# Package WOWA
# for calculating weighted OWA functions and extending bivariate means.
# Version 1.0
# User Manual

Gleb Beliakov
gleb@deakin.edu.au

## License agreement

# Contents


**1 Summary** **5**

**2 Theoretical background** **7**
2.1 Means . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
2.2 Multivariate extension of bivariate means . . . . . . . . . . . 9
2.3 Binary tree construction . . . . . . . . . . . . . . . . . . . . 10
2.4 Weighted Ordered Weighted Averaging (WOWA) . . . . . . . 12
2.4.1 Weighted OWA approach by Torra . . . . . . . . . . . 13
2.4.2 Interpolation of the RIM quantifier function . . . . . . 14
2.4.3 $n$-ary tree construction for OWA by Dujmovic and Beliakov . . . . . . . . . . . . . . . . . . . . . . . . 15
2.4.4 Implicit WOWA . . . . . . . . . . . . . . . . . . . . . 18
2.5 Methods implemented in wowa library . . . . . . . . . . . . . 20

**3 Description of the library** **21**
3.1 Installation . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
3.2 Programming interface . . . . . . . . . . . . . . . . . . . . . 21
3.3 Description of the functions in `WOWA` . . . . . . . . . . . . . 21
3.3.1 wowa.OWA . . . . . . . . . . . . . . . . . . . . . . . . 21
3.3.2 wowa.ImplicitWOWA . . . . . . . . . . . . . . . . . . 22
3.3.3 wowa.WAn . . . . . . . . . . . . . . . . . . . . . . . . 22
3.3.4 wowa.WOWATree . . . . . . . . . . . . . . . . . . . . 24
3.3.5 wowa.weightedOWAQuantifier . . . . . . . . . . . . . 24
3.3.6 wowa.weightedOWAQuantifierBuild . . . . . . . . . . 25
3.3.7 wowa.WAM . . . . . . . . . . . . . . . . . . . . . . . . 26
3.4 Examples . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27
3.5 Where to get help . . . . . . . . . . . . . . . . . . . . . . . . 28

# Chapter 1

# Summary

This manual describes the package `WOWA` , which provides various tools for calculating the Weighted Ordered Weighted Averaging (WOWA) functions.

Chapter 2 provides some background on weighted means and OWA functions. In particular it details the binary tree extension of any bivariate mean by Beliakov and Dujmovic, as well as three different approaches to adding weighs to Yager's OWA functions. A more detailed overview can be found in [7, 8] and references therein. The description of the package `WOWA` and its functions is given in Chapter 3. Examples of its usage are provided in Section 3.4.

To cite `WOWA` package, use references [2–8].

# Chapter 2

# Theoretical background

## 2.1 Means

Aggregation functions play an important role in many applications including decision making, fuzzy systems and image processing [4, 8]. Averaging functions, aka mean, whose prototypical examples are the arithmetic mean and the median, allow compensation between low values of some inputs and high values of the others.

We consider weighting vectors $\mathbf{w}$ such that $w_i \geq 0$ and $\sum w_i = 1$ of appropriate dimensions.

**Definition 1** *For a given generating function $g : \mathbb{I} \to [-\infty, \infty]$, and a weighting vector $\mathbf{w}$, the weighted quasi-arithmetic mean is the function*

$$M_{\mathbf{w},g}(\mathbf{x}) = g^{-1}\left(\sum_{i=1}^{n} w_i g(x_i)\right). \tag{2.1}$$

**Definition 2** *Let $\varphi : \mathbb{I} \to \mathbb{I}$* be a bijection. The *$\varphi$-**transform** of a function $f : \mathbb{I}^n \to \mathbb{I}$ is the function $f_\varphi(\mathbf{x}) = \varphi^{-1}\left(f\left(\varphi(x_1), \varphi(x_2), ..., \varphi(x_n)\right)\right)$.*

The weighted QAM is a $\varphi$-transform of the weighted arithmetic mean with $\varphi = g$. The weighted arithmetic mean is therefore expression (2.1) with $g = Id$. Many means belong to the class of QAM (harmonic, geometric, quadratic, power means), but not all.

**Definition 3** *Let $n = 2$, $x, y > 0$, $x \neq y$ and $p \in [-\infty, \infty]$. The generalized logarithmic mean is the function*

$$L^p(x,y) = \begin{cases} \frac{y-x}{\log y - \log x}, & \text{if } p = -1, \\ \frac{1}{e}\left(\frac{y^y}{x^x}\right)^{1/(y-x)}, & \text{if } p = 0, \\ \min(x,y), & \text{if } p = -\infty, \\ \max(x,y), & \text{if } p = \infty, \\ \left(\frac{y^{p+1}-x^{p+1}}{(p+1)(y-x)}\right)^{1/p} & \text{otherwise.} \end{cases} \tag{2.2}$$

*For $x = y$ it is $L^p(x,x) = x$.*

The generalized logarithmic mean is symmetric. The function $L^0(x,y)$ is called the *identric* mean $\mathcal{I}$; $L^{-2}(x,y) = \mathcal{G}(x,y)$, the geometric mean $\mathcal{G}$; $L^{-1}$ is called the logarithmic mean $\mathcal{L}$; $L^{-1/2}$ is the power mean with $p = -1/2$; $L^1$ is the arithmetic mean $\mathcal{A}$. Only $L^{-1/2}$, $L^{-2}$ and $L^1$ are the quasi-arithmetic means.

