Quick Guide
for the
Generalized Fuzzy Index Generator

Tullio Facchinetti
Agnese Marchini

February 24, 2009

Contents

Chapter 1

Introduction

This document briefly describes how to use the Java package of the Generalized Fuzzy Index Generator. We illustrate the features available to easily define custom ecological indexes.

Chapter 2

Index definition

2.1 Model description file

A model description file contains the references to the set of input variables and output functions which compose the index. Figure 2.1 shows an example of index model.

A model definition can include the following items:

- a set of one or more *input variables*, preceeded by the `[IN]` marker;
- one *output function*, preceeded by the `[OUT]` marker;
- optional comments preceeded by the # character.

The reference of an input variable definition is made by the following statement:

```
[IN] weight filename
```

where:

- the `[IN]` marker indicates that we are referencing a variable to be used as *input variable*;
- the *weight* is a real number which specify the relative weight (i.e., the importance) of the variable with respect to other variables;
- *filename* indicates the file containing the variable definition; it can optionally include a relative/absolute path.

NOTE: filenames **must not contain any space** (blank), otherwise it will not be possible to correctly load the variable. Therefore, text files containing the definition of variables, must have a name without spaces. Use underscores or traits instead. For example, the following filenames are correct:

```
variable1.txt
variable-1.txt
variable_1.txt
```

while the following ones are not:

```
variable 1.txt
var 1.txt
my variable.txt
```

The definition of input variables and ouput functions are provided within separate files. Such files are referenced in the model definition file.

The reference of output function is made by the following statement:

```
[OUT] filename
```

where:

- the `[OUT]` marker indicates that we are referencing a function to be used as *output function*;
- *filename* indicates the file containing the function definition; it can optionally include a relative/absolute path.

2.2 Variable description file

The variable definition file contains the definition of a single fuzzy variable, including the definition of all the membersip functions which compose the variable. Figure 2.2 shows an example of variable definition.

The variable definition file is divided into sub-sections introduced by specific tags:

- the tag `[variable]` introduces the data related to the variable
- the tags `[function]` introduce the data related to each single membership function

ATTENTION: tags are **case-sensitive**, so `[variable]` is ok while `[Variable]` or `[VARIABLE]` are not.

```
#
# Comments MUST have a '#' char as first character
# No empty lines are currently allowed nor supported
#
[IN]  1.0 ./___varH.txt
[IN]  1.0 ./___varW.txt
[IN]  1.0 ./___varRED.txt
[IN]  1.0 ./___varSAV.txt
[OUT] ___ecological_status.txt
```

Figure 2.1: Example of model definition.

2.2.1 Variable definition

The variable definition indicates the parameters which are specific of the variable.

The variable definition is introducted by the `[variable]` tag.

Currently, the only parameter supported is the *variable name*. Figure 2.3 shows an example of variable's parameter definition.

2.2.2 Membership functions definition

The membership function definition within a variable definition file contains the parameters referring to the membership function. Figure 2.4 shows an example of membership function's parameter definition.

The function is introducted by the `[function]` tag. After the `[function]` tag appear the function definition's parameters:

- the `name` is a string which specifies the function name (it is not necessarily unique);
- the `type` indicates the category type of the function; several different types are available; the complete list is described in Section **??**;
- the `points` are a list of co-ordinates which describe the membership function; the number of points depend on the function type;
- an optional `status` indicates whether the function represents the paradigmatic `worst` or `best` cases

```
[variable]
name = SAV

[function]
name = low
type = left_parabola
points = 0.0 1.0 0.5 0.5 1.0 0.0
status = worst

[function]
name = medium
type = center_parabola
points = 0.0 0.0 0.5 0.5 1.0 1.0 1.0 1.0 5.5 0.5 10.0 0.0

[function]
name = high
type = center_parabola
points = 1.0 0.0 5.5 0.5 10.0 1.0 10.0 1.0 55.0 0.5 100.0 0.0

[function]
name = excess
type = right_parabola
points = 10.0 0.0 55.0 0.5 100.0 1.0
status = best
```

Figure 2.2: Example of variable definition.

```
[variable]
name = SAV
```

Figure 2.3: Example of variable parameters.

```
[function]
name = low
type = left_parabola
points = 0.0 1.0 0.5 0.5 1.0 0.0
status = worst
```

Figure 2.4: Example of function parameters.

