataset, options, indexlow, indexhihg, option1, orness)`

data - an array of size $K \times (n+1)$, where each row is the pair $(\mathbf{x}_k, y_k)$, $\mathbf{x}_k \in [0,1]^n$, $y_k \in [0,1]$, K data altogether.

$Kadd$ - k in k-additive fuzzy measures, $1 < Kadd < n+1$, if $Kdd = n$ - f.m. is unrestricted.

options (default value is 0)

- 1 - lower bounds on Shapley values supplied in *indexlow*
- 2 - upper bounds on Shapley values supplied in *indexhigh*
- 3 - lower and upper bounds on Shapley values supplied in *indexlow* and *indexhigh*
- 4 - lower bounds on all interaction indices supplied in *indexlow*
- 5 - upper bounds on all interaction indices supplied in *indexhigh*
- 6 - lower and upper bounds on all interaction indices supplied in *indexlow* and *indexhigh*.

all these value will be treated as additional constraints in the LP.

indexlow, *indexhigh* - array of size n (options =1,2,3) or m (options=4,5,6) containing the lower and upper bounds on the Shapley values or interaction indices

*options*1 (default 0) is a flag whose bits indicate which additional properties are needed. If the first bit is set then the desired interval of orness values specified in array *orness* will be used. If the second bit is set, then the f.m. will be forced to be balanced (currently not implemented). If the third bit is set, then in addition to fitting the data, the order of output values will be preserved.

If the fourth bit is set, the fuzzy measure will be forced to be submodular.

Note that this constraint may lead to inconsistent set of conditions, in which case the problem will have no solution (and no output vector returned).

orness - array of size 2 which contains the lower and the upper bounds on the orness value. These values should be from [0,1], and could coincide if an exact orness value is needed. If the bounds are 0 and 1 respectively they are ignored. Only used if the first bit of *options*1 is set.

Notes: 1. arrays *indexlow* and *indexhigh* are 0-based if they contain Shapley values (i.e., the bound on Shapley value of the first input is in indexlow[0], etc. but when

these arrays contain interaction indices, these are 1-based (since there is a non-zero value of the interaction index corresponding to empty set). In this case the bounds are arranged in cardinality order, i.e., the bounds correspond to the sets in this ordering (for n=3) $\emptyset\{1\}\{2\}\{3\}\{12\}\{13\}\{23\}\{123\}$.
2. If an exact value of some index or Shapley value is needed, use $indexlow[i] = indexhigh[i] = thisvalue$ if no value for some index is required, use $indexlow[i] = -1, indexhigh[i] = 1$.
3. Note that Shapley values have range [0,1], whereas interaction indices have range [-1,1].

Note that the options as listed above are not implemented for the following methods.

`fm.py_FuzzyMeasureFitLPKinteractive(datanum, additive, env, v, dataset, K)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). Example:

```
datanum=10
additive = 2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
K=np.array([0.5],np.float)
pK = ffi.cast("double *", K.ctypes.data)
fm.py_FuzzyMeasureFitLPKinteractive(datanum, additive, env, pv, pdataset,
pK)
```

`fm.py_FuzzyMeasureFitLPKinteractiveAutoK(datanum, additive, env, v, dataset, K, maxiters)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with the parameter KC automatically determined from the data. Example:

```
datanum=10
additive = 2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
K=np.array([0.5],np.float)
pK = ffi.cast("double *", K.ctypes.data)
maxiters=np.array([100],np.intc)
pmaxiters = ffi.cast("int *", maxiters.ctypes.data)
fm.py_FuzzyMeasureFitLPKinteractiveAutoK(datanum, additive, env, pv, pdataset,
pK, pmaxiters)
```

`fm.py_FuzzyMeasureFitLPKinteractiveMaxChains(datanum, additive, env, v, dataset, K)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses maximal chain approach and is more efficient for smaller data sets. Example:

```
datanum=10
additive = 2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
K=np.array([0.5],np.float)
pK = ffi.cast("double *", K.ctypes.data)
fm.py_FuzzyMeasureFitLPKinteractiveMaxChains(datanum, additive, env, pv,
pdataset, pK)
```

