DESARROLLO DE UNA LIBRERÍA INTEGRAL DE SOFTWARE PARA LA IMPLEMENTACIÓN DE TÉCNICAS DE LÓGICA DIFUSA

SUSANA CAROLINA RUGE SALAZAR

UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
BOGOTÁ D.C.
2004

DESARROLLO DE UNA LIBRERÍA INTEGRAL DE SOFTWARE PARA LA IMPLEMENTACIÓN DE TÉCNICAS DE LÓGICA DIFUSA

SUSANA CAROLINA RUGE SALAZAR

Director
Ing. Oscar Germán Duarte Velasco M.Sc. Ph.D.

Trabajo Presentado para Optar al Título
Magíster en Automatización Industrial

UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA
BOGOTÁ D.C.
2004

Nota de aceptación

Jurado

Jurado

Bogotá, D.C. (21 de julio de 2004)

A Constanza, por el apoyo y amor
incondicionales, siempre presente.

A mis abuelos, los padres.

A mi familia, por preguntar tanto.

A Alfonso, quien me mostró a Daneel.

A mis amigos, los de cerca y los de lejos, a
los que se les extraña como el primer día.

AGRADECIMIENTOS

La autora expresa sus agradecimientos a:

Gabriel Fernando Navas R. por la oportunidad de trabajar en horarios no convencionales.

Fredy S. Duque R. por la ayuda con la discusión de ideas para la programación.

CONTENIDO

pág.

LISTA DE TABLAS

pág.

LISTA DE FIGURAS

LISTA DE ANEXOS

LISTA DE CLASES

ABREVIATURAS

α-corte	Alfa-Corte
CD	Conjunto Difuso
fpert	*Función de Pertenencia*
GUI	Graphics User
SLD	Sistema de Lógica Difusa
TSK	Takagi-Sugeno-Kang
VL	*Variable Lingüística*

RESUMEN

En este documento se hace referencia a una librería desarrollada en C++, que surgió como una necesidad de unificación de técnicas de Lógica Difusa.

La clase para Conjuntos Difusos creada en esta librería ofrece un manejo dual de la caracterización del conjunto: como *Función de Pertenencia* y como α-cortes.

La librería también ofrece clases para realizar operaciones entre conjuntos difusos y aritmética difusa, además de Sistemas de Lógica Difusa (SLD) basados en reglas tipo Mamdani y Takagi-Sugeno-Kang (TSK).

This document is related to a library developed in C++ as an answer to necessity of unification of several Fuzzy Logic Techniques.

The Fuzzy Sets class created in this library is characterized by: membership function and α-cuts.

There are classes for operation between fuzzy sets a fuzzy arithmetic, besides Fuzzy Logic Systems rule-based Mamdani and TSK types.

Keywords: Lógica Difusa, librería, C++, Sistema de Lógica Difusa, wxWindows, Técnicas Difusas, *Función de Pertenencia*, α-cortes, reglas, Fuzzy Logic, α-cuts, membership function, Fuzzy Logic Systems, rules, Mamdani, TSK.

1. INTRODUCCIÓN

Debido a la manifiesta necesidad de una herramienta de software para manejo e implementación de técnicas difusas que tuviese características diferentes al software disponible en el mercado y que llenara ciertos requerimientos de desarrollos realizados en la Universidad Nacional, surgió la inquietud de la creación de una librería que se adaptara a estas condiciones.

La librería materia de la Presente Tesis, en adelante llamada Librería para Técnicas Difusas, fue diseñada como base y plataforma para permitir trabajos posteriores de ampliación de la misma, dentro de los que se podrían encontrar herramientas como redes neuronales y técnicas neuro-difusas.

Una de las principales características de la librería es la descripción dual de Conjuntos Difusos en *Función de Pertenencia* entendida como sucesión de puntos unidos por interpolación lineal y, *α-cortes*, que ofrece una mayor flexibilidad para el manejo de los Conjuntos Difusos.

Además de esto, la librería contiene operaciones sobre conjuntos difusos, aritmética difusa, agrupamiento y Sistemas de Lógica Difusa tipo Mamdani y TSK (Takagi-Sugeno-Kang).

1.1. Trabajos Preliminares Realizados en Software para Lógica Difusa

1.1.1. Trabajos Comerciales

En el Software comercial se encuentran herramientas para el manejo de Técnicas Difusas, sin embargo, dicho Software resulta muy costoso, poco modular, no es transportable a aplicaciones de la vida real ó no integra las diversas Técnicas Difusas. Dichas herramientas han sido creadas para un único sistema operativo cuyas consecuencias son poca flexibilidad y portabilidad casi cero.

A continuación se mencionarán las principales características de algunos de las herramientas comerciales.

1.1.1.1. Matlab

A pesar de ser una herramienta altamente difundida y probada, es de alto costo pues además de requerir la Toolbox de Lógica Difusa que tiene un costo aproximado de USD 200, también se necesita el programa básico cuyo costo aproximado es de USD 600. Funciona con reglas e inferencia tipo Mamdani, genera código en ANSI C y agrupamiento por dos métodos diferentes incluyendo el Fuzzy C-Means. Ofrece características de análisis y visualización además de otros tipos de inferencia. La información se puede encontrar en la página web http://www.mathworks.com.

1.1.1.2. Mathematica

La Toolbox de Lógica Difusa para Mathematica fue desarrollada por Wolfram. Presenta características de modelamiento, graficación, inferencia, variables lingüísticas, Fuzzy C-means y aritmética. Retorna los puntos superiores a un alfa-corte determinado, pero no trabaja por alfa-cortes. La información se puede encontrar en la página web http://www.wolfram.com/products/applications/fuzzylogic/

1.1.1.3. Fide

Esta herramienta es tiene características de modelamiento, simulación y análisis, además de propiedades de diseño CAD. Retorna algoritmos difusos en Java, ANSI-C y Matlab (.m). Trabaja Sistemas de Lógica Difusa tipo Mamadami. La información se puede encontrar en http://www.aptronix.com/index-fuzzy.htm

1.1.1.4. FuzzyCLIPS

Ofrece diferentes técnicas de inferencia y portabilidad en ANSI C. La información se puede encontrar en http://www.ortech-engr.com/fuzzy/fzyclips.html

1.1.1.5. Otras

Existen otras herramientas como el fuzzyTECH que presenta la generación de código en C y cuya versión extendida ofrece implementaciones neuro-difusas. Presenta la posibilidad de exportación a ActiveX y DLL.

Otros desarrollos son el XpertRule Knowledge Builder para construir inferencias tipo Mamdani y el Xiera que exporta código C++.

1.1.2. Trabajos Realizados en la Universidad Nacional

- **UNFUZZY - Software para el diseño, análisis, simulación e implementación de sistemas de lógica difusa**. Duarte V., Oscar G. “En este trabajo se presenta el programa UNFUZZY, desarrollado en la Universidad Nacional de Colombia. UNFUZZY es una herramienta de diseño, análisis, simulación e implementación de sistemas de Lógica Difusa de Múltiples entradas y Múltiples Salidas, (MIMO)”. Bogotá. 1997.
- **Metodología para determinar la factibilidad de implementación de la tecnología de banda ancha LMDS(local multipoint distribution system), en una organización.** Alvarado Forero, Mónica y otros. “En este proyecto se desarrolla una metodología que permite la comparación de las principales tecnologías de banda ancha que existen en el mercado y mediante técnicas de lógica y aritmética difusas se consideran los diferentes

parámetros que le permitirán decidir de acuerdo a su presupuesto y condiciones en el mercado la solución óptima, o por lo menos conocer si son aptas o no para la adopción de la nueva tecnología de telecomunicaciones inalámbricas como lo es LMDS (Local Multipoint Distribution System)". Bogotá. 2004.

- **Sistema experto basado computación flexible para la selección de parámetros de maquinado.** Arroyo Osorio, José Manuel y otros. "La selección de los parámetros para manufacturar componentes mediante el proceso de maquinado por torneado se realiza normalmente con base en la experiencia o usando información experimental publicada. En el proyecto se diseñó un prototipo de programa para seleccionar automáticamente los parámetros apropiados. Para la determinación de la geometría de la herramienta: los ángulos alfa n, gama n y lambda se establecen mediante sistemas de lógica difusa (SLD)". Bogotá. 2003.
- **Desarrollo de una librería genérica en C++ para el procesamiento de imágenes con lógica difusa.** Delgado Rangel, Leonardo Javier y otros. " La librería desarrollada para el procesamiento de imágenes con lógica difusa para la reducción de ruido tipo uniforme e impulsivo, preservando la estructura de una imagen y conservando sus características más importantes (bordes y detalle)". Bogotá. 2002.
- **Editor gráfico para el prototipo de un sistema experto como herramienta en la negociación de energía eléctrica en la bolsa de Santafé de Bogotá.** Cárdenas, Erasmo y otros. "La simulación la realiza apoyado en clases de lógica difusa generadas por la herramienta UNFUZZY empleada para hacer evaluaciones sucesivas de los datos que le son suministrados al sistema hasta obtener el precio de la energía que genera el agente en cada hora". Bogotá. 1999.
- **Implantación de un prototipo para control borroso.** Espejo Niño, Jairo Humberto y otros. "Análisis, diseño y construcción de una biblioteca en C++ para crear y manejar conjuntos difusos. Desarrollo de un sistema para la creación de controladores neuro-difusos basado en tres módulos (Editor, entrenador y simulador) ; y por último, el desarrollo de una metodología para la implementación de controladores neuro-difusos en la industria moderna". Bogotá. 1996.
- **Diseño de un sistema para el control del proceso de pasteurización de la planta piloto de leches del ICTA-UN.** Fernández Díaz, Olga Patricia y otros. "Este proyecto

presenta dos grandes aportes, el primero de ellos corresponde a la elaboración de un manual de operación y el segundo de ellos es el diseño de un sistema de control difuso para el proceso de pasteurización que actualmente se lleva a cabo en la planta piloto de leches del ICTA . Este sistema de control fue construido y calibrado en el programa UNFUZZY". Bogotá. 2002.