Notice that the `status` parameter accepts only one of the two values between `worst` and `best`.

ATTENTION: each variable must contain EXACTLY ONE membership function defined as `worst` AND EXACTLY ONE membership function defined as `best`.

2.3 Lineguides for the correct index definition

2.3.1 The output variable

The output variable is a normal fuzzy variable with its own membership functions. At the moment, however, to obtain a final index that ranges between 0 and 100, it is required for the output variable to be defined in the range $[0, 1]$, otherwise the index value will vary in a scaled range instead of $[0, 100]$.

2.3.2 Filenames

It is worth to recall that filenames **must not contain any space** (blank), otherwise it will not be possible to correctly load the variable. Therefore, text files containing the definition of variables, must have a name without spaces. Use underscores or traits instead. For example, the following filenames are correct:

```
variable1.txt
variable-1.txt
variable_1.txt
```

while the following ones are not:

```
variable 1.txt
```

```
var 1.txt
my variable.txt
```

Chapter 3

Membership functions

This section describes the available membership functions that can be used to build an index.

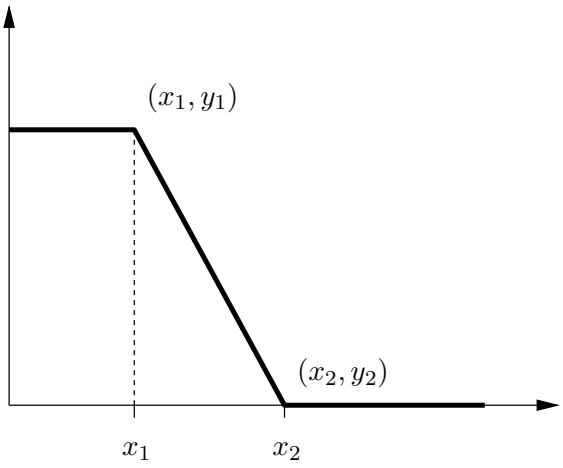
3.1 `left_trapezoid`

- `type = right_trapezoid`
- number of points: 2
- points: x_1 y_1 x_2 y_2
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_1 + \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) & \text{if } x_1 < x \leq x_2 \\ y_2 & \text{if } x_2 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 1.0$ and $y_2 = 0.0$



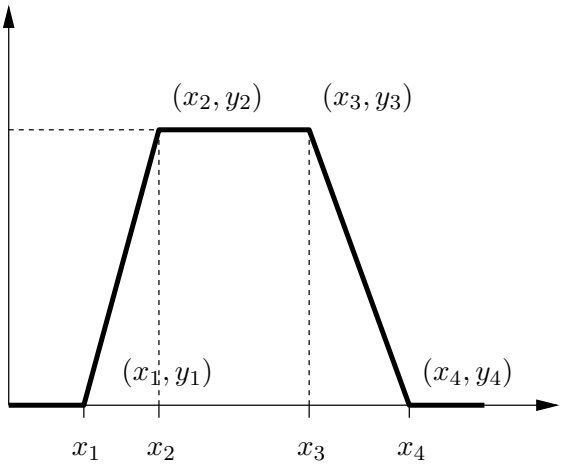
3.2 `center_trapezoid`

- `type = center_trapezoid`
- number of points: 4
- points: x_1 y_1 x_2 y_2 x_3 y_3 x_4 y_4
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_1 + \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) & \text{if } x_1 < x \leq x_2 \\ y_3 + \frac{y_2 - y_3}{x_2 - x_3}(x - x_3) & \text{if } x_2 < x \leq x_3 \\ y_3 + \frac{y_3 - y_4}{x_3 - x_4}(x - x_3) & \text{if } x_3 < x \leq x_4 \\ y_4 & \text{if } x_4 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$, $y_2 = 1.0$, $y_3 = 1.0$ and $y_4 = 0.0$