**Definition 4** *Let us take two differentiable functions $g, h : \mathbb{I} \to \mathbb{R}$ such that $g' \neq 0$ and $\frac{g'}{h'}$ is invertible. Then the Cauchy mean is given for $x \neq y$ by*

$$C^{g,h}(x,y) = \left(\frac{g'}{h'}\right)^{-1}\left(\frac{g(x)-g(y)}{h(x)-h(y)}\right). \tag{2.3}$$

*For $x = y$, we set $C^{g,h}(x,x) = x$.*

The Cauchy means are continuous, symmetric and strictly increasing. The special case of $h(t) = t$ is called the Lagrangean mean $L^g$ . The generalized logarithmic means are Lagrangean means $L^g$ with $g(t) = t^{p+1}, p \notin \{-1, 0\}$, $g(t) = \log(t)$ for $p = -1$, and $g(t) = t \log t$ for $p = 0$. The Cauchy mean $C^{g,h}$ is a $\varphi$-transform of the Lagrangean mean $L^{g \circ h^{-1}}$ with $\varphi = h$.

Some Lagrangean (resp. Cauchy) means are quasi-arithmetic means (e.g., the arithmetic and geometric means), but some are not. For instance the harmonic mean is not Lagrangean, and the logarithmic mean is not quasi-arithmetic. The Lagrangean mean generated by $g(t) = t^{p+1}$ is called Stolarsky mean. The Cauchy mean generated by two power functions $g(t) = t^p, h(t) = t^s$ is called the extended mean (sometimes also referred to as Stolarsky mean). For more details about these means refer to [4].

Another bivariate mean which has attracted some attention recently is the Heronian mean. In the bivariate case it is defined as follows.

**Definition 5** *The Heronian mean is the function*

$$Her(x, y) = \frac{x + y + \sqrt{xy}}{3}. \tag{2.4}$$

Note that the Heronian mean can be written as

$$Her(x, y) = \frac{\mathcal{G}(x, y) + 2\mathcal{A}(x, y)}{3}. \tag{2.5}$$

The notation $\mathbf{x}_{\searrow}$ denotes the vector obtained from $\mathbf{x}$ by arranging its components in *non-increasing* order $x_{(1)} \geq x_{(2)} \geq \ldots \geq x_{(n)}$.

**Definition 6 (OWA)** *For a given weighting vector* $\mathbf{w}$, $w_i \geq 0$, $\sum w_i = 1$, *the OWA function is given by*

$$OWA_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{n} w_i x_{(i)} =< \mathbf{w}, \mathbf{x}_{\searrow} > . \tag{2.6}$$

Calculation of the value of an OWA function involves using a `sort()` operation.

## 2.2 Multivariate extension of bivariate means

The major class of averaging functions is the class of weighted quasi-arithmetic means (QAM). These functions are well studied and are convenient to work with as they have a natural definition for any number of arguments. Yet there are many other means, that often generalize quasi-arithmetic means, which are defined with respect to two arguments only, and do not offer a straightforward multivariate extension. A basic example here is the logarithmic mean [4]

$$\mathcal{L}(x, y) = \frac{x - y}{\ln x - \ln y},$$

which belongs to a rather broad class of Cauchy means. Cauchy means are defined with respect to two differentiable generating functions $g$ and $h$ such that $h' \neq 0$, by the use the Cauchy mean value theorem. In turn, Cauchy means have several prominent subclasses, such as the Lagrangean mean, the generalized logarithmic mean, Stolarsky means, and also quasi-arithmetic means. Frequently used members include the already mentioned logarithmic means, the identric mean and the Stolarsky means.

Another class of bivariate means are the neo-Pythagorean means, which are defined in terms of ratios between the inputs and the outputs. Here again, no obvious multivariate extension is present. One particular case is the Heronian mean.

In this contribution we use one generic approach for extending the bivariate means and incorporating weighting vectors based on repetitive application of the given bivariate function, reported in [6, 10]. It does not require knowledge of the properties of the bivariate function or its alternative representations, and is not based on an analytic formula but on an efficient computational procedure. The approach we present here is a recursive application of the bivariate function by constructing a binary tree with a suitable number of levels, where at each node the bivariate function is applied to the arguments of the child nodes. By using the idempotency of the means, we prune this tree to design a computationally efficient procedure. On the other hand, we are able to incorporate the weighting vectors by repeating the arguments as needed, following the approach of Calvo, Mesiar and Yager [9].

Our binary tree approach is generic in terms of the starting bivariate idempotent function being used, but it is not exact, in the sense that it is aimed at *approximating* a weighted multivariate mean (with any desired accuracy). Indeed, the binary tree will not reproduce exactly the weighting vectors with *irrational* coefficients, or coefficients that do not have finite binary representation (e.g., $\mathbf{w} = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$), in a finite number of iterations. We argue, however, that for computational purposes this is not inferior than even the explicit formulas: after all, all weighting vectors have finite binary representation in machine arithmetic, which we can match *exactly.*

## 2.3 Binary tree construction

We want to construct a weighted $n$-variate idempotent function $f_n$ with the weighting vector $\mathbf{w}$, by using only an *unweighted bivariate idempotent function* $f$ [6, 10]. To introduce the weights we use the approach from [9] where each argument $x_i$ is replicated a suitable number of times. That is, we consider an auxiliary vector of arguments $\mathbf{X} = (x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_n, \ldots, x_n)$, so that $x_1$ is taken $k_1$ times, $x_2$ is taken $k_2$ times and so on, and $\frac{k_1}{2^L} \approx w_1$, $\frac{k_2}{2^L} \approx w_2$, ..., and $\sum k_i = 2^L$, where $L \geq 1$ is a specified number of levels of the binary tree shown in Figure 2.1. One way of doing so is to take $k_i = \lfloor w_i 2^L + \frac{1}{2} \rfloor$, $i = 1, \ldots, n-1$ and $k_n = 2^L - k_1 - k_2 - \ldots - k_{n-1}$.