`fm.py_FuzzyMeasureFitLPKinteractiveMarginal(datanum, additive, env, v, dataset, K, submod)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses marginal contributions representation and may involve more variables and constraints than the other methods. It is useful when imposing sub- or super-modularity constraints, in which case parameter *submod* should be set to -1 (supermodularity), +1 (submodularity), or 0 (none). Example:

```
datanum=10
additive = 2
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
K=np.array([0.5],np.float)
pK = ffi.cast("double *", K.ctypes.data)
submod=1
fm.py_FuzzyMeasureFitLPKinteractiveMarginal(datanum, additive, env, pv,
pdataset, pK, submod)
```

`fm.py_FuzzyMeasureFitLPKinteractiveMarginalMaxChain(datanum, additive, env, v, dataset, K, maxiters, submod)`

Returns k-interactive fuzzy measure fitted to data in standard representation, with fixed parameter KC (the value $v(A)$ at sets of cardinality $k+1$). This method uses marginal contributions representation and maximal chains approach, and may involve more variables and constraints than the other methods. It is useful when imposing sub- or super-modularity constraints, in which case parameter *submod* should be set to -1 (supermodularity), +1 (submodularity), or 0 (none). It is more efficient than $fittingKinteractiveMarginal$ in case of a small data set.

Note: at the moment only supermodular option works in this method. If a submodular measure is needed, fit its dual supermodular to dual data. Example:

```
datanum=10
additive = 2
```

```
v=np.array([0,0,0,0,0,0,0,0],np.float)
pv = ffi.cast("double *", v.ctypes.data)
dataset=np.random.rand(datanum,n+1)
pdataset = ffi.cast("double *", dataset.ctypes.data)
K=np.array([0.5],np.float)
pK = ffi.cast("double *", K.ctypes.data)
maxiters=np.array([100])
pmaxiters = ffi.cast("int *", maxiters.ctypes.data)
submod=1
fm.py_FuzzyMeasureFitLPKinteractiveMarginalMaxChain(datanum, additive,
env, pv, pdataset, pK, pmaxiters, submod)
```

4.5 Random generation of fuzzy measures

The package `pyfmtools` provides several methods to randomly generate fuzzy measures of different types: general, k-interactive, k-additive convex and concave (ie. supermodular and submodular). General and k-interactive fuzzy measures require initialisation of the *env* structures, but sparse and k-additive (for small k) do not, see section 4.3.

4.5.1 Using `Python` wrapper

`fm.generate_fm_tsort(num, n, kint, markov, option, K, env)`
`fm.generate_fm_minplus(num, n, kint, markov, option, K, env)`

Both functions generate several random fuzzy measures (num is their number) stored in **cardinality** ordering in the array v (the ith fuzzy measure starts at position $i * lengthv$ and has length $length$, which is returned by this function). The tsort method is based on topological sort and minplus is based on MinimalsPlus method (see Section 2.7). Other parameters: $0 < kint \leq n$ is for k-interactive fuzzy measures, $markov$ - how many Markov steps to take, the randomness increases with that number, but slows down the process, K is the constant in k-interactive fuzzy measures, v is the array to store the generated values (preallocated of size $num * 2^n$ or less for k-interactive fuzzy measures, see `fm.py_fm_arraysize`). $option = 1$ employs internal rejection method to improve uniformity, but for $n > 5$ is is not essential. Returns the size of the array for each generated fuzzy measure. Example:

```
num=10
n=6
kint=3
env=fm.fm_init(n)
lengthv, v =fm.generate_fm_tsort(num, n, kint, 1000, 0, 0.7, env)
```