- **Metodología para al análisis de riesgo de daño por rayos en Colombia utilizando técnicas difusas.** Gallego Vega, Luis Eduardo y otros. "Se presenta una metodología para la estimación del riesgo de daño por rayos en Colombia mediante técnicas que permitan la valoración de incertidumbres y la combinación de variables linguisticas y numéricas propias de la subjetividad en la valoración de cualquier problema de riesgo. Se plantea el uso de lógica difusa para responder al problema de la aceptabilidad del riesgo". Bogotá. 2002.
- **Módulo de implementación de controladores difusos de entradas/salidas de un P.L.C.** Leiva Ramírez, Oscar Enrique y otros. "El objetivo de este trabajo es elaborar una herramienta capaz de ejecutar Algoritmos de Control Difuso, diseñados previamente con el software UNFUZZY; además de ejecutar labores de supervisión sobre el sistema controlado. Se creó una herramienta basada en el control difuso "FUZZYCOM", que permita la comunicación entre el Computador y el Controlador Lógico Programable SUCOS PS306 de marca "Klockner Moeller"". Bogotá. 1998.
- **Nuevas alternativas de secado mecánico de granos.** López Fontal, Elkin Mauricio. "Secado en lecho fluidizado utilizando sistemas de lógica difusa". Bogotá. 2002.
- **Control de un secador rotatorio con sistema de lógica difusa.** Mariño Martínez, Daniel y otros. Diseño de un módulo de control con sistema de razonamiento difuso implementado en Labview, para supervisar y controlar las principales variables que intervienen en un proceso de secado de material cristalino, en un secador rotatorio en contraflujo, aprovechando de una mejor forma los recursos técnicos dispuestos". Bogotá. 1999.
- **Diseño y control de un control difuso basado en micro-controladores.** Mariño Pérez, Jorge Enrique y otros. "Construcción de un controlador difuso utilizando para ello el micro-controlador 80C31 el cual pertenece a la familia del 8051 de Intel. La tarjeta construida y el software que la acompaña han sido bautizados UN-8031. El sistema de

lógica difusa diseñado puede ser implementado en la tarjeta UN-8031 a través del puerto serial o a través del comando Generar Código del menú simulación".

- **Sistema experto para negociación de energía en sistemas hidrotérmicos.** Martínez Forero, Julián Arturo y otros. "El sistema experto para la negociación de energía en sistemas hidrotérmicos (SEPNEHT), es un software diseñado para proporcionar a los agentes generadores una herramienta que les facilite el análisis del mercado de energía mayorista, con el cual puedan tomar una decisión acerca del precio de oferta de la energía que generan por medio de sistemas de lógica difusa". Bogotá. 2002.
- **Modelamiento evolutivo en sistemas de lógica difusa.** Molano Pulido, Luz Yenny y otros. "En este proyecto se desarrolló una herramienta de software para la optimización de sistemas de lógica difusa a través de la evolución de uno o varios de sus componentes utilizando algoritmos genéticos; las bases del trabajo son las aplicaciones UN-Genético y UN-Fuzzy". Bogotá. 2002.
- **Modelo de PSS y AVR basado en lógica difusa para un generador sincrónico en sistemas de potencia.** Robayo Almanza, Raul Iván y otros. "Este trabajo propone un modelo de control para el sistema de excitación de la máquina sincrónica, el cuál consta fundamentalmente de dos dispositivos esenciales. Para el modelamiento de estos dispositivos se ha utilizado lógica difusa". Bogotá. 1999.
- **Segmentación de imágenes usando algoritmos fuzzy.** Rojas Camacho, Oswaldo y otros. "Los modelos de la teoría de lógica difusa y conjuntos difusos se aplican los campos de reconocimiento de patrones y procesamiento de imágenes; entre estos tenemos: funciones de membresía, agrupamiento difuso, sistemas basados en reglas difusas, medidas difusas, integral difusa y entropía difusa". Bogotá. 1999.
- **Control difuso de posición para un motor de corriente continua.** Roncancio F., Luis Felipe y otros. "En este estudio se muestran y analizan las simulaciones y los resultados experimentales de la implementación de un sistema de control difuso de posición para un motor de corriente continua y su comparación con un sistema de control de posición utilizando un compensador P.I. clásico." Bogotá. 1995.
- **Métodos y Modelos de Programación Lineal Difusa.** Báez Sánchez, Andrés David. Bogotá. 2003.

1.2. Justificación para la creación de una Nueva Herramienta

Dados los costos de las herramientas comerciales para Lógica Difusa con valores superiores a $1.500.000 para cada licencia, para La Universidad era de la mayor importancia tener a su disposición una herramienta cuyo uso no estuviese limitado por las licencias comerciales y que brindara la flexibilidad necesaria para los numerosos desarrollos en Computación Flexible que se están trabajando en este momento y que se proyectan hacia el futuro dentro de los cuales se encuentran los mencionados en la sección 1.1.2.

Por esta razón se planteó la necesidad de desarrollar una herramienta que se ajustase a dichos requerimientos con el propósito implementarse en los proyectos adelantados en La Universidad. Entre otros requisitos, se encontraba la modularidad para extender la herramienta en posteriores desarrollos a otras ramas de la computación flexible como computación neuro-difusa y desarrollo de una interfaz gráfica amigable.

1.3. Características de la Librería

La *Librería para Técnicas Difusas* fue desarrollada como una herramienta integral de software de Computación Flexible, esperando que para trabajos posteriores se ofrezca como base para la integración de otras herramientas. El objetivo básico de la librería es el manejo de Conjuntos Difusos, Aritmética Difusa, Operaciones con números difusos, Agrupamiento y Sistemas de Lógica Difusa tipo Mamdani y TSK (Takagi-Sugeno-Kang). Se habían realizado desarrollos independientes previos en la Universidad para cada uno de estos ítems, pero esta es la primera vez que un solo desarrollo los integra a todos, además de permitir su expansibilidad hacia nuevas integraciones.

Uno de los principales aportes es la concepción de los conjuntos difusos con una representación dual mediante *Función de Pertenencia* y α-cortes que facilita el manejo y cálculos entre conjuntos, y ofrece una modularidad antes no presentada en las herramientas

desarrolladas en la Universidad y no ofrecida por casi ninguna de las herramientas comerciales, con los costos que ellas implican.

Por estar escrita en C++ permite la agregación de nuevos módulos mediante la adición de código y herencia de clases. Ha sido creada como una librería de libre distribución para permitir que cualquier persona dentro o fuera de la Universidad la incluya en sus propios desarrollos, y así mismo todas las potencialidades que puede ofrecer la creación de por ejemplo ActiveX y DLLs.

La librería además tiene la funcionalidad de poder compilarse en diferentes plataformas.

Al presente documento se adjunta un CD que contiene el código fuente de la librería desarrollada.

1.3.1. Por qué se empleó C++

Se escogió C++ como lenguaje de programación por estar Orientado a Objetos, lo que permite al programador diseñar aplicaciones con un código estructurado y la reutilizabilidad del código en una forma más lógica y productiva. Se puede compilar el código en prácticamente cualquier computador y sistema operativo sin realizar ningún cambio drástico. El código escrito en C++ es más corto en comparación con otros lenguajes. La compilación de Código en C++ es muy eficiente debido a su dualidad de lenguaje Alto-Bajo Nivel, además de ser el lenguaje más difundido del mundo.

1.3.2. Librería wxWindows

wxWindows es un marco GUI (*Graphics User Interface*) de código abierto para programar aplicaciones funcionales en diferentes plataformas.

Contiene un grupo de bibliotecas que se pueden compilar y correr bajo diferentes tipos de computador. Existe una biblioteca para cada GUI. Dentro de las plataformas soportadas se encuentran:

- Windows 3.1, Windows 95/98, Windows NT, Windows 2000, Windows ME
- Linux y otras plataformas Unix con GTK+
- Unix con Motif u otros clones Lesstif gratis de Motif
- Mac OS
- Un puerto OS/2 se encuentra en progreso.

También ofrece un conjunto de facilidades de programación que permiten funcionalidad en la programación. Lo que facilita la codificación sin grandes esfuerzos en la programación.

Se descarga de forma gratuita de la página http://www.wxwindows.org. Se instala según las instrucciones ofrecidas en dicha página y de acuerdo con el compilador con que se vaya a trabajar. Es indispensable instalarlo antes de compilar la librería. Se encuentra bien documentado y bien establecido. Además de ofrecer una interfaz amigable para el usuario, tiene facilidades para los diálogos de pantalla y el manejo de memoria, arreglos, sistemas de archivos, etc.

Dentro de los objetivos iniciales del proyecto se encontraba el permitir en futuros trabajos el desarrollo de una interfaz gráfica amigable multi-plataforma, por ello se empleó la librería wxWindows porque facilita esta implementación gráfica, es de libre distribución y además posee herramientas prácticas multi-plataforma para manejo de archivos y arreglos.

2. MAPA CONCEPTUAL

De las diferentes técnicas difusas se implementaron:

- Sistemas Basados en Reglas
- Lógica y Aritmética Difusas
- Agrupamiento Difuso

En la Figura 2.1. se observan las interrelaciones entre las técnicas difusas implementadas. En los capítulos posteriores se mostrarán detalladamente los componentes de cada una de ellas.

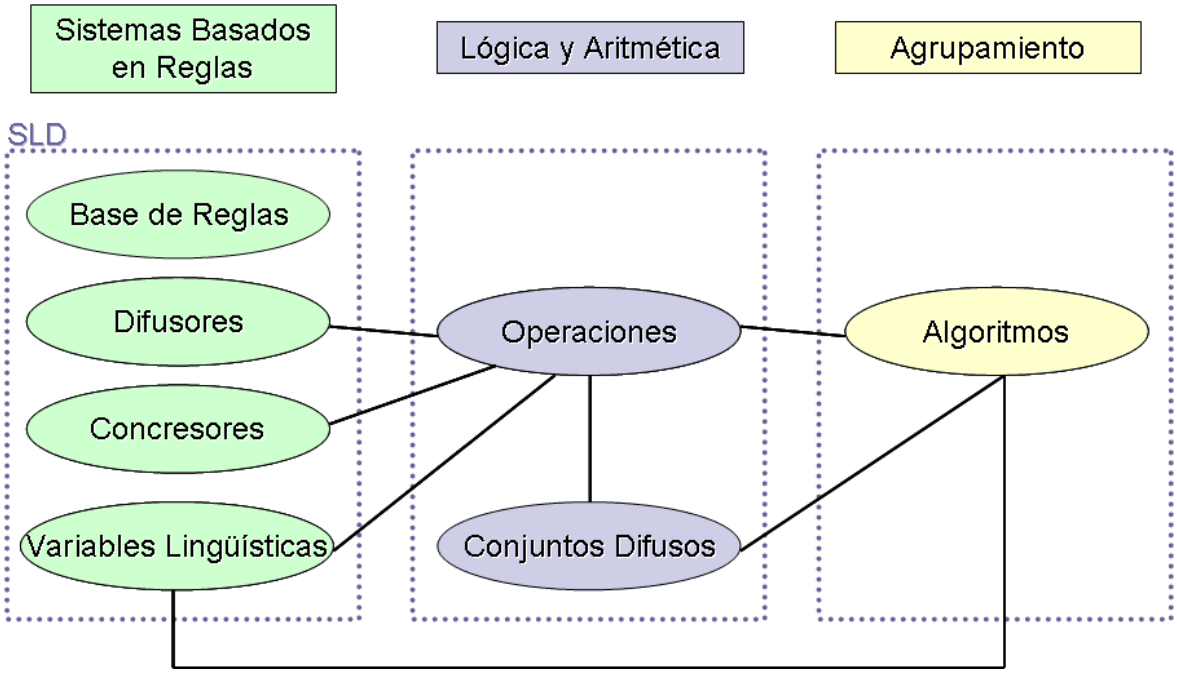


Figura 2.1. Esquema del Conjunto Difuso

3. DESCRIPCIÓN GENERAL DE LA LIBRERÍA

En este capítulo se muestran los principales componentes de la librería y sus operaciones.