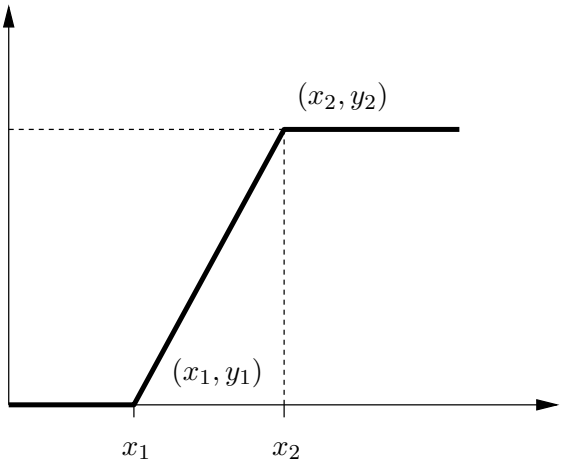
3.3 right_trapezoid

- `type = right_trapezoid`
- number of points: 2
- points: x_1 y_1 x_2 y_2
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_1 + \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) & \text{if } x_1 < x \leq x_2 \\ y_2 & \text{if } x_2 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$ and $y_2 = 1.0$



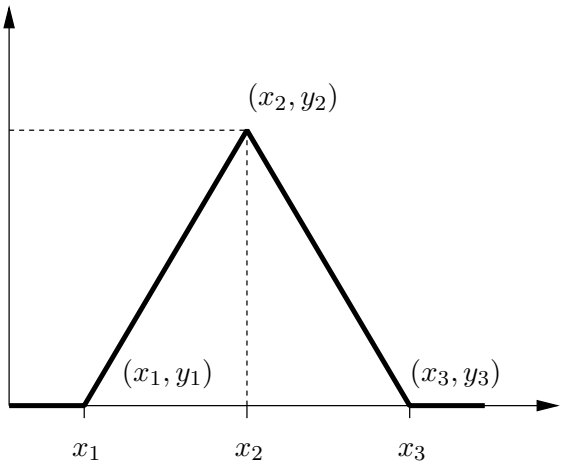
3.4 `triangle`

- `type = triangle`
- number of points: 3
- points: x_1 y_1 x_2 y_2 x_3 y_3
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_1 + \frac{y_1 - y_2}{x_1 - x_2}(x - x_1) & \text{if } x_1 < x \leq x_2 \\ y_3 + \frac{y_2 - y_3}{x_2 - x_3}(x - x_2) & \text{if } x_2 < x \leq x_3 \\ y_3 & \text{if } x_3 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$, $y_2 = 1.0$ and $y_3 = 0.0$



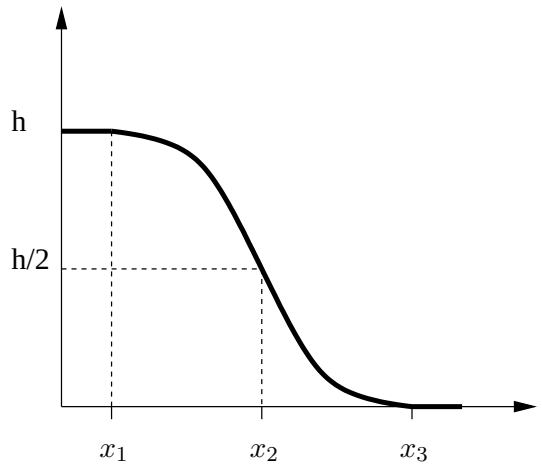
3.5 `left_parabola`

- `type = left_parabola`
- number of points: 3
- points: x_1 y_1 x_2 y_2 x_3 y_3
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ 1.0 - y_2 \frac{(x_1 - x)^2}{(x_1 - x_2)^2} & \text{if } x_1 < x \leq x_2 \\ y_2 \frac{(x - x_3)^2}{(x_2 - x_3)^2} & \text{if } x_2 < x \leq x_3 \\ y_3 & \text{if } x_3 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$, $y_2 = 0.5$, $y_3 = 1.0$, $y_4 = 1.0$, $y_5 = 0.5$, $y_6 = 0.0$