`fm.generate_fmconvex_tsort(num, n, kint, markov, option, K, env)`

Generates num convex random fuzzy measures stored consecutively in **cardinality** ordering in the array v (the ith fuzzy measure starts at position $i*2^n$ and has length

2^n). The parameters are as for the previous two functions. Returns the size of the array for each generated fuzzy measure. *kint* for the moment is not implemented. This method is quite reliable for $n < 10$ but then becomes slow. There is an internal rejection process which guarantees supermodularity for small $n < 9$, but otherwise the result must be checked as in the example below. Increase Markov chain length for $n > 6$. Example:

```
num=1
n=5
env=fm.fm_init(n)
len=fm.fm_arraysize(n, n, env)
lengthv,v=fm.py_generate_fmconvex_tsort(num, n, n, 1000, 0, 1, env)
w=fm.ConvertCard2Bit( v, env)
super=fm.IsMeasureSupermodular(w, env)
```

The following routines do not require initialisation of the fuzzy measure with $_fm\_initandcanhencebeusedwithlarge$`n`$(limitedbytheavailablememory).Notethat2-additivefuzzymeasuresrequire$`O(n`^{2}`) parameters. They are stored in cardinality ordering.`

```
fm.generate_fm_2additive_convex(num, n)
fm.generate_fm_2additive_concave(num, n)
fm.generate_fm_2additive_convex_withsomeindependent(num, n)
```

`Generates` *num* `2-additive convex (supermodular) or concave (submodular) fuzzy measures for` *n* `inputs. Returns the length of the part of the array` *v* `allocated for each fuzzy measure, and the array with singletons and pairs in Mobius representation. The array needs to be reserved in the calling program of size at most` $num{*}n^2$`. The routine ‘‘withsomeindependent’’ means that a proportion of pairwise interaction indices will be set to 0, making these pairs independent. The different fuzzy measures are stored consecutively, so that the` *i*`th measure starts at position` $i{*}lengthv$`. Example:`

```
num=10
n=20
lengthv,v=fm.py_generate_fm_2additive_convex(num, n)
```

```
fm.generate_fm_2additive_convex_sparse( n, envsp)
```

`Generates a random 2-additive supermodular fuzzy measure in sparse representation. Some Möbius values will be set to 0. Example:`

```
n=5
tup=0
envsp=fm.py_prepare_fm_sparse(n, tup)
fm.py_generate_fm_2additive_convex_sparse(n, envsp)
```

```
fm.generate_fm_kadditive_convex_sparse( n, k, nonzero, envsp)
```

`Generates a random k-additive Belief fuzzy measure (i.e. totally monotone) in sparse representation. Some Möbius values will be set to 0. Example:`

```
n=5
tup=0
envsp=fm.py_prepare_fm_sparse(n, tup)
k=4
fm.generate_fm_Kadditive_convex_sparse(n, k, 10, envsp)
```

4.5.2 Without Python wrapper, using numpy and cffi

`fm.py_generate_fm_tsort(num, n, kint, markov, option, K, v, env)`
`fm.py_generate_fm_minplus(num, n, kint, markov, option, K, v, env)`

Both functions generate several random fuzzy measures (num is their number) stored in cardinality ordering in the array v (the ith fuzzy measure starts at position $i{*}lengthv$ and has length $length$, which is returned by this function). The tsort method is based on topological sort and minplus is based on MinimalsPlus method (see Section 2.7). Other parameters: $0 < kint \leq n$ is for k-interactive fuzzy measures, $markov$ - how many Markov steps to take, the randomness increases with that number, but slows down the process, K is the constant in k-interactive fuzzy measures, v is the array to store the generated values (preallocated of size $num*2^n$ or less for k-interactive fuzzy measures, see fm.py_fm_arraysize). $option = 1$ employs internal rejection method to improve uniformity, but for $n > 5$ is is not essential. Returns the size of the array for each generated fuzzy measure. Example:

```
num=10
n=6
kint=3
env=ffi.new("struct fm_env *")
fm.py_fm_init(n, env)
len=fm.py_fm_arraysize(n, kint, env)
v=np.zeros(num * len, np.float)
pv = ffi.cast("double *", v.ctypes.data)
lengthv=fm.py_generate_fm_tsort(num, n, kint, 1000, 0, 0.7, pv, env)
```