3.1. Conjuntos Difusos

La clase para los Conjuntos Difusos descrita en la sección 4.3.1.1 contiene la definición de *Conjunto Difuso*. En la Figura 3.1. se muestran los componentes de los conjuntos difusos. La Figura 3.2 ilustra las partes principales de un *Conjunto Difuso*: Nombre, *Universo*: $[x_{ini},x_{fin}]$ (cotas inicial y final del universo del Conjunto Difuso), α-cortes (cada uno de nivel α_i), *Función de Pertenencia*.

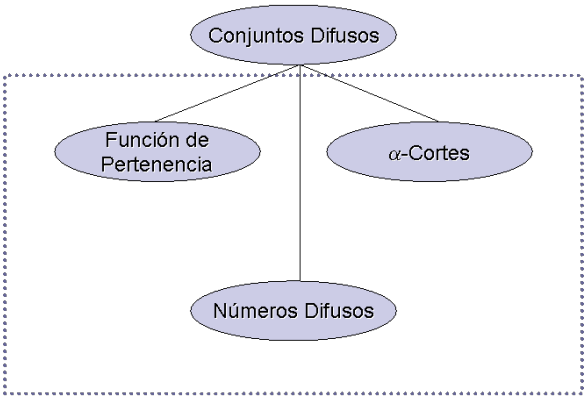


Figura 3.1. Componentes de los Conjuntos Difusos

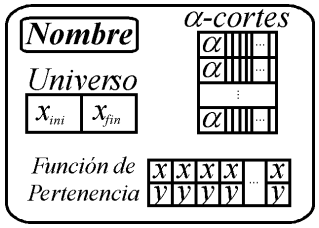


Figura 3.2. Esquema del Conjunto Difuso

Aunque la *Función de Pertenencia* y los *α-cortes* son dos formas diferentes de representar los *Conjuntos Difusos* se ha implementado una caracterización dual redundante para los conjuntos difusos en la *Librería para Técnicas Difusas* pues en ciertas ocasiones es más fácil efectuar cálculos mediante la *Función de Pertenencia*, pero en otras resulta más sencillo mediante *α-cortes*. Esta caracterización dual redundante ha permitido reducir y facilitar las operaciones de cálculo. Dado que garantizar la redundancia puede ser complejo se crearon funciones que garantizan que la información redundante sea coherente.

3.2. Operaciones Difusas

Se definieron cinco tipos de operaciones difusas diferentes: las operaciones *uniarias* que operan sobre un Conjunto Difuso y retornan un Conjunto Difuso, las operaciones *binarias* operan sobre dos Conjuntos Difusos y retornan un Conjunto Difuso, las operaciones que operan sobre un arreglo de Conjuntos Difusos y que retornan *valores*, *otras* operaciones que retornan valores diferentes, intervalos, etc, y *ordenamiento* de arreglos de Conjuntos Difusos. Las operaciones se encuentran descritas en la sección 4.3.1.2.

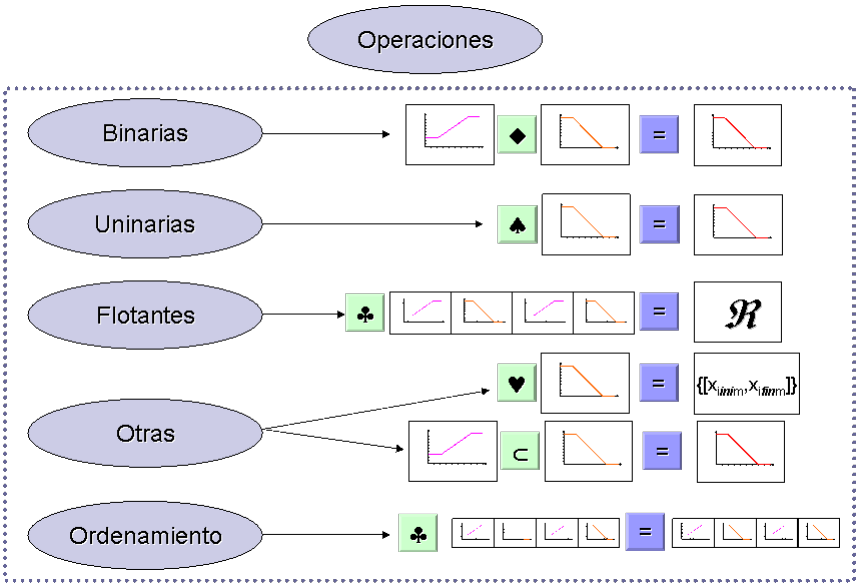


Figura 3.3. Esquema de Operaciones Difusas

3.2.1. Operaciones Uniarias

Se definió una clase genérica llamada **`modificador`** de la que se heredan todas las clases que calculan operaciones uniarias entre *Conjuntos Difusos*. Las clases para estas operaciones se encuentran en la sección 4.4.1. Estas operan sobre un conjunto difuso y su resultado es un conjunto difuso.

Figura 3.4. Operaciones Uniarias

La Tabla 3.1 muestra las Operaciones Uniarias implementadas.

Nombre	Operación
Complemento	$\overline{X} = \int_U \frac{1-\mu_X(u)}{u}$
Muy	$muy(X) = \int_U \frac{(\mu_X(u))^2}{u}$
Más o Menos	$mas\_o\_menos(X) = \int_U \frac{(\mu(X))^{\frac{1}{2}}}{u}$
Más	$mas(X) = \int_U \frac{(\mu(X))^{1.25}}{u}$
Algo	$a\lg o(X) = \int_U \frac{(\mu(X))^{\frac{1}{3}}}{u}$
Extremadamente	$extremadamente(X) = \int_U \frac{(\mu(X))^3}{u}$
No muy	$no\_muy(X) = \int_U \frac{1-(\mu_X(u))^2}{u}$
Multiplicador	$factor(X) = \int_U \frac{(\mu(X)) * factor}{u}$
Normalizar	$normalizar(X) = \int_U \begin{cases} 1 & \mu_X(u) \geq altura(X) * 0.8 \\ 0 & \mu_X(u) \leq altura(X) * 0.2 \end{cases}$

Tabla 3.1. Operaciones Difusas Uniarias

3.2.2. Operaciones Binarias

Se definió una clase genérica llamada **`operacion_difusa`** de la que se heredan todas las clases que calculan operaciones binarias y retornan un *Conjunto Difuso*. Las clases para estas operaciones se encuentran en la sección 4.4.2. Estas operan sobre dos conjuntos difusos y retornan un Conjunto Difuso.

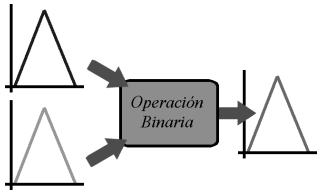


Figura 3.5. Operaciones Binarias

3.2.2.1. Operaciones Binarias calculadas por Función de Pertenencia

Para efectuar los cálculos en estas operaciones se almacenan en un arreglo los puntos de las *Funciones de Pertenencia* de ambos conjuntos y se agregan además, los puntos en donde se intersectan las *Funciones de Pertenencia* de estos, como muestra la Figura 3.6.

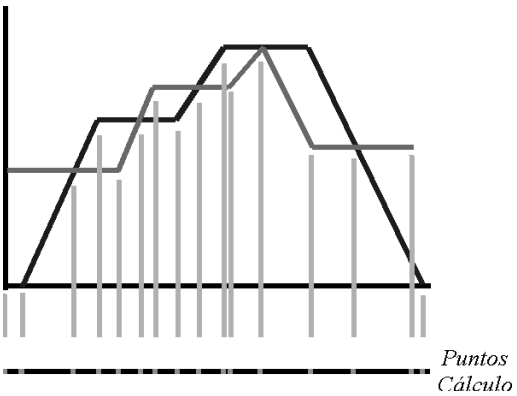


Figura 3.6. Puntos de Cálculo para las Operaciones Binarias

La Tabla 3.2. muestra las Operaciones Binarias implementadas.

Nombre	Operación
Mínimo	$\sum_{i=1}^{j} \min(y_1(x_i), y_2(x_i))$

Máximo	$\sum_{i=1}^{j} \max(y_1(x_i), y_2(x_i))$

Tabla 3.2. Operaciones Difusas Binarias Calculadas por Función de Pertenencia

3.2.2.2. Operaciones Binarias calculadas por α-cortes

Para efectuar los cálculos en estas operaciones se almacenan en una matriz los límites de los intervalos de los *α-cortes* de ambos conjuntos y se agregan además, los *α-cortes* en donde se intersectan las Funciones de Pertenencia de estos, como muestra la Figura 3.7.

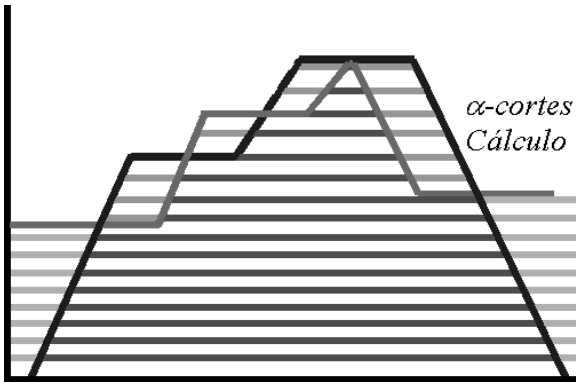


Figura 3.7. α-cortes de Cálculo para las Operaciones Binarias

La Tabla 3.3. muestra las Operaciones Binarias implementadas.