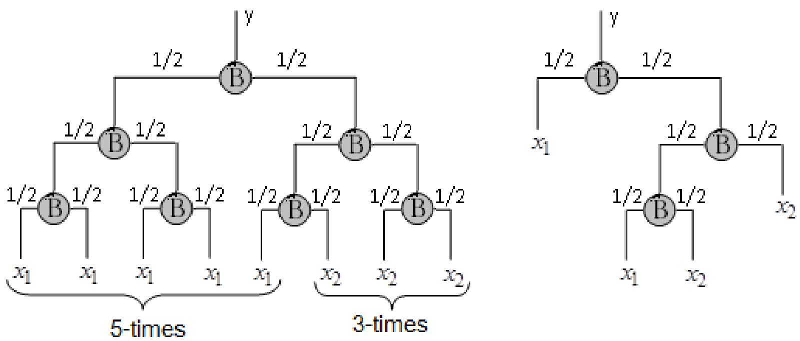


Figure 2.1: Representation of a weighted arithmetic mean in a binary tree construction. The tree on the right is pruned by using idempotency.

Next, let us build a binary tree presented in Figure 2.1, where at each node a value is produced by aggregating the values of two children nodes with the given bivariate symmetric idempotent function $f$ (denoted by $B$ on the plot). We start from the leaves of the tree which contain the elements of the vector $\mathbf{X}$. The value $y$ at the root node will be the desired output of the $n$-variate weighted function.

A straightforward binary tree traversal algorithm for doing so, which starts from the vector $\mathbf{X}$ computed as before, is as follows:

**Aggregation by Levels (ABL) Algorithm**

1. $N := 2^L$;
2. Repeat $L$ times:
   (a) $N := N/2$;
   (b) For $i := 1 \ldots N$ do $X[i] := f(X[2i-1], X[2i])$;
3. return $X[1]$.

Note that the bivariate function $f$ is assumed to be symmetric (the weights of the bivariate mean are symbolically denoted by $\frac{1}{2}$ as shown in Figure 2.1, although some symmetric means such as Logarithmic and Heronian do not have visible weights). The algorithm is obviously terminating. The runtime of the ABL algorithm is $O(2^L)$.

One practical disadvantage of the ABL algorithm is that its computational complexity is $O(2^L)$ in terms of the number of invocations of $f$. However it is possible to appropriately prune the binary tree by relying on idempotency of $f$, see Figures 2.1, 2.2. Indeed no invocation of $f$ is necessary if both of its arguments are equal. Such a pruning was presented in [6]. Below we present a general algorithm for the $n$-variate case whose (worst case) complexity is $O(L(n-1))$. This complexity is the lower bound, as at each level of the binary tree one can get at most $n-1$ nodes with different values of the child nodes, so that pruning is impossible and $f$ must be executed. The first $m$ levels of the binary tree have less than $n-1$ nodes each, the total number of nodes for these $m$ levels is $\sum_{k=1}^{m} 2^{m-1} = 2^m - 1$ nodes, hence $f$ can be called at most $2^m - 1 + (L-m)(n-1)$ times, where $m = \lfloor \log_2(n) \rfloor$.

The algorithm is recursive depth-first traversal of the binary tree. A branch is pruned if it is clear that all its leaves have exactly the same value, and by idempotency this is the value of the root node of that branch. The complexity of this algorithm is linear in $L$ and $n$.

A key property of the binary tree construction is the following.

**Theorem 1 (The Inheritance Theorem)** *[6] The multivariate extension $f_n$ of a bivariate idempotent function $f$ by the ABL algorithm preserves the intrinsic properties of the parent function $f$ as follows:*

1. *$f_n$ is idempotent since $f$ is idempotent;*
2. *if $f$ is monotone increasing then $f_n$ is monotone increasing;*
3. *if $f$ is continuous then $f_n$ is continuous;*
4. *if $f$ is convex (resp. concave) then $f_n$ is convex (resp. concave);*
5. *if $f$ is homogeneous then $f_n$ is homogeneous;*
6. *if $f$ is shift-invariant then $f_n$ is shift-invariant;*
7. *if $f$ is averaging then $f_n$ is averaging;*
8. *$f_n$ has the same absorbing element as $f$ (if any).*

## 2.4 Weighted Ordered Weighted Averaging (WOWA)

The weights in weighted means and in OWA functions represent different things. In weighted means $w_i$ reflects the importance of the $i$-th input, whereas in the OWA, $w_i$ reflects the importance of the $i$-th largest input. We now list some proposed frameworks for incorporating both types of weighting schemes following [3].

### 2.4.1 Weighted OWA approach by Torra

In [11] Torra proposed a generalisation of both weighted means and OWA, called WOWA. This aggregation function has two sets of weights $\mathbf{w}$ and $\mathbf{p}$. Vector $\mathbf{p}$ plays the same role as the weighting vector in weighted means, and $\mathbf{w}$ plays the role of the weighting vector in OWA functions.

**Definition 7 (Weighted OWA)** *Let $\mathbf{w}, \mathbf{p}$ be two weighting vectors, $w_i, p_i \geq 0$, $\sum w_i = \sum p_i = 1$. The following function is called the Weighted OWA function*

$$WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = \sum_{i=1}^{n} u_i x_{(i)},$$

*where $x_{(i)}$ is the i-th largest component of $\mathbf{x}$, and the weights $u_i$ are defined as*

$$u_i = g\left(\sum_{j \in H_i} p_j\right) - g\left(\sum_{j \in H_{i-1}} p_j\right),$$

*where the set $H_i = \{j | x_j \geq x_i\}$ is the set of indices of the $i$ largest elements of $\mathbf{x}$, and $g$ is a monotone non-decreasing function with two properties:*

1. *$g(i/n) = \sum_{j \leq i} w_j, i = 0, \ldots, n$ (of course $g(0) = 0$);*
2. *$g$ is linear if the points $(i/n, \sum_{j \leq i} w_j)$ lie on a straight line.*

Thus computation of WOWA involves a very similar procedure as that of the OWA (i.e., sorting components of $\mathbf{x}$ and then computing their weighted sum), but the weights $u_i$ are defined by using both vectors $\mathbf{w}, \mathbf{p}$, a special monotone function $g$, and depend on the components of $\mathbf{x}$ as well. One can see WOWA as an OWA function with the weights $\mathbf{u}$.