`fm.py_generate_fmconvex_tsort(num, n, kint, markov, option, K, v, env)`

Generates num convex random fuzzy measures stored consecutively in cardinality ordering in the array v (the ith fuzzy measure starts at position $i{*}2^n$ and has length 2^n). The parameters are as for the previous two functions. Returns the size of the array for each generated fuzzy measure. $kint$ for the moment is not implemented. This method is quite reliable for $n < 10$ but then becomes slow. There is an internal rejection process which guarantees supermodularity for small $n < 9$, but otherwise the result must be checked as in the example below. Increase Markov chain length for $n > 6$. Example:

```
num=1
n=5
```

```
env=ffi.new("struct fm_env *")
fm.py_fm_init(n, env)
len=fm.py_fm_arraysize(n, n, env)
v=np.zeros(num * len, np.float)
pv = ffi.cast("double *", v.ctypes.data)
lengthv=fm.py_generate_fmconvex_tsort(num, n, n, 1000, 0, 1, pv, env)
w=np.zeros(num * len, np.float)
pw = ffi.cast("double *", w.ctypes.data)
fm.py_ConvertCard2Bit(pw, pv, env)
super=fm.py_IsMeasureSupermodular(pw, env)
```

The following routines do not require initialisation of the fuzzy measure with py_fm_init and can hence be used with large n (limited by the available memory). Note that 2-additive fuzzy measures require $O(n^2)$ parameters. They are stored in cardinality ordering.

fm.py_generate_fm_2additive_convex(num, n, v)
fm.py_generate_fm_2additive_concave(num, n, v)
fm.py_generate_fm_2additive_convex_withsomeindependent(num, n, v)

Generates num 2-additive convex (supermodular) or concave (submodular) fuzzy measures for n inputs. Returns the length of the part of the array v allocated for each fuzzy measure, and the array with singletons and pairs in Mobius representation. The array needs to be reserved in the calling program of size at most $num*n^2$. The routine ''withsomeindependent'' means that a proportion of pairwise interaction indices will be set to 0, making these pairs independent. The different fuzzy measures are stored consecutively, so that the ith measure starts at position $i*lengthv$. Example:

```
num=10
n=20
v=np.zeros(num * n*n, np.float)
pv = ffi.cast("double *", v.ctypes.data)
lengthv=fm.py_generate_fm_2additive_convex(num, n, pv)
```

fm.py_generate_fm_2additive_convex_sparse(n, envsp)

Generates a random 2-additive supermodular fuzzy measure in sparse representation. Some Möbius values will be set to 0. Example:

```
env_sp=ffi.new("struct fm_env_sparse *")
n=5
tup=0
tuples=np.zeros( 0, np.intc)
ptuples = ffi.cast("int *", tuples.ctypes.data)
fm.py_prepare_fm_sparse(n, tup, ptuples, env_sp)
fm.py_generate_fm_2additive_convex_sparse(n, env_sp)
```

fm.py_generate_fm_kadditive_convex_sparse(n, k, nonzero, envsp)

Generates a random k-additive Belief fuzzy measure (i.e. totally monotone) in sparse representation. Some Möbius values will be set to 0. Example:

```
env_sp=ffi.new("struct fm_env_sparse *")
n=5
tup=0
tuples=np.zeros( 0, np.intc)
ptuples = ffi.cast("int *", tuples.ctypes.data)
fm.py_prepare_fm_sparse(n, tup, ptuples, env_sp)
k=4
fm.py_generate_fm_Kadditive_convex_sparse(n, k, 10, env_sp)
```