Nombre	**Operación**
Suma	$\left[x_{1\alpha A+B}, x_{2\alpha A+B}\right] = \left[x_{1\alpha A} + x_{1\alpha B}, x_{2\alpha A} + x_{2\alpha B}\right]$
Resta	$\left[x_{1\alpha A-B}, x_{2\alpha A-B}\right] = \left[x_{1\alpha A} - x_{2\alpha B}, x_{2\alpha A} - x_{1\alpha B}\right]$
Multiplicación	$\left[x_{1\alpha A-B}, x_{2\alpha A-B}\right] = \begin{bmatrix} \min(x_{1\alpha A} * x_{1\alpha B}, x_{1\alpha A} * x_{2\alpha B}, x_{2\alpha A} * x_{1\alpha B}, x_{2\alpha A} * x_{2\alpha B}), \\ \max(x_{1\alpha A} * x_{1\alpha B}, x_{1\alpha A} * x_{2\alpha B}, x_{2\alpha A} * x_{1\alpha B}, x_{2\alpha A} * x_{2\alpha B}) \end{bmatrix}$
Mínimo Intervalar	$\left[x_{1\alpha \min(A,B)}, x_{2\alpha \min(A,B)}\right] = [\min(x_{1\alpha A}, x_{1\alpha B}), \min(x_{2\alpha A}, x_{2\alpha B})]$
Máximo Intervalar	$\left[x_{1\alpha \max(A,B)}, x_{2\alpha \max(A,B)}\right] = [\max(x_{1\alpha A}, x_{1\alpha B}), \max(x_{2\alpha A}, x_{2\alpha B})]$

Tabla 3.3. Operaciones Difusas Binarias Calculadas por α-cortes

3.2.3. Operaciones que retornan un valor Flotante

Se definió una clase genérica llamada **`valores`** de la que se heredan todas las clases que operan sobre un Arreglo de Conjuntos Difusos y calculan operaciones que retornan un valor *double*. Las clases para estas operaciones se encuentran en la sección 4.4.3.

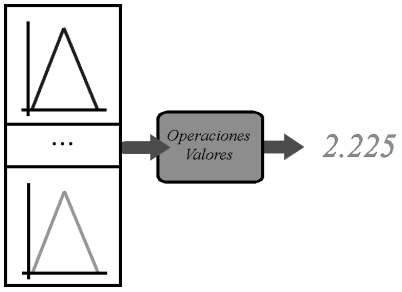


Figura 3.8. Operaciones que retornan un valor flotante

La Tabla 3.4 muestra las Operaciones Uniarias implementadas.

Nombre	Operación
Consistencia	$consistencia(ArregloCD) = H(\min(ArregloCD))$
Cardinalidad	$M(A) = \int_i \mu A(u_i)$
Subconjuntez	$S(B,A) = \frac{M(\min(A,B))}{M(B)}$
Altura	$H(A) = \max(Funcion\ de\ Pertenencia(A))$
Distancia	$d(A,B) = \sqrt{\int_U [\mu_A(u) - \mu_B(u)]du}$
P-Distancia	$p(A,B) = \left[\int_U [\mu_A(u) - \mu_B(u)]^p\right]^{\frac{1}{p}}$
Distancia por *Valor Representativo*	$d_{VR}(A,B) = VR(A) - VR(B)$

Tabla 3.4.Operaciones Difusas que Retornan Valores

3.2.4. Otras Operaciones

Se diseñó para aquellas operaciones que no se ajustan a ninguno de los formatos de operaciones anteriores. Pueden retornar diversos valores. Las clases para estas operaciones se encuentran en la sección 4.4.4.

La Tabla 3.5. muestra las Operaciones Uniarias implementadas.

Nombre	**Operación**	**Retorna**
Soporte	$\alpha(0)$	Intervalos
Subconjunto	$B \subset A$	Booleano

Tabla 3.5. Otras Operaciones Difusas

3.2.5. Ordenamiento

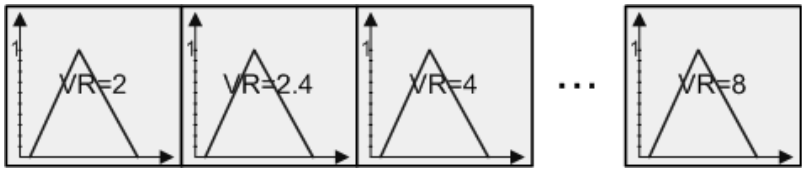


Figura 3.9. Ordenamiento por *Valor Representativo*

Se implementó el ordenamiento en los conjuntos difusos calculando para cada uno el *Valor Representativo* calculado según la ecuación:

$$V_N(\beta) = \frac{\int_0^1 \alpha^r v_N(\alpha,\beta) d\alpha}{\int_0^1 \alpha^r d\alpha} \tag{3.1}$$

donde

$$v_N(\alpha,\beta) = (1-\beta)[a + \alpha(b-a)] + \beta[c + \alpha(d-c)] \tag{3.2}$$

Siendo *(a,b,c,d)* las coordenadas de un trapecio típico.

3.3. Variables Lingüísticas

Una *Variable Lingüística* es una variable que se representa con palabras. Está compuesta por varios *Conjuntos Difusos*. La clase que define las *Variables Lingüísticas* se encuentra en la sección 4.3.1.2. La Figura 3.10 ilustra el esquema de las *Variables Lingüísticas*. Cada

Variable Lingüística está compuesta por un *Nombre* y *un Arreglo de Conjuntos Difusos* (*Términos de Variable*).

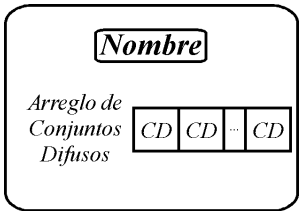


Figura 3.10. Esquema de la *Variable Lingüística*

3.4. Concresores

Los concresores se emplean para hallar un valor que represente a un Conjunto Difuso. Se ha implementado la clase genérica **`concresor`** cuyo cálculo retorna un valor *double*. Los concresores implementados se pueden encontrar en la sección 4.5.2.

Sea $\alpha(max)$ el α-corte mayor, compuesto por k intervalos así: $\sum_{i=1}^{k}\left[x_{ini\ i}, x_{fin\ i}\right]$ los concresores implementados se calculan como indica la Tabla 3.6.

Nombre	Operación
Primer Máximo	$PM = x_{ini\ 1}$, *para* $\alpha(\max)$
Último Máximo	$UM = x_{fin\ k}$, *para* $\alpha(\max)$
Media de Máximos	$MM = x_{ini\ 1} + x_{fin\ k}$, *para* $\alpha(\max)$
Centro de Gravedad	$CG = \frac{\int_U y \cdot \mu(y)dy}{\int_U \mu(y)dy}$

Tabla 3.6. Concresores Implementados

3.5. Difusores

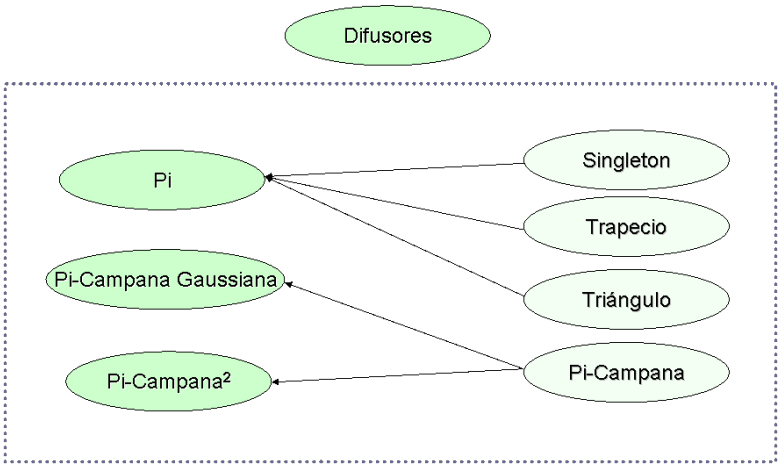
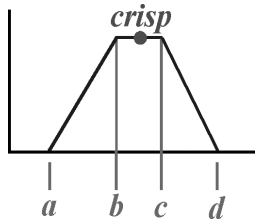


Figura 3.11. Esquema de Difusores

Los difusores se emplean para convertir un valor numérico (*crisp*) en un Conjunto Difuso. Se ha implementado la clase genérica **`difusor`** cuyo cálculo retorna un valor Conjunto Difuso. Los difusores implementados se pueden encontrar en la sección 4.5.1.

Los difusores que se emplean más comunmente se encuentran en la parte derecha de la Figura 3.11. Estos difusores se pueden generalizar en tres grupos (en la parte izquierda de la figura): Pi, Pi-Campana Gaussiana (representada mediante campanas de Gauss) y Pi-Campana Cuadrática (representada mediante curvas parabólicas). Estos tres difusores se resumen en la Tabla 3.7.

Nombre	Operación
Pi 	$\mu(x)=\begin{cases} 0 & x < a \\ \frac{1}{b-a}(x-a) & a \le x \le b \\ 1 & b \le x \le c,\ crisp=\frac{c-b}{2} \\ -\frac{1}{d-c}(x-c) & c \le x \le d \\ 0 & x > d \end{cases}$

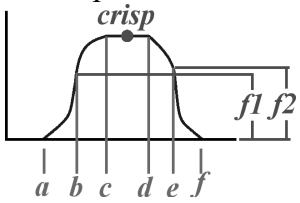
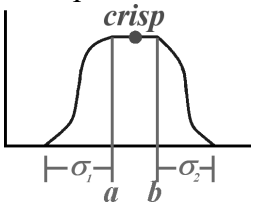
Pi-Campana Cuadrática 	$\mu(x)=\begin{cases} 0 & x < a \\ \sqrt{\frac{x(b-a)^2}{f1}} + a & a \le x \le b \\ c-\sqrt{\frac{(x-1)(b-a)^2}{f1-1}} & b \le x \le c \\ 1 & c \le x \le d,\ crisp=\frac{c-b}{2} \\ d-\sqrt{\frac{(x-1)(e-d)^2}{f2-1}} + a & d \le x \le e \\ f-\sqrt{\frac{x(e-f)^2}{f2}} & e \le x \le f \\ 0 & x > f \end{cases}$
Pi-Campana Gaussiana 	$\alpha(y)=\begin{cases} \left\{crisp-\frac{b-a}{2}-2\sigma_1,\ crisp+\frac{b-a}{2}+2\sigma_2\right\} & y=0 \\ \left\{b-\sigma_1\sqrt{-\lg y},\ c+\sigma_2\sqrt{-\lg y}\right\} & 0<y<1, \\ \{a,b\} & y=1 \end{cases}$ $crisp=\frac{c-b}{2}$