In [11, 12], the weights were introduced through an auxiliary interpolation function. It allows one to operate with two weighting vectors, one vector $\mathbf{p}$ related to the inputs magnitude, another, $\mathbf{w}$, related to the inputs themselves.

Let us list some of the properties of the WOWA function.

- The weighting vector $\mathbf{u}$ satisfies $u_i \geq 0$, $\sum u_i = 1$.
- If $w_i = \frac{1}{n}$, then $WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = WAM_{\mathbf{p}}(\mathbf{x})$, the weighted arithmetic mean.
- If $p_i = \frac{1}{n}$, $WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = OWA_{\mathbf{w}}(\mathbf{x})$.

- The WOWA is an idempotent aggregation function.

As noted, the weights $\mathbf{u}$ also depend on the generating function $g$. This function can be chosen as a linear spline (i.e., a broken line interpolant), interpolating the points $(i/n, \sum_{j\leq i} w_j)$ (in which case it automatically becomes a linear function if these points are on a straight line), or as a monotone quadratic spline, as was suggested in [11, 12], see also [1] where Schumaker's quadratic spline algorithm was used, which automatically satisfies the straight line condition when needed.

### 2.4.2 Interpolation of the RIM quantifier function

Let us now consider an alternative approach based on interpolating a RIM quantifier function. Here we use a method from [9], in which the weights of a function are computed by repeating the inputs a suitable number of times. Consider an auxiliary vector of arguments $\mathbf{X} = (x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_n, \ldots, x_n)$, so that $x_1$ is taken $k_1$ times and $x_2$ is taken $k_2$ times, and so on, so that $\frac{k_1}{M} = p_1$, $\frac{k_2}{M} = p_2, \ldots$, and $k_1 + k_2 + \ldots + k_n = M$. We assume the weights $p_i$ are rational numbers, which is not a strong restriction if we look at a computer implementation of the method in finite precision arithmetics.

The approach from [9] consists in using the auxiliary vector $\mathbf{X}$ in a strongly idempotent symmetric function (such as OWA induced by a quantifier) whose output will be a weighted function of the inputs $\mathbf{x}$.

In the case of OWA, in order to apply it to a larger dimensional auxiliary input vector $\mathbf{X}$ we need to produce the weighting vector $\mathbf{w}$ of the corresponding dimension $M$, denoted here by $\mathbf{u}$. We apply a similar approach to Torra's construction, that is, we construct a generating function $g$ by interpolating the data $g(i/n) = \sum_{j\leq i} w_j, i = 0, \ldots, n$ and $g(0) = 0$, and the straight line condition, that is, using the two conditions in Definition 7. The latter is necessary to obtain the standard weighted mean in case all $w_i = \frac{1}{n}$.

Hence we can use a piecewise linear or piecewise quadratic interpolation as in [1, 11, 12] to construct the RIM quantifier $g$. Now, in a way that differs to Torra's approach, we calculate the weighted OWA as

$$WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = OWA_{\mathbf{u}}(\mathbf{X}) = \sum_{i=1}^{n} u_i X_{(i)},$$

where the weights $u_i$ are defined as

$$u_i = g\left(\frac{i}{M}\right) - g\left(\frac{i-1}{M}\right),\ i = 1, \ldots, M.$$

If we compare both methods based on the generating RIM quantifier function, we can see that both produce exactly the same WOWA function [3]. We conclude that Torra's formula for the weights of WOWA can be seen as an instance of the approach from [9] based on replicating the inputs, although it obviously predates that work.

### 2.4.3 $n$-ary tree construction for OWA by Dujmovic and Beliakov

We apply the method of incorporating weights into any symmetric function by using binary trees [6,10]. We already saw this method in Section 2.3.

To introduce the weights into a symmetric function we use the approach from [9], where each argument $x_i$ is replicated a suitable number of times. We consider an auxiliary vector of arguments $\mathbf{X} = (x_1, \ldots, x_1, x_2, \ldots, x_2)$, so that $x_1$ is taken $k_1$ times and $x_2$ is taken $k_2$ times, so that $\frac{k_1}{2^L} \approx p_1$, $\frac{k_2}{2^L} \approx p_2$, and $k_1 + k_2 = M = 2^L$. Here $M$ is a power of two and $L \geq 1$ is a specified number of levels of the binary tree. One way of doing so is to take $k_1 = \lfloor p_1 2^L + \frac{1}{2} \rfloor$ and $k_2 = 2^L - k_1$. The vector $\mathbf{X}$ needs to be sorted into increasing or decreasing order.

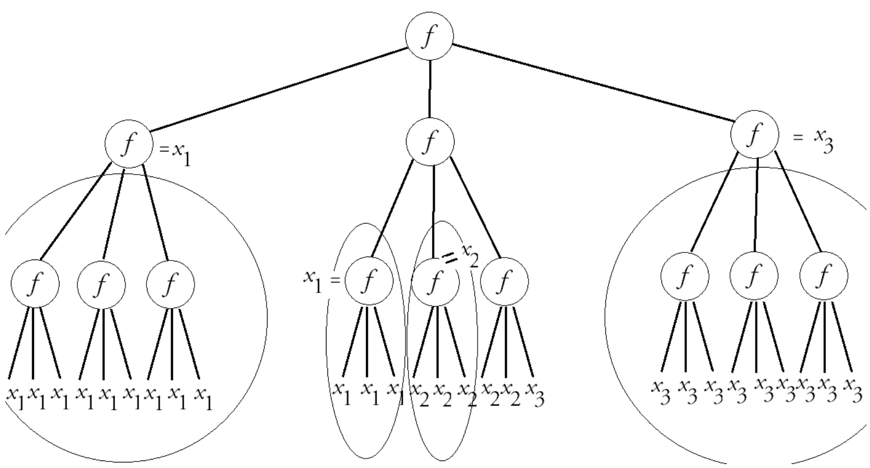


Figure 2.2: Representation of a weighted 3-variate mean in a binary tree construction. The tree on the right is pruned by using idempotency. The weights $\mathbf{w} = (\frac{1}{2}, \frac{3}{8}, \frac{1}{8})$ are matched exactly.