4.6 Examples

```
import numpy as np
import math
from  pyfmtools import ffi,lib as fm

n=3
env=ffi.new("struct fm_env *")
fm.py_fm_init(n,  env);

A=np.zeros(env.m, int)
pA = ffi.cast("int *", A.ctypes.data)
fm.py_ShowCoalitions(pA, env)
print("Fuzzy measure wrt n=3 criteria has ",env.m," in binary order")
print(A)
fm.py_fm_free( env);

total=1
n=4
fm.py_fm_init(n,  env);
v=np.zeros(env.m,float);
pv = ffi.cast("double *", v.ctypes.data);
vb=np.zeros(env.m,float);
pvb = ffi.cast("double *", vb.ctypes.data);

size=fm.py_generate_fm_2additive_concave(total,n,pv)
print("2-additive concave FM in Mobius and its length (n=4)")
print(v)

print("A convex FM in cardinality ordering ")
A=np.zeros(env.m, int)
pA = ffi.cast("int *", A.ctypes.data)
fm.py_ShowCoalitionsCard(pA, env)
print(A)

size=fm.py_generate_fmconvex_tsort(total,n, n-1 , 1000, 8, 1, pv,env)
#size=fm.py_generate_fm_tsort(total,n, 2 , 10, 0, 0.1, pv,env)
print(v)

fm.py_ConvertCard2Bit(pvb,pv,env)
print("a convex FM in binary ordering ")
fm.py_ShowCoalitions(pA, env)
print(A)
print(vb)
r=fm.py_IsMeasureSupermodular(pvb,env)
```

```
print("Is it convex (test)?", r)
r=fm.py_IsMeasureAdditive(pvb,env)
print("Is it additive (test)?", r)

mc=np.zeros(math.factorial(n)*n,float)
pmc = ffi.cast("double *", mc.ctypes.data)

fm.py_export_maximal_chains(n,pvb,pmc,env)
print("Export maximal chains ")
print(mc)

S=np.zeros(n,float)
pS = ffi.cast("double *", S.ctypes.data)
fm.py_Shapley(pvb,pS,env)
print("Shapley values")
print(S)

x=np.array([0.2,0.1,0.6,0.2])
px = ffi.cast("double *", x.ctypes.data)

z=fm.py_Choquet(px,pvb,env)
print("Choquet integral of ",x, " is ",z)
z=fm.py_Sugeno(px,pvb,env)
print("Sugeno integral of ",x,  " is ",z)

fm.py_fm_free( env);
```

4.7 Where to get help

The software library pyfmtools and its components, are distributed by G.Beliakov AS IS, with no warranty, explicit or implied, of merchantability or fitness for a particular purpose. G.Beliakov, at his sole discretion, may provide advice to registered users on the proper use of pyfmtools and its components.

Any queries regarding technical information, sales and licensing should be directed to gleb@deakin.edu.au. I am interested to learn about your experiences using pyfmtools , bugs, suggestions, its usefulness, applying it in practice and so on.

If you want to cite pyfmtools package, use references [2--7, 21,22,24].

Bibliography

[1] J.F. Banzhaf. Weight voting doesn't work: A mathematical analysis. *Rutgers Law Review*, 19:317--343, 1965.

[2] G. Beliakov, H. Bustince, and T. Calvo. *A Practical Guide to Averaging Functions*. Studies in Fuzziness and Soft Computing. Springer-Verlag, Berlin, Heidelberg, Berlin, 2016.

[3] G. Beliakov, S. James, and G. Li. Learning Choquet-integral-based metrics for semisupervised clustering. *IEEE Trans. on Fuzzy Systems*, 19:562--574, 2011.

[4] G. Beliakov, S. James, and J.-Z. Wu. *Discrete Fuzzy Measures: Computational Aspects*. Springer, Cham, 2020.

[5] G. Beliakov, A. Pradera, and T. Calvo. *Aggregation Functions: A Guide for Practitioners*, volume 221 of *Studies in Fuzziness and Soft Computing*. Springer-Verlag, Berlin, 2007.