Tabla 3.7. Difusores Implementados

3.6. Base de Reglas

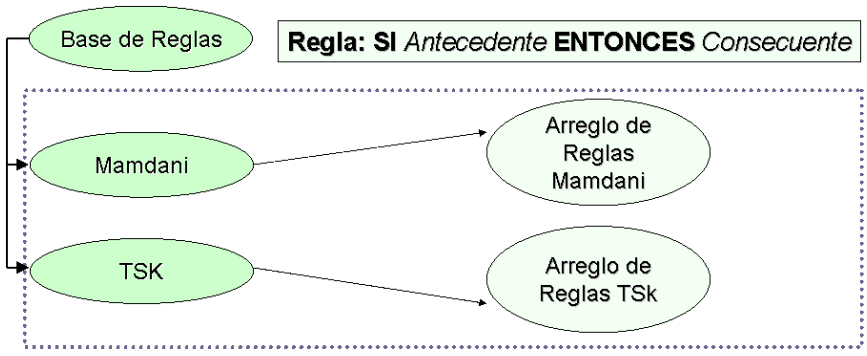


Figura 3.12. Esquema de Bases de Reglas

Se implementaron dos tipos de bases de reglas clasificadas según las reglas que las representan: las tipo Mamdani, cuya definición de clases se encuentra en la sección 4.5.4.1.

y las tipo Takagi-Sugeno-Kang (TSK), cuya definición de clases se encuentra en la sección 4.5.4.2.

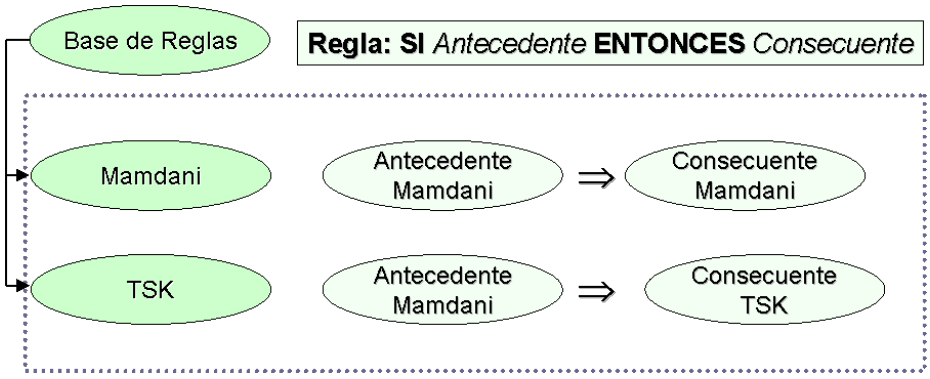


Figura 3.13. Esquema de Reglas

Las reglas Tipo Mamdani están compuestas por Términos de Conjunción tipo Mamdani tanto en el antecedente como en el consecuente y las reglas Tipo TSK están compuestas por Términos de Conjunción tipo Mamdani en el antecedente, y TSK en el consecuente.

Los Términos Tipo Mamdani son del estilo:

A es αa

Los Términos Tipo TSK son del estilo:

A es *f(entradas)*

Las reglas Tipo Mamdani son del estilo:

IF *A* es αa Y B es βb Y ... C es γc THEN D es δd Y E es εe Y ... Y F es ϕf

Donde las letras Mayúsculas indican una *Variable Lingüística*, las letras Minúsculas un *término* de Variable y las letras griegas un *Modificador*.

Las reglas Tipo TSK son del estilo:

IF *A* es αa Y B es βb Y ... C es γc

THEN D es $f_1(A,B,\ldots,C)$ Y E es $f_2(A,B,\ldots,C)$ Y ... Y F es $f_3(A,B,\ldots,C)$

3.7. Sistemas Basados en Reglas

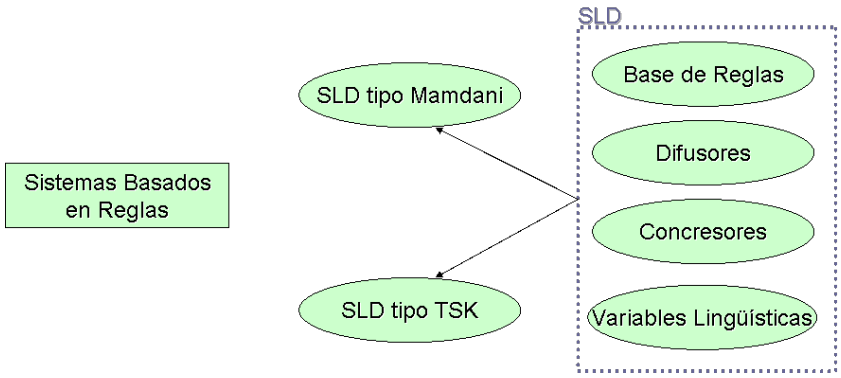


Figura 3.14. Esquema de Sistemas Basados en Reglas

La Base de Reglas, los Difusores, los Concresores y las Variables Lingüísticas componen un Sistema de Lógica Difusa (Sistema Basado en Reglas).

Los Sistemas de Lógica Difusa Implementados (*SLD*) fueron los tipo *Mamdani* cuyo esquema se ajusta a la Figura 3.15, y *TSK* cuyo esquema se ajusta a la Figura 3.16; cada uno recibe el nombre del tipo de reglas que describen el sistema.

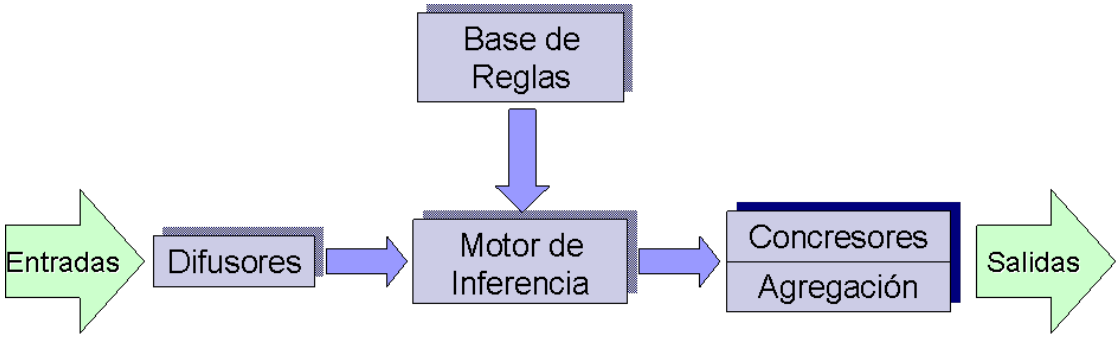


Figura 3.15. Sistema de Lógica Difusa tipo Mamdani

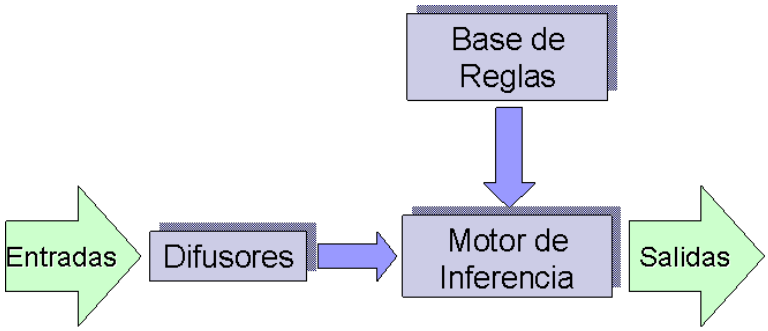


Figura 3.16. Sistema de Lógica Difusa tipo TSK

3.8. Agrupamiento Difuso

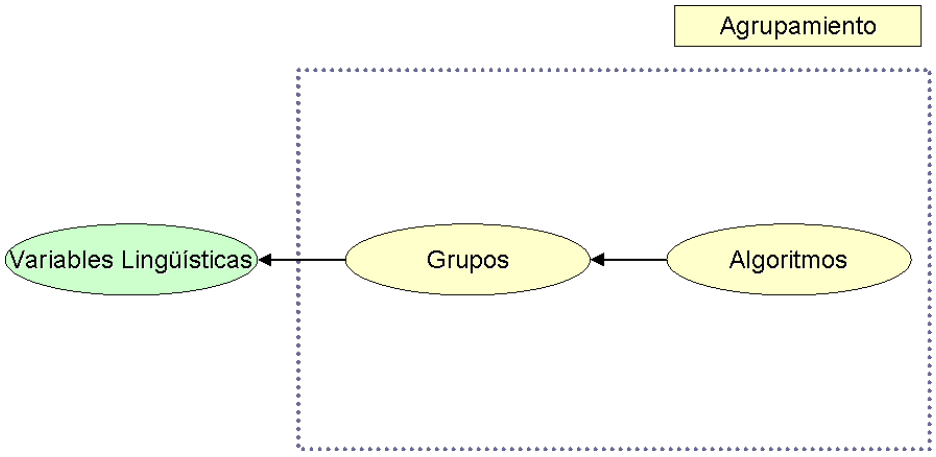


Figura 3.17. Esquema de Agrupamiento Difuso

Las técnicas de agrupamiento difuso consisten en algorimos para generar grupos de datos numéricos que más tarde pueden convertirse en Variables Lingüísticas.

Se implementó el agrupamiento por el método *Fuzzy C-Means* (sección 4.5.5.). También se implementó una clase (sección 4.5.5.2.) basada en el algoritmo de agrupamiento para la creación de *Variables Lingüísticas* con un número de *términos* igual a la cantidad de conjuntos a agrupar.