An efficient algorithm based on pruning the binary tree was presented in [6]. The pruning of the binary tree is done by using the idempotency of

$f$. No invocation of $f$ is necessary if both of its arguments are equal. A branch is pruned if it is clear that all its leaves have exactly the same value, and by idempotency this is the value of the root node of that branch. The algorithm is recursive depth-first traversing of the binary tree. The pruned tree algorithm has worst case complexity $O(L)$, which makes it practically applicable for large $L$.

The properties of the binary tree construction are listed in the Inheritance theorem 1 and include preservation of idempotency, monotonicity, continuity, convexity (concavity), homogeneity and shift-invariance, due to preservation of these properties in function composition. Furthermore, when the weights are given in a finite binary representation (as is always the case in machine arithmetic), the sequence of the outputs of the PTA algorithm with increasing $L = 2, 3, \ldots$,etc., converges to a weighted mean with the specified weights, and in fact $L$ needs not exceed the number of bits in the mantissa of the weights $p_i$ to match these weights exactly. When $f$ is a quasi-arithmetic mean, $f_p$ is a weighted quasi-arithmetic mean with the same generator.

We now extend the algorithm PTA to $n$-variate OWA functions following [2, 3]. Our goal here is to incorporate a vector $\mathbf{p}$ of non-negative weights (which add to one) into a symmetric $n$-variate function, by replicating the arguments a suitable number of times. As in the binary tree construction we build an $n$-ary tree with $L$ levels. As the base symmetric aggregator $f$ we take an OWA function $OWA_{\mathbf{w}}$ with specified weights $\mathbf{w}$ (although the origins of $f$ are not important for the algorithm).

Let us create an auxiliary vector $\mathbf{X} = (x_1, \ldots, x_1, x_2, \ldots, x_2, \ldots, x_n, \ldots, x_n)$, so that $x_1$ is taken $k_1$ times, $x_2$ is taken $k_2$ times, and so on, and $\frac{k_1}{n^L} \approx p_1$, $\frac{k_2}{n^L} \approx p_2$, …, and $\sum k_i = n^L$, where $L \geq 1$ is a specified number of levels of the tree. One way of doing so is to take $k_i = \lfloor p_i n^L + \frac{1}{n} \rfloor$, $i = 1, \ldots, n-1$ and $k_n = n^L - k_1 - k_2 - \ldots - k_{n-1}$.

The algorithm PnTA works in the same way as the PTA algorithm for binary trees. The function $f$ is executed only when some of its arguments are distinct, and since the elements of $X$ are ordered, there are at most $n-1$ such possibilities at each level of the tree, hence the complexity of the algorithm is $O((n-1)L)$.

Note that the complexity is linear in terms of $L$, as that of the PTA algorithm, which means that the dimension of the base aggregator $f$ does not matter in this respect. Of course, nominally the $n$-ary tree is larger than the binary tree, but since we only track the multiplicities of the arguments, never creating the array $\mathbf{X}$ explicitly, memorywise the complexity of the

PnTA algorithm is the same as that of PTA.

**Pruned $n$-Tree Aggregation (PnTA) Algorithm**
function $node(n, m, N, K, x)$

1. If $N[K] \geq n^m$ then do:
   (a) $N[K] := N[K] - n^m$;
   (b) $y := x[K]$;
   (c) If $N[K] = 0$ then $K := K + 1$;
   (d) return $y$;

   else

2. for $i := 1, \ldots, n$ do

   $z[i] := node(n, m-1, N, K, x)$

3. return $f(\mathbf{z})$.

---

function $f\_n(n, x, p, L)$

1. create the array $N := (k_1, k_2, \ldots, k_n)$ by using
   $k_i := \lfloor p_i n^L + \frac{1}{n} \rfloor$, $i = 1, \ldots, n-1$, and $k_n := n^L - k_1 - \ldots - k_{n-1}$;
2. $K := 1$;
3. return $node(n, L, N, K, x)$.

The vector $\mathbf{X}$ needs to be sorted, which is equivalent to sorting the inputs $\mathbf{x}$ jointly with the multiplicities of the inputs $N$ (i.e., using the components of $\mathbf{x}$ as the key), so the complexity of the sort operation is the same $O(n \log n)$ as for OWA functions.

We list some useful properties of the function $f_p$ generated by the PnTA algorithm established in [2]. They mimic those in Theorem 1.

**Theorem 2 (The Inheritance Theorem)** *The weighted extension $f_p$ of a function $f$ by the PnTA algorithm preserves the intrinsic properties of the parent function $f$ as follows:*

1. *$f_p$ idempotent since $f$ is idempotent;*
2. *if $f$ is monotone increasing then $f_p$ is monotone increasing;*
3. *if $f$ is continuous then $f_p$ is continuous;*

4. *if $f$ is convex (resp. concave) then $f_p$ is convex (resp. concave);*

5. *if $f$ is homogeneous then $f_p$ is homogeneous;*

6. *if $f$ is shift-invariant then $f_p$ is shift-invariant;*

7. *$f_p$ has the same absorbing element as $f$ (if any);*

8. *if $f$ generates $f_p$ then a $\varphi$-transform of $f$ generates the correspponting $\varphi$-transform of $f_p$.*

The next results are applicable when an OWA function is taken as the base aggregator $f$.