[6] G. Beliakov and J.-Z. Wu. Learning fuzzy measures from data: simplifications and optimisation strategies. *Information Sciences*, 494:100--113, 2019.

[7] Gleb Beliakov and Jian-Zhang Wu. Learning fuzzy measures from data: simplifications and optimisation strategies. *Information Sciences*, 494:100--113, 2019.

[8] E. Combarro, J. Hurtado de Saracho, and I Díaz. Minimals plus: An improved algorithm for the random generation of linear extensions of partially ordered sets. *Information Sciences*, 501:50 -- 67, 2019.

[9] J.J. Dujmovic. Weighted conjunctive and disjunctive means and their application in system evaluation. *Univ. Beograd. Publ. Elektrotechn. Fak.*, pages 147--158, 1974.

[10] J. Fodor and M. Roubens. *Fuzzy Preference Modelling and Multicriteria Decision Support*. Kluwer, Dordrecht, 1994.

[11] M. Grabisch. The applications of fuzzy integrals in multicriteria decision making. *Europ. J. Operations Research*, 89:445--456, 1996.

[12] M. Grabisch. k-order additive discrete fuzzy measures and their representation. *Fuzzy Sets and Systems*, 92:167--189, 1997.

[13] M. Grabisch. The interaction and Möbius representation of fuzzy measures on finite spaces, k-additive measures: a survey. In M. Grabisch, T. Murofushi, and M. Sugeno, editors, *Fuzzy Measures and Integrals. Theory and Applications*, pages 70--93. Physica-Verlag, Heidelberg, 2000.

[14] M. Grabisch, I. Kojadinovic, and P. Meyer. A review of methods for capacity identification in Choquet integral based multi-attribute utility theory. *Europ. J. Operations Research*, 186:766--785, 2008.

[15] M. Grabisch and C. Labreuche. To be symmetric or asymmetric? A dilemma in decision making. In J. Fodor, B. De Baets, and P. Perny, editors, *Preferences and Decisions under Incomplete Knowledge*. Physica--Verlag, Heidelberg, 2000.

[16] M. Grabisch, J.-L. Marichal, and M. Roubens. Equivalent representations of set functions. *Mathematics of Oper. Research*, 25:157--178, 2000.

[17] J.-L. Marichal. On Choquet and Sugeno integrals as agregation functions. In M. Grabisch, T. Murofushi, and M. Sugeno, editors, *Fuzzy measures and integrals. Theory and Applications*, pages 247--272. Physica-Verlag, Heidelberg, 2000.

[18] J.-L. Marichal. Aggregation of interacting criteria by means of the discrete Choquet integral. In T. Calvo, G. Mayor, and R. Mesiar, editors, *Aggregation Operators. New Trends and Applications*, pages 224--244. Physica-Verlag, Heidelberg, New York, 2002.

[19] J.-L. Marichal. Tolerant or intolerant character of interacting criteria in aggregation by the Choquet integral. *Europ. J. Oper. Research*, 155:771--791, 2004.

[20] M. Sugeno. *Theory of Fuzzy Integrals and Applications*. PhD thesis, Tokyo Inst. of Technology, 1974.

[21] J.-Z. Wu and G. Beliakov. Nonadditivity index and capacity identification method in the context of multicriteria decision making. *Information Sciences*, 467:398--406, 2018.

[22] J.-Z. Wu and G. Beliakov. Marginal contribution representation of capacity based multicriteria decision making. *Int Journal of Intelligent Systems*, in press, doi:10.1002/int.22209, 2019.

[23] J.-Z. Wu and G. Beliakov. Nonmodularity index for capacity identifying with multiple decision criteria. *Information Sciences*, 492:164--180, 2019.

[24] J.-Z. Wu and G. Beliakov. Probabilistic bipartition interaction index of multiple decision criteria associated with the nonadditivity of fuzzy measures. *International Journal Intelligent Systems*, 34(2):247--270, 2019.