En la Ecuación (3.3) la matriz *X* representa las *p* entradas, con los *n* casos a agrupar.

$$\mathrm{X} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdots & x_{pn} \end{bmatrix} \tag{3.3}$$

En la Ecuación (3.4) la matriz *V* representa los *c* centros de los para los grupos para las *p* entradas.

$$\mathrm{V} = \begin{bmatrix} v_{11} & v_{12} & \mathsf{L} & v_{1c} \\ v_{21} & v_{22} & \mathsf{L} & v_{2c} \\ \mathsf{L} & \mathsf{M} & \mathsf{O} & \mathsf{M} \\ v_{p1} & v_{p2} & \mathsf{L} & v_{pc} \end{bmatrix} \tag{3.4}$$

En la Ecuación (3.5) la matriz, *U* contiene las *Funciones de Pertenencia* de los *n* casos de x_i a los *c* sub-conjuntos.

$$\mathrm{U} = \begin{bmatrix} u_{11} & u_{12} & \mathsf{L} & u_{1n} \\ u_{21} & u_{22} & \mathsf{L} & u_{2n} \\ \mathsf{M} & \mathsf{M} & \mathsf{O} & \mathsf{M} \\ u_{c1} & u_{c2} & \mathsf{L} & u_{cn} \end{bmatrix} \tag{3.5}$$

Para cada μ_{ik} la *Función de Pertenencia* se encuentra según la Ecuación (3.6). Mediante ella se calculan los puntos de cada variable.

$$\mu_{ik}\left(x_k\right) = \frac{1}{\sum_{j=1}^{k}\left[\frac{d\left(x_k, v_{ik}\right)}{d\left(x_k, v_{jk}\right)}\right]^{\frac{2}{m-1}}} \tag{3.6}$$

Para crear *Variables Lingüísticas* con *c términos* se proyecta la matriz de la Ecuación (3.5) sobre cada x_i, es decir, empleando la Ecuación (3.6) se halla la *Función de Pertenencia* para cada punto del universo de x_i recorriéndolo a intervalos determinados.

4. MANUAL DE USUARIO

En esta sección se describen las clases creadas en la *Librería para Técnicas Difusas* además de las clases `wxString` y `wxArray` de la Librería wxWindows que son las empleadas con mayor frecuencia durante el desarrollo de la presente tesis. En el CD adjunto se han insertado diferentes archivos generados a partir de herramientas de documentación automática, que pueden emplearse directamente sobre la librería, gracias a que esta fue creada con comentarios estructurados.

4.1. Librería wxWindows

4.1.1. `wxString`

Clase que representa un arreglo de caracteres (*string*) de longitud arbitraria, limitado por un tamaño de 2147483647 para máquinas de 32 bits que contiene caracteres arbitrarios que pueden ser ASCII o Unicode. Tine las operaciones estándar, un manejo dinámico de memoria, construcción a partir de otros strings, extracción de sub-strings, conversión de mayúsculas/minúsculas, acceso individual a los caracteres, relleno y corte, etc.

operador +:	Adjunta un wxString
Clear():	Limpia el wxString
GetChar(i):	Devuleve un Caracter de la posición i
SetChar(i,j):	Escribe el caracter j en la posición i
operador <<:	Inserta un caracter convirtiéndolo por ejemplo de double
ToDouble(&arr):	Convierte un wxString y lo deja en *arr.

4.1.2. `wxArray`

Es una clase que describe los arreglos dinámicos de objetos. En wxWindows existen tres clases de arreglos. En la Librería para Técnicas Difusas se empleó básicamente el arreglo de objetos que se define como **`WX_DEFINE_OBJARRAY()`**.

También se emplea una variante pre-definida para arreglos de enteros: **`wxArrayInt`**.

Declaración:	WX_DECLARE_OBJARRAY(MyDirectory, ArrayOfDirectories); WX_DEFINE_OBJARRAY(ArrayOfDirectories);
Item(i):	Retorna el objeto de la posición i
Last():	Retorna el último objeto del arreglo
Add(j):	Agrega al arreglo el objeto j
RemoveAt(i):	Elimina el objeto de la posición i
GetCount():	Retorna la cantidad de objetos del arreglo
Clear():	Elimina todos los objetos del arreglo
Sort(func):	Ordena el arreglo según la función func

4.2. Clases, funciones y otras definiciones Auxiliares

4.2.1. Clase `Double`

Figura 4.1. Clase Double

Clase auxiliar definida para la creación de `wxArray` de *doubles*. Almacena un número *double*. Su implementación se encuentra en los archivos *DoubleArray.h*, y *DoubleArray.cpp*. La Tabla 4.1. muestra los métodos públicos; la Tabla 4.2. muestra los atributos privados.

Double(double x)	Función constructora. Almacena el número double *x*.
virtual ~Double()	Función Destructora.
void Set(double x)	Función que cambia el Valor del double a *x*.

double Get()	Función que retorna el valor del double *numero*

Tabla 4.1. Métodos Públicos de la Clase `Double`

double numero	Número double.

Tabla 4.2. Atributos Privados de la Clase `Double`

4.2.2. `ArrayOfDoubles`

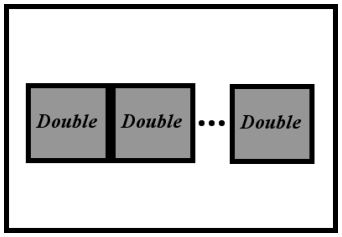


Figura 4.2. Esquema de los `ArrayOfDoubles`

`wxArray` que agrupa los elementos de la clase **`Double`**. Se emplean para añadir y retornar vectores en las matrices **`DoubleArray`**.

4.2.3. `ArregloDeDoubles`

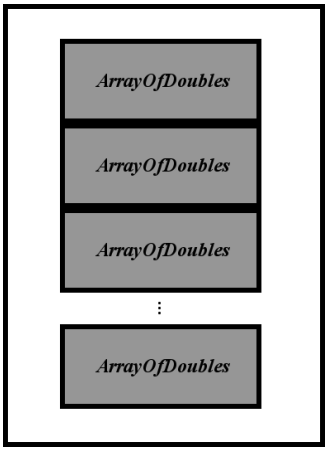


Figura 4.3. Esquema de los `ArrergloDeDoubles`

`wxArray` que agrupa los elementos del tipo **`ArrayOfDoubles`**. Inicialmente se creó para manipular las matrices, pero debido a que se observó que se generaba una complicación en el manejo y se habían creado muchas funciones auxiliares, se creó

DoubleArray (sección 4.2.4.) que agrupa dichas funciones en donde se emplea **ArregloDeDoubles**.

4.2.4. Clase DoubleArray

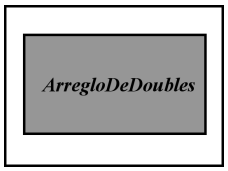


Figura 4.4. Esquema de los DoubleArray

Clase que define una matriz de *(n*m)* elementos empleando un **ArregloDeDoubles**. Las columnas se van agregando poco a poco mediante **ArrayOfDoubles** de tamaño *m*. Su implementación se encuentra en los archivos *DoubleArray.h* y *DoubleArray.cpp*. La Tabla 4.3. muestra los métodos públicos; la Tabla 4.4. muestra los atributos privados.

DoubleArray(int filas=1)	Función constructora de una matriz de **Doubles** de tamaño *m*=filas. Su tamaño por defecto es uno.
~DoubleArray()	Función destructora de una matriz de **Doubles**.
void Add(ArrayOfDoubles &Array)	Función que agrega una columna a la matriz. Pasa como parámetro un **ArrayOfDoubles**.
ArrayOfDoubles GetColumn(int posicion)	Función que retorna una columna de la matriz en la posición **posicion**.
ArrayOfDoubles GetRow(int posicion)	Función que retorna una fila de la matriz en la posición **posicion**.
Double Get(int x, int y)	Función que retorna el valor de una matriz en la posición *(x,y)*.
void OrdenarX()	Función que ordena la matriz según la primera fila.
void QuitarRepetidos()	Función que quita los elementos repetidos de una matriz si todos los elementos de una columna son iguales.
unsigned int GetCount()	Función que retorna la cantidad de filas.
void Clear()	Función que vacía la matriz.
unsigned int GetSize()	Función que retorna el tamaño en columnas de una matriz.
void RemoveAt(int posicion)	Función que elimina una columna de la matriz.
ArrayOfDoubles Last()	Función que retorna la última columna de la matriz.

Tabla 4.3. Métodos Públicos de la Clase DoubleArray

ArregloDeDoubles Arreglo	Arreglo que preserva los elementos de la matriz.

Tabla 4.4. Atributos Privados de la Clase `DoubleArray`

4.2.5. Clase `alfa_corte`

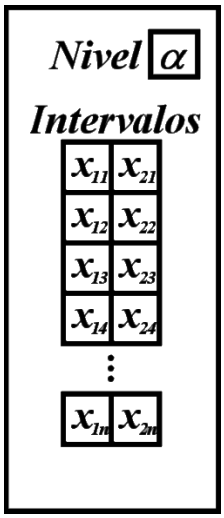


Figura 4.5. Esquema de un `alfa_corte`

Clase que define un *α-corte* de nivel α. Cada *α-corte* tiene *n* intervalos, cada intervalo *i* con cotas $[x_{1i},x_{2i}]$. Su implementación se encuentra en los archivos *AlfaCortes.h* y *AlfaCortes.cpp*. La Tabla 4.5. muestra los métodos públicos; la Tabla 4.6. muestra los atributos privados.

alfa_corte()	Función Constructora. Se define el arreglo de intervalos de dimensión 2.
~alfa_corte()	Función Destructora.
[void RetornarNivel(double &Niv)	Función que retorna el nivel del *α-corte*.
void RetornarIntervalos(DoubleArray &Aux)	Función que retorna los intervalos del α-corte.
void AgregarIntervalo(double x1, double x2)	Función que agrega un intervalo al *α-corte*.
void DefinirNivel(double nivel)	Función que Define el Nivel del *α-corte*.
int RetornarCantidadIntervalos()	Función que retorna la cantidad de intervalos que tiene el *α-corte*.
void QuitarRepetido()	Función que quita los intervalos repetidos en el *α-corte*.

Tabla 4.5. Métodos Públicos de la Clase `alfa_corte`

double Nivel	Nivel del *α-corte*.
DoubleArray Intervalos	Intervalos del *α-corte*. Cada pareja (x_1,x_2) almacena las coordenadas de las cotas de los intervalos.

Tabla 4.6. Atributos Privados de la Clase `alfa_corte`

4.2.6. Clase `termino`

Clase que define los términos de conjunción del estilo ***A es δa*** de los que están hechas las reglas. Se componen de un número entero que representan una *Variable Lingüística* (para *A*), apuntadores a un *modificador* (para *δ*) y un entero *Término de Variable* (para *a*). Su implementación se encuentra en los archivos *BaseReglas.h* y *BaseReglas.cpp*. La Tabla 4.7. muestra los métodos públicos; la Tabla 4.8. muestra los atributos privados.

termino(int var, modificador *modif, int conj)	Función Constructora que define un *Término de Conjunción* compuesto de una *Variable Lingüística* (*var*), un *término* de dicha variable (conjunto difuso *conj*) y un *modificador* de dicho conjunto (**modif.*).
Virtual ~termino()	Función Destructora.
modificador *GetMod()	Función que retorna el apuntador al *modificador*.
int GetCon()	Función que retorna la posición del término de variable.
int GetVar()	Función que retorna la posición de la *Variable Lingüística*.