**Theorem 3** *Let $f = OWA_w$. Then the algorithm PnTA generates the weighted function $f_p$ which is the discrete Choquet integral (and is hence homogeneous and shift-invariant).*

### 2.4.4 Implicit WOWA

A different approach to introducing weights into averaging functions was recently presented in [5] under the name of implicit averaging. Here, following an analogy with weighted arithmetic means, which can be written in this form

$$y \cdot \frac{\sum_{j=1}^{n} p_j}{n} = \frac{\sum_{i=1}^{n} p_i x_i}{n}, \tag{2.7}$$

with $y = WAM(x_1, \ldots, x_n)$, we use the following equation to compute the values of a weighted function $f_p$ from a symmetric mean $M$,

$$C(M(p_1, \ldots, p_n), f_p(\mathbf{x})) = M(C(p_1, x_1), \ldots, C(p_n, x_n)). \tag{2.8}$$

Here $M$ is a mean and $C$ is a suitable bivariate operation such as a t-norm. The motivation behind the study in [5] is to produce alternative ways of incorporating weights $\mathbf{p}$ by replacing the product with another suitable operation and replacing the arithmetic mean with an arbitrary mean $M$. The function $f_p$ is given implicitly through solution to the algebraic equation (2.8). This equation can be written in a compact form as

$$C(\bar{p}, \bar{x}_p) = \overline{C(p_i, x_i)},$$

where $\bar{p}$ denotes the (unweighted) average weight, $\overline{C(p_i, x_i)}$ denotes the average value of $C$ and $y = \bar{f}_p(\mathbf{x}) = \bar{x}_p$ is the weighted average of $x_i$.

The work [5] established a number of useful theoretical properties of the implicit averages, which also apply to the case of $M = OWA$. Instantiating Equation (2.8) with $C$ being the product and $M$ being an OWA function with weights $\mathbf{w}$, we can resolve it explicitly and obtain

$$WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = \frac{OWA_w(p_1x_1,\ldots,p_nx_n)}{OWA_w(p_1,\ldots,p_n)} = \frac{OWA_w(\mathbf{px})}{OWA_w(\mathbf{p})}. \tag{2.9}$$

Note that the weights $p_i$ are otherwise unrestricted (i.e., they need not add to one) as the denominator in (2.9) will produce the required normalising factor to ensure idempotency.

Special care should be taken when the denominator vanishes, as this WOWA may be discontinuous or not well defined if we allow 0 weights. For strictly positive weighting vectors $\mathbf{w}$ the proposed WOWA is well defined and continuous. It is a piecewise linear, increasing, idempotent and homogeneous function. However, unlike the other mentioned WOWA, this function *is not* a discrete Choquet integral.

It is not difficult to see that the special case of equal weights $p_i = \frac{1}{n}$ corresponds to the unweighted OWA and $w_i = \frac{1}{n}$ corresponds to the WAM. However reversing the weights of OWA does not produce the dual of the original function. The implicit WOWA in (2.9) is a valid alternative to the existing WOWA where the arguments are weighted both according their position and magnitude.

**Example 1** *Consider the Hurwitz operator $M = H(\mathbf{x}) = a\max(\mathbf{x}) + (1-a)\min(\mathbf{x})$, $a \in ]0,1]$ and $C = \prod$. We have*

$$WOWA_{(a,1-a)}(\mathbf{x}) = \frac{a\max(\mathbf{px}) + (1-a)\min(\mathbf{px})}{a\max(\mathbf{p}) + (1-a)\min(\mathbf{p})}.$$

*Let us consider a particular case where $\mathbf{x} = (x_1, x_2)$, $a = \frac{9}{10}$, $\mathbf{p} = (p_1, p_2) = (\frac{1}{3}, \frac{2}{3})$, then*

$$WOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = \frac{\frac{9}{10}\max(\frac{1}{3}x_1, \frac{2}{3}x_2) + \frac{1}{10}\min(\frac{1}{3}x_1, \frac{2}{3}x_2)}{\frac{9}{10}\max(\frac{1}{3}, \frac{2}{3}) + \frac{1}{10}\min(\frac{1}{3}, \frac{2}{3})} = \frac{9\max(x_1, 2x_2) + \min(x_1, 2x_2)}{19}.$$

If we replace the products $p_ix_i$ with a more general function $C$, strictly increasing on $\mathbb{R}_{++} \times ]0,1]$, we obtain a generalisation of the WOWA function.

**Example 2** *When $C$ is replaced by the square of the product function in Example 1, we have*

$$GenWOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = \left[\frac{\frac{9}{10}\max(\frac{1}{9}x_1^2, \frac{4}{9}x_2^2) + \frac{1}{10}\min(\frac{1}{9}x_1^2, \frac{4}{9}x_2^2)}{\frac{9}{10}\max(\frac{1}{9}, \frac{4}{9}) + \frac{1}{10}\min(\frac{1}{9}, \frac{4}{9})}\right]^{1/2} = \left[\frac{9\max(x_1^2, 4x_2^2) + \min(x_1^2, 4x_2^2)}{37}\right]^{1/2}.$$

*Therefore,*

$$GenWOWA_{\mathbf{w},\mathbf{p}}(\mathbf{x}) = \left[\frac{9\max(x_1^2, 4x_2^2) + \min(x_1^2, 4x_2^2)}{37}\right]^{1/2}.$$

*Unlike Example 1, $GenWOWA$ is not piecewise linear, yet it is strictly increasing and idempotent.*

## 2.5 Methods implemented in wowa library

This R library implements the following methods:

1. The weighted arithmetic mean function WAM. Section 3.3.7
2. The ordered weighted averaging function OWA. Section 3.3.1
3. The Implicit WOWA by Beliakov and Calvo ImplicitOWA. Section 3.3.2
4. The RIM quantifier based WOWA (same as Torra's approach) WeightedOWAQuantifier. Section 3.3.5
5. The PnTA tree-based WOWA by Beliakov and Dujmovic WOWATree. Section 3.3.4
6. The binary tree weighted extension of bivariate means WAn. Section 3.3.3

In particular, the WAn method allows one to calculate the multivariate weighted extensions of the Logaritmic, Cauchy, Lagrangean, Heronian and other means with no obvious extensions, by providing the bivariate mean coded in R. There are several examples illustrating this technique.