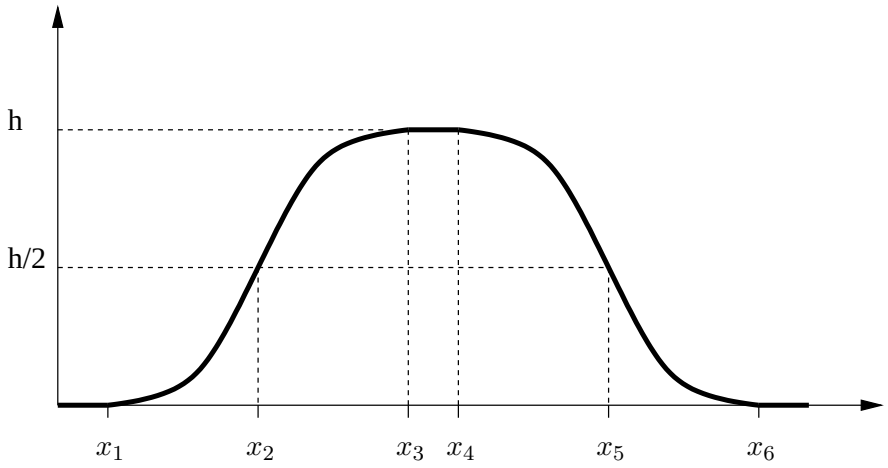
3.6 center_parabola

- `type = center_parabola`
- number of points: 6
- points: $x_1\ y_1\ x_2\ y_2\ x_3\ y_3\ x_4\ y_4\ x_5\ y_5\ x_6\ y_6$
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_2 \frac{(x_1-x)^2}{(x_1-x_2)^2} & \text{if } x_1 < x \leq x_2 \\ 1.0 - y_2 \frac{(x-x_3)^2}{(x_2-x_3)^2} & \text{if } x_2 < x \leq x_3 \\ y_4 & \text{if } x_3 < x \leq x_4 \\ 1.0 - y_5 \frac{(x_4-x)^2}{(x_4-x_5)^2} & \text{if } x_4 < x \leq x_5 \\ y_5 \frac{(x-x_6)^2}{(x_5-x_6)^2} & \text{if } x_5 < x \leq x_6 \\ y_6 & \text{if } x_6 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$, $y_2 = 0.5$, $y_3 = 1.0$, $y_4 = 1.0$, $y_5 = 0.5$, $y_6 = 0.0$



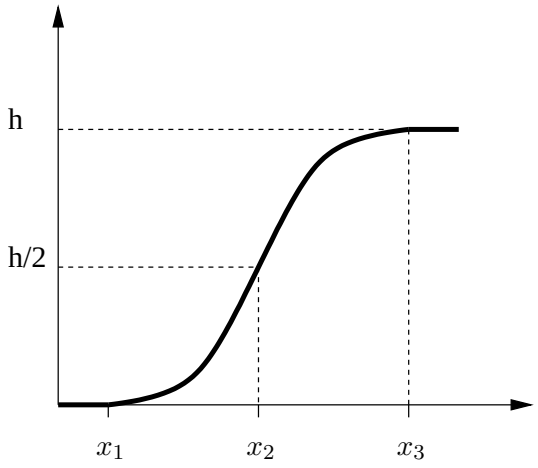
3.7 `right_parabola`

- `type = right_parabola`
- number of points: 3
- points: x_1 y_1 x_2 y_2 x_3 y_3
- corresponding function:

$$f(x) = \begin{cases} y_1 & \text{if } x \leq x_1 \\ y_2 \frac{(x_1-x)^2}{(x_1-x_2)^2} & \text{if } x_1 < x \leq x_2 \\ 1.0 - y_2 \frac{(x-x_3)^2}{(x_2-x_3)^2} & \text{if } x_2 < x \leq x_3 \\ y_3 & \text{if } x_3 < x \end{cases}$$

For the correct behavior of the index it must hold:

- $y_1 = 0.0$, $y_2 = 0.5$, $y_3 = 1.0$, $y_4 = 1.0$, $y_5 = 0.5$, $y_6 = 0.0$



3.8 crisp

A crisp variable behaves in a little different way with respect to "normal" variables. They are used when there is no way to express a variable with a quantitative measurement (i.e., a number).

Single functions represent different input classes. The user must supply the relative ranking among input classes. The supplied ranks become the weights used to build the fuzzy rules. As for other variable types, the worst case is associated with weight equal to zero, while the best case is associated with the highest weight. Other weights are scaled accordingly.

Figure 3.1 shows an example of crisp variable with 4 different classes (i.e. categories), where the input class "category 2" have a weight 5 times less than "category 4".

Notice that the worst function must receive a weigth equal to 0.

```
[variable]
name = CrispVariable

[function]
name = category 1
type = crisp
points = 0.0
status = worst

[function]
name = category 2
type = crisp
points = 0.2

[function]
name = category 3
type = crisp
points = 0.5

[function]
name = category 4
type = crisp
points = 1.0
status = best
```

Figure 3.1: Example of crisp variable.