Tabla 4.7. Métodos Públicos de la Clase `termino`

int variable	Entero que representa la posición de una *Variable Lingüística* en un arreglo de *Variables Lingüísticas*.
int conjunto	Entero que representa la posición de un término de variable de dicha *Variable Lingüística*.
modificador *mod	Apuntador a un **`modificador`** a "conjunto".

Tabla 4.8. Atributos Privados de la Clase `termino`

4.2.7. Clase `terminoTSK`

Clase que define los *términos de consecuente* para reglas tipo *TSK* de el estilo $f_1(A,B,C..,D)$. Se componen de funciones de polinomios en función de las entradas. Tiene de números *doubles* que representan los coeficientes de los polinomios. Su implementación se encuentra en los archivos *BaseReglas.h* y *BaseReglas.cpp*. La Tabla 4.9. muestra los métodos públicos; la Tabla 4.10. muestra los atributos privados.

terminoTSK(DoubleArray &equ)	Función Constructora
virtual ~terminoTSK ()	Función Destructora
DoubleArray GetTerm ()	Función que retorna el arreglo que representa los coeficientes.

Tabla 4.9. Métodos Públicos de la Clase `terminoTSK`

DoubleArray ecuacion	Arreglo de Doubles que representan los coeficientes de una ecuación tipo *Coef_i_0 + Coef_i_1*X_i + Coef_i_2*X_i^2 + ... + Coef_i_n*X_i^n*. Donde *n* representa la posición de la columna del Arreglo, *Coef_i* son los coeficientes de la variable de entrada *X_i*. La cantidad de Filas indican las Entradas, las columnas indican el grado del exponente del polinomio.

Tabla 4.10. Atributos Privados de la Clase `terminoTSK`

4.3. Clases

4.3.1. Clases Básicas

4.3.1.1. Clase `fs_cont_r`

Esta Clase define los conjuntos difusos continuos en los reales. Su implementación se encuentra en los archivos *fs_cont_r.h* y *fs_cont_r.cpp*. La Tabla 4.11. muestra los métodos públicos; la Tabla 4.12. muestra los atributos privados.

fs_cont_r()	Función Constructora. Define la matriz de *Función de Pertenencia* de dimensión=2.
~fs_cont_r()	Función Destructora.
wxString fsLeerNombre()	Función para leer el nombre del *Universo* de un conjunto difuso continuo en los reales.
double *fsLeerUniverso()	Función que retorna el *Universo* de un conjunto difuso continuo en los reales.
bool fsValidarUniverso()	Función para validar los datos introducidos al *Universo* de un conjunto difuso continuo en los reales: Si el límite inferior es mayor que el superior retorna *false*.
void fsNombrarUniverso (wxString name)	Función que permite ponerle un nombre al *Universo* de un conjunto difuso continuo en los reales.
void fsDefinirUniverso (double x, double y)	Función para definir el *Universo* de un conjunto difuso continuo en los reales: *(x,y)* corresponden a los límites inferior y superior respectivamente.

Funciones relativas a la *Función de Pertenencia*	
int fpLeerNoPt()	Función que retorna el número de puntos de la *Función de Pertenencia.*
void fpDefinir (DoubleArray &Fpert)	Función empleada para definir los puntos de la *Función de Pertenencia*. Se ordenan los puntos según su posición en X y se verifica si los límites están incluidos. Si no, se extrapolan el primer y el último punto.
DoubleArray fpLeer ()	Función que retorna los puntos de la *Función de Pertenencia*.
void fpBorrarTodos ()	Función que borra todos los puntos de la *Función de Pertenencia*. Cada vez que se altera la *Función de Pertenencia* es necesario redefinir los *α-cortes*.
Funciones relativas a los α-cortes:	
void acAgregar (alfa_corteACorte)	Función que agrega un *α-corte* al arreglo de *α-cortes* verificando si este existe. De ser así, agrega los intervalos nuevos al *α-corte*.
void acRetornar (int posicion, alfa_corte &Array)	Función que retorna el α-corte de posición **posicion** en el *α-corte* auxiliar **Array**.
int acRetornarPosicion (double nivel)	Función que retorna la posición en el arreglo de *α-cortes* con *α-cortes* de nivel **nivel**.
int acRetornarCantidad ()	Función que retornar la cantidad de los *α-cortes* en el arreglo de *α-cortes*.
void acBorrarTodos ()	Función que borra todos los *α-cortes* del arreglo. Es responsabilidad del usuario re-calcularlos o crearlos nuevamente pues las funciones que dependan de los *α-cortes* producirán errores.
void acCalcularFP ()	Función que Calcula la *Función de Pertenencia* según los *α-cortes* del arreglo. Cada punto de los intervalos de los α-cortes de nivel *α* se convierte en coordenadas *(x, α)* en la *Función de Pertenencia*. Al definir o redefinir la *Función de Pertenencia* es necesario que el usuario defina o redefina la los *α-cortes*. Lo mismo sucede en el caso contrario. Es decir, que por tratarse de una definición dual, no debe olvidarse ninguna de las dos definiciones.
void acBorrar (double nivel)	Borrar un *α-corte* y calcularlo según los puntos de la *Función de Pertenencia*.
void acCalcularAC()	Función para calcular los *α-co*rtes según la *Función de Pertenencia*. Cada punto *(x,y)* de la *Función de Pertenencia* crea un punto del *α-corte* de nivel *y* y las coordenadas del intervalo de dicho *α-corte* corresponden a cada *x*.
void acCalcular (ArrayOfDoubles &puntos_calculo)	Función que calcula los *α-cortes* según los puntos dados en una matriz. Es empleada como función auxiliar de **acCalcularFP**.
void acAgregarIntervalos	Función que agrega un intervalo *(xA,xB)* a un *α-corte* de

(double nivel, double xA, double xB)	determinado nivel.
bool acNivelExiste (double nivel)	Función que retorna *true* si el nivel está en el arreglo de *α-cortes*.
void RetornarAlfacortes (ArregloDeAlfaCortes &ArrAC)	Función que retorna el arreglo de *α-cortes* en el arreglo **ArrAC**.
DoubleArray RetornarAlfacorteN (int posicion)	Función que retorna los intervalos de la posición **posicion**.
void acOrdenar()	Función que ordena los *α-cortes* según el nivel.
double acRetornarNivel (int posicion)	Función que retorna el nivel del *α-corte* de la posición **posicion** en el arreglo de *α-cortes*.
bool ndEsNormal ()	Función que retorna *true* si el conjunto difuso es *normal*. Se busca en todos los valores de *Función de Pertenencia* que por lo menos en un punto esta tome el valor de 1.
bool ndEsConvexo ()	Función que retorna *true* si el conjunto difuso es convexo. Se comprueba que en todos los α-cortes no haya más de un intervalo por nivel.
bool ndEsSemicontinuo ()	Función que retorna *true* si el conjunto difuso es *semi-continuo*. Para que el conjunto sea semi-continuo la *Función de Pertenencia* no debe tener más de un valor en un punto, por tanto si en la definición de puntos existe un punto con más de dos valores de *Función de Pertenencia*, la función no es *semi-continua*. No se implementó por la dificultad de representación en un sistema digital. Por tanto se creó la condición por defecto de que todos los conjuntos difusos sean siempre *semi-continuos*.
bool ndEsNoDifuso ()	Función que retorna *true* si el conjunto difuso es número difuso: es normal, convexo y semi-continuo.
double ValorRepresentativo (double opt, double r)	Retorna el *Valor Representativo* de un ***Número Difuso***. Pasan como parámetros el valor de optimismo β (*opt*) y el valor *r* de α^r. El *Valor Representativo* se halla según la ecuación $V_N(\beta) = \frac{\int_0^1 g(\alpha)\cdot v_N(\alpha,\beta)d\alpha}{\int_0^1 g(\alpha)d\alpha}$ en donde $g(\alpha) = \alpha^r$

Tabla 4.11. Métodos Públicos de la Clase `fs_cont_r`

Variables relativas al *Universo*:	
wxString nombre	*Nombre* del Conjunto Difuso Continuo en ℜ. Es un wxString que es empleado únicamente con fines de identificación.
double universo[2]	Arreglo que guarda las cotas que definen el *Universo* del

	Conjunto. universo[0] es el límite inferior del universo, universo[1] es el límite superior del *Universo*.
Variables Relativas a la *Función de Pertenencia*:	
DoubleArray FuncionPertenencia	*Función de Pertenencia* con coordenadas *(x,μ(x))*. La *Función de Pertenencia* está definida por la interpolación lineal entre todos los puntos que conforman la matriz, es decir, que si *x* no está explícitamente definido por ninguno de los puntos en la *Función de Pertenencia*, se calcula según la ecuación definida por la recta que une los puntos a,b ∈ U tales que $a<x<b$ y *a,b* están definidos explícitamente en la matriz. La *Función de Pertenencia* para *x* está definido por la siguiente ecuación: $\mu_X(x) = \frac{\mu_X(b) - \mu_X(a)}{b-a}(x-a) + \mu_X(a)$, con $b-a \neq 0$.
Variables Relativas a los α-cortes:	
ArregloDeAlfaCortes Alfacortes	Arreglo de *α-cortes*. *Un α-corte* es un intervalo sobre la recta real a una altura *α* determinada, con $\alpha \in [0,1]$. Está definido una matriz de *i α-cortes*. Cada *α-corte* de nivel *α* está compuesto por *k* intervalos cerrados que describen la *Función de Pertenencia*. $$A(\alpha) = \sum_{j=0}^{k} \left[u_{ini}(j), u_{fin}(j)\right]$$

Tabla 4.12. Atributos Privados de la Clase `fs_cont_r`

4.3.1.2. Clase `VariableLinguistica`

Esta Clase define una *Variable Lingüística*. No se encuentra definido explícitamente el *Universo* mismo de la *Variable Lingüística* pero el *Universo* de cada *Término Calificador* está definido explícitamente. Se considera que todos los términos tienen el mismo *Universo*. Para poder garantizar dicha afirmación se ha implementado la función **CorregirUniverso**. Su implementación se encuentra en los archivos *VariableLinguistica.h* y *VariableLinguistica.cpp*. La Tabla 4.13. muestra los métodos públicos; la Tabla 4.14. muestra los atributos privados.