As implied by the name of the library three distinct weighted OWA methods are also provided.

# Chapter 3

# Description of the library

## 3.1 Installation

Installation of `WOWA` package can be done from CRAN by following the usual package install process. Installaction can also be done (Linux, OSX) by `R CMD INSTALL wowa.tar.gz`, or an equivalent method for Windows. The `wowa.tar.gz` contains the necessary files and will be expanded into a suitable directory.

## 3.2 Programming interface

The subroutines in `WOWA` are implemented in C++ language. They reside in the file `wowa.cpp`. The files `RcppExports.cpp` and `wowawrapper.cpp` provide wrapper functions between R and C++.

## 3.3 Description of the functions in `WOWA`

### 3.3.1 wowa.OWA

**Ordered weigted average function.**

Function for computing the ordered weigted averages Formula (2.6). See Section 2.4.1 and Definition 7

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

| Argument | Description |
|---|---|
| n | Dimension of the array x |
| x | Input array |
| w | The OWA weights array |

**Example**

```
n <- 4
OWA <- wowa.OWA(n, c(0.3,0.4,0.8,0.2), c(0.4,0.35,0.2,0.05))
```

### 3.3.2 wowa.ImplicitWOWA

**Impicit Weighted OWA Computation Function.**

Function for calculating Implicit Weighted OWA function presented in Section 2.4.4

```
wowa.ImplicitWOWA(x, p, w, n,)
```

| Argument | Description |
|---|---|
| x | Input array |
| p | The weights array of input x. it should be non-negative. |
| w | The OWA weights array |
| n | Dimension of the array x, p and w |

**Example**

```
n <- 4
example <- wowa.ImplicitWOWA(c(0.3,0.4,0.8,0.2), c(0.3,0.25,0.3,0.15),
c(0.4,0.35,0.2,0.05), n)
```

### 3.3.3 wowa.WAn

**Extension of binary averaging.**
Function for calculating a binary tree multivariate extension of a binary averaging function in section 2.4.3

```
wowa.WAn(x, w, n, Fn, L)
```

| Argument | Description |
|---|---|
| x | Input array |
| w | The OWA weights array |
| n | Dimension of the array x and w |
| Fn | Bivariate symmetric mean that is extended to n arguments |
| L | The number of levels of the binary tree (see docs) |

**Example**

Extending the bivariate arithmetic mean function to 4 arguments with weights

```
Fn <- function(x, y) {
out <- (x+y)/2
return(out)
}
n <- 4
example <- wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn,
10)
```

Now extending the Heronian mean to 4 arguments with weights

```
Fn <- function(x, y) {
out <- (x+y+sqrt(x*y))/3
return(out)
}
n <- 4
example <- wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn,
10)
```

Now extending the Logarithmic mean function to 4 arguments with weights

```
Fn <- function(x, y) {
if(x==y) out=x
else if(x>0 & y>0)
out <- (y-x)/(log(y) - log(x))
else out<-0;
return(out)
}
n <- 4
example <- wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn,
10)
```

### 3.3.4 wowa.WOWATree

**Ordered weigted averages fuzzy system.**
Function for order weigted averages presented in Section 2.4.3

```
wowa.WOWATree(x, p, w, n, Fn, L)
```

| Argument | Description |
|---|---|
| x | Input array |
| p | The weights array of input x. |
| w | The OWA weights array |
| n | Dimension of the array x, p and w |
| Fn | Base n-variate symmetric function defined in R |
| L | The number of levels of the binary tree (see docs) |

**Example**

```
 Fn <- function(n, x, w) {
out <- 0.0
for(i in 1:n){ out <- out + x[i]*w[i] }
return(out)
}

n <- 4
example <- wowa.WOWATree(c(0.3,0.4,0.8,0.2), c(0.3,0.25,0.3,0.15),
c(0.4,0.35,0.2,0.05), n, Fn, 10)
```

### 3.3.5 wowa.weightedOWAQuantifier

**WOWA value computation.**
Function for calculating the value of the quantifier-based WOWA function.
See section 2.4.2

```
wowa.weightedOWAQuantifier(x, p, w, n, spl)
```

| Argument | Description |
|---|---|
| x | Input array |
| p | The weights array of input x |
| w | The OWA weights array |
| n | The dimension of the arrays x, p and w |
| spl | Structure that keeps the spline knots and coefficients |

**Example**

```
n <- 4
x <- c(0.3,0.4,0.8,0.2)
pweights <- c(0.3,0.25,0.3,0.15)
wweights <- c(0.4,0.35,0.2,0.05)
```

Create last argument called spl using weightedOWAQuantifierBuild function

```
tempspline <- wowa.weightedOWAQuantifierBuild(pweights, wweights,
n)
example <- wowa.weightedOWAQuantifier(x, pweights , wweights, n,
tempspline)
```

### 3.3.6 wowa.weightedOWAQuantifierBuild

**RIM quantifier of the Weighted OWA Computation function.**
Function for Building the RIM quantifier of the Weighted OWA

```
wowa.weightedOWAQuantifierBuild(p, w, n)
```

| Argument | Description |
|---|---|
| p | The weights array of input x |
| w | The OWA weights array |
| n | Dimension of the array x and w |

**Example**

```
n <- 4
x <- c(0.3,0.4,0.8,0.2)
pweights <- c(0.3,0.25,0.3,0.15)
wweights <- c(0.4,0.35,0.2,0.05)
```

weightedOWAQuantifierBuild function creates last argument called spl for the weightedOWAQuantifier function

```
tempspline <- wowa.weightedOWAQuantifierBuild(pweights, wweights,
n)
example <- wowa.weightedOWAQuantifier(x, pweights , wweights, n,
tempspline)
```

### 3.3.7 wowa.WAM

**WAM computation function.**
Function for calculating the Weighted Arithmetic Mean function. See Definition 1 and 2

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

| Argument | Description |
|---|---|
| n | Dimension of the array x and w |
| x | Input array |
| w | The OWA weights array |

**Example**

```
n <- 4
wowa.WAM(n, c(0.3,0.4,0.8,0.2), c(0.4,0.35,0.2,0.05))
```

## 3.4 Examples

**OWA** Section 3.3.1

```
n = 4
OWA <- wowa.OWA(n, c(0.3,0.4,0.8,0.2), c(0.4,0.35,0.2,0.05))
```

**ImplicitWOWA** Section 3.3.2

```
wowa.ImplicitWOWA(c(0.3,0.4,0.8,0.2), c(0.3,0.25,0.3,0.15), c(0.4,0.35,0.2,0.05),
n)
```

**WAn** Section 3.3.3

Extending the bivariate arithmetic mean function to 4 arguments with weights (this is trivial, of course, just for the sake of familiar example)

```
Fn <- function(x, y)
out <- (x+y)/2
return(out)
n <- 4
wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn, 10)
```

Extending the Heronian mean to 4 arguments with weights (this is not trivial, there is no obvious extension)

```
Fn <- function(x, y)
out <- (x+y+sqrt(x*y))/3
return(out)

n <- 4
wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn, 10)
```

Extending the Logarithmic mean function to 4 arguments with weights

```
Fn <- function(x, y) {
if(x==y) out=x
else if(x>0 & y>0)
out <- (y-x)/(log(y) - log(x))
else out<-0;
return(out)
}

n <- 4
wowa.WAn(c(0.3,0.4,0.8,0.2), c(0.4,0.3,0.2,0.1), n, Fn, 10)
```

**WOWATree** Section 3.3.4

```
Fn <- function(n, x, w)
out <- 0.0
for(i in 1:n)
out <- out + x[i]*w[i]
return(out)
Fn(n,x,w)

n <- 4
wowa.WOWATree(c(0.3,0.4,0.8,0.2), c(0.3,0.25,0.3,0.15), c(0.4,0.35,0.2,0.05),
n, Fn, 10)
```

**weightedOWAQuantifier and weightedOWAQuantifierBuild** Section 3.3.5

```
x <- c(0.3,0.4,0.8,0.2)
pweights <- c(0.3,0.25,0.3,0.15)
wweights <- c(0.4,0.35,0.2,0.05)
```

create last argument called spl using weightedOWAQuantifierBuild function

```
spl <- wowa.weightedOWAQuantifierBuild(pweights, wweights, n)
wowa.weightedOWAQuantifier(x, pweights , wweights, n, spl)
```

**WAM** Section 3.3.7

```
WAM <- wowa.WAM(n, c(0.3,0.4,0.8,0.2), c(0.4,0.35,0.2,0.05))
```

## 3.5 Where to get help

The software library WOWA 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 WOWA 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 WOWA , bugs, suggestions, its usefulness, applying it in practice and so on.

If you want to cite WOWA package, use references [2–8].

# Bibliography


[1] G. Beliakov. Shape preserving splines in constructing WOWA operators: Comment on paper by V. Torra in Fuzzy Sets and Systems 113 (2000) 389-396. *Fuzzy Sets and Systems*, 121:549–550, 2001.

[2] G. Beliakov. A method of introducing weights into owa operators and other symmetric functions. In V. Kreinovich, editor, *Uncertainty Modeling. Dedicated to B. Kovalerchuk*. Springer, Berlin, Heidelberg, 2015.

[3] G. Beliakov. Comparing apples and oranges: the weighted OWA function. *International Journal of Intelligent Systems*, 33:1089–1108, 2018.

[4] G. Beliakov, H. Bustince, and T. Calvo. *A Practical Guide to Averaging Functions*. Springer, New York, 2016.

[5] G. Beliakov, T. Calvo, and P. Fuster. Implicit averaging functions. *Information Sciences*, 417:96–112, 2017.

[6] G. Beliakov and J.J. Dujmovic. Extension of bivariate means to weighted means of several arguments by using binary trees. *Information Sciences*, 331:137–147, 2016.

[7] G. Beliakov, S. James, and J.-Z. Wu. *Discrete Fuzzy Measures: Computational Aspects*. Springer, Berlin, Heidelberg, 2019.

[8] G. Beliakov, A. Pradera, and T. Calvo. *Aggregation Functions: A Guide for Practitioners*. Springer, Berlin, Heidelberg, 2007.

[9] T. Calvo, R. Mesiar, and R.R. Yager. Quantitative weights and aggregation. *IEEE Transactions on Fuzzy Systems*, 12:62–69, 2004.

[10] J.J. Dujmovic and G. Beliakov. Idempotent weighted aggregation based on binary aggregation trees. *International Journal of Intelligent Systems*, 32:31–50, 2017.

[11] V. Torra. The weighted OWA operator. *International Journal of Intelligent Systems*, 12:153–166, 1997.

[12] V. Torra. The WOWA operator and the interpolation function W*: Chen and Otto's interpolation revisited. *Fuzzy Sets and Systems*, 113:389–396, 2000.