VariableLinguistica()	Función Constructora.
~VariableLinguistica()	Función Destructora.
void AgregarTermino (fs_cont_r &termino)	Función que agrega un término a la *Variable Lingüística*.
void AgregarNTerminos (ArregloDeConjuntosDifusos	Función que agrega varios términos a la *Variable Lingüística*.

&aux)	
fs_cont_r RetornarTermino (unsigned int posicion)	Función que retorna un término e la *Variable Lingüística*.
void CorregirUniverso ()	Función que corrige el *Universo* de la *Variable Lingüística*. Hace que los límites de todos los términos coincidan.
unsigned int GetCount()	Función que retorna la cantidad de términos de la *Variable Lingüística*.
void EliminarTermino (unsigned int posicion)	Función que elimina un término de la *Variable Lingüística* en la posición **posicion**.
void CambiarTermino (fs_cont_r termino, unsigned int posicion)	Función que cambia el conjunto del término que se encuentra en **posicion** por **termino**.
void NombrarVL (wxString name)	Función que define el nombre de la *Variable Lingüística*.
wxString RetornarNombre()	Función que retorna el nombre de la *Variable Lingüística*.
void Clear()	Función que borra todos los términos de la *Variable Lingüística*.
ArregloDeConjuntosDifusos Get()	Retorna los *Términos de Variable*.
void RenombrarTermino (unsigned int posicion, wxString &name)	Cambia el nombre de un término de Variable en la posición **posicion**.

Tabla 4.13. Métodos Públicos de la Clase `VariableLinguistica`

ArregloDeConjuntosDifusos variable	Arreglo de Conjuntos Difusos que contiene los términos de la Variable. Son los Términos Calificadores de la *Variable Lingüística*.
wxString nombre	Nombre de la *Variable Lingüística*. Es un wxString que se emplea con propósitos de identificación de la variable.

Tabla 4.14. Atributos Privados de la Clase `VariableLinguistica`

4.4. Operaciones

4.4.1. Operaciones Uniarias

4.4.1.1. Clase `modificador`

Esta Clase define las operaciones *uniarias*. Es una clase genérica de la que se heredan los denominados "modificadores" que reciben su nombre porque operan *modificando* el Conjunto Difuso original. El diagrama de herencias se encuentra en la Figura 4.6. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y

OperacionesUniarias.cpp. La Tabla 4.15. muestra los métodos públicos; la Tabla 4.16. muestra los atributos públicos.

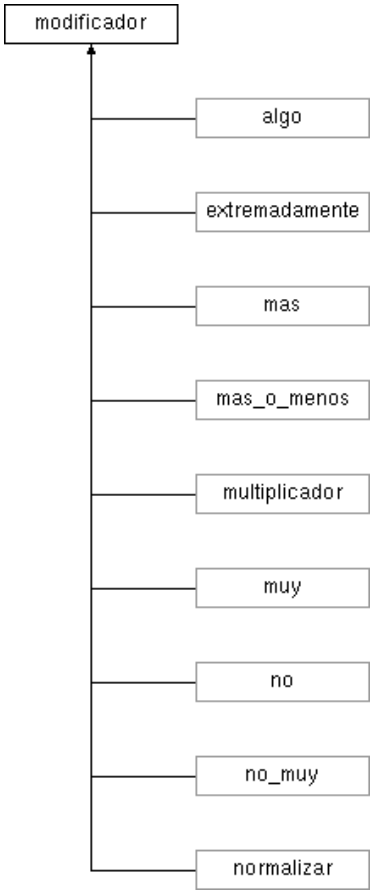


Figura 4.6. Diagrama de herencias de `modificador`

modificador ()	Función Constructora. Define el nombre del *modificador* como "modificador"
virtual ~modificador()	Función Destructora.
virtual fs_cont_r Calcular (fs_cont_r &C1)=0	Función para calcular el modificador. Pasa como parámetro un Conjunto Difuso y retorna un Conjunto Difuso. Implementada en **`no`** (p.65), **`muy`** (p.65), **`mas_o_menos`** (p.67), **`mas`** (p.66), **`algo`** (p.66), **`extremadamente`** (p.68), **`no_muy`** (p.65), **`multiplicador`** (p.67), y **`normalizar`** (p.68).
wxString GetNombre()	Función que retorna el nombre del modificador.
virtual double Get()	Función que retorna un *double*. Se dejó abierta para usar un parámetro en una clase futura. Reimplementado en **`multiplicador`** (p.67)

Tabla 4.15. Métodos Públicos de la Clase `modificador`

wxString wxNombre	Define el nombre del modificador.

Tabla 4.16. Atributos Públicos de la Clase `modificador`

4.4.1.2. Clase `no`

Clase para calcular el complemento de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.17. muestra los métodos públicos.

no()	Función Constructora. Define el nombre del modificador como "no"
~no()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el complemento de un Conjunto Difuso. **nochar** es el resultado de la operación *1-μ(x)* para cada *μ* de la *Función de Pertenencia*. Heredada de **`modificador`** (p.63).

Tabla 4.17. Métodos Públicos de la Clase `no`

4.4.1.3. Clase `no_muy`

Clase para calcular el modificador "no muy" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.18. muestra los métodos públicos.

no_muy()	Función Constructora. Define el nombre del modificador como "no muy"
~no_muy()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "no muy" para un Conjunto Difuso. "no muy" es el resultado de la operación *1-μ(u)^2* para cada punto *u* de la *Función de Pertenencia*. Heredada de **`modificador`** (p.63).

Tabla 4.18. Métodos Públicos de la Clase `no_muy`

4.4.1.4. Clase `muy`

Clase para calcular el modificador "muy" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.19. muestra los atributos públicos.

muy()	Función Constructora. Define el nombre del modificador como "muy"
~muy()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "muy" para un Conjunto Difuso. "muy" es el cuadrado de cada punto de la *Función de Pertenencia*. Heredada de **`modificador`** (p. .63)

Tabla 4.19. Métodos Públicos de la Clase `muy`

4.4.1.5. Clase `algo`

Clase para calcular el modificador "algo" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.20. muestra los atributos públicos.

algo()	Función Constructora. Define el nombre del modificador como "algo"
~algo()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "algo" para un Conjunto Difuso. "algo" es la raíz cúbica de cada punto de la *Función de Pertenencia*. Heredada de **`modificador`** (p.63).

Tabla 4.20. Métodos Públicos de la Clase `algo`

4.4.1.6. Clase `mas`

Clase para calcular el modificador "más" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.21. muestra los atributos públicos.

mas()	Función Constructora. Define el nombre del modificador como "mas"
~mas()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "más" para un Conjunto Difuso. "más" es el resultado de elevar cada punto de la *Función de Pertenencia* a la 1.25. Heredada de **`modificador`** (p. 63).

Tabla 4.21. Métodos Públicos de la Clase `mas`

4.4.1.7. Clase `mas_o_menos`

Clase para calcular el modificador "más o menos" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.22. muestra los atributos públicos.

mas_o_menos ()	Función Constructora. . Define el nombre del modificador como "mas o menos"
~mas_o_menos ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "más o menos" para un Conjunto Difuso. "más o menos " es la raíz cuadrada de cada punto de la *Función de Pertenencia*. Heredada de modificador (p. 63).

Tabla 4.22. Métodos Públicos de la Clase `mas_o_menos`

4.4.1.8. Clase `multiplicador`

Clase para calcular el modificador "multiplicador" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.23. muestra los métodos públicos; la Tabla 4.24. muestra los atributos privados.

multiplicador ()	Función Constructora. . Define el nombre del modificador como "multiplicador"
~multiplicador ()	Función Destructora.
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "multiplicador" para un Conjunto Difuso. "multiplicador" es el resultado de la operación *factor*μ(u)* para cada punto u de la *Función de Pertenencia*. Heredada de **`modificador`** (p. 63).
void Set (double mult)	Función que fija el valor del factor multiplicador.
double Get ()	Función que retorna el valor del factor multiplicador. Reimplementado de **`modificador`** (p. 63).

Tabla 4.23. Métodos Públicos de la Clase `multiplicador`

double factor	Factor para multiplicar por el valor de μ en cada punto de la *Función de Pertenencia*.

Tabla 4.24. Atributos Privados de la Clase `multiplicador`

4.4.1.9. Clase `extremadamente`

Clase para calcular el modificador "extremadamente" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.25. muestra los métodos públicos.

extremadamente ()	Función Constructora. Define el nombre del modificador como "extremadamente"
~extremadamente ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "extremadamente" para un Conjunto Difuso. "extremadamente" es el cubo de cada punto de la *Función de Pertenencia*. Heredada de **`modificador`** (p. 63).

Tabla 4.25. Métodos Públicos de la Clase `extremadamente`

4.4.1.10. Clase `normalizar`

Clase para calcular el modificador "normalizar" de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesUniarias.h* y *OperacionesUniarias.cpp*. La Tabla 4.26. muestra los métodos públicos.

Normalizar ()	Función Constructora. Define el nombre del modificador como "normalizar"
~normalizar ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1)	Función que calcula el modificador "normalizar" para un Conjunto Difuso. "normalizar" convierte los valores inferiores al 20% de la altura de la *Función de Pertenencia* en cero y superiores al 80% de la altura en 1, los demás valores permanecen sin cambio. Heredada de **`modificador`** (p. 63).

Tabla 4.26. Métodos Públicos de la Clase `normalizar`

4.4.2. Operaciones Difusas Binarias

4.4.2.1. Clase `operacion_difusa`

Esta Clase define las operaciones entre dos conjuntos difusos que retorna otro conjunto difuso. Es una clase genérica de la que se heredan diversas operaciones. El diagrama de herencias se encuentra en la Figura 4.7. Su implementación se encuentra en los archivos

Operaciones.h y *Operaciones.cpp*. La Tabla 4.27. muestra los métodos públicos; la Tabla 4.28. muestra los atributos públicos; la Tabla 4.29 muestra los atributos privados.

Figura 4.7. Diagrama de herencias de `operación_difusa`

operacion_difusa ()	Función Constructora. Asigna la dimensión 3 a PuntosCalculo y los valores por defecto de *extrapolar=true* y que se empleen los alfa-cortes ya establecidos.
Virtual ~operacion_difusa ()	Función Destructora.
void BuscarPuntosAnalisis (fs_cont_r &C1, fs_cont_r &C2)	Función que busca los puntos de análisis de la operación. Estos puntos incluyen los puntos sobre el universo que se encuentran en la *Función de Pertenencia* de cada conjunto y si no están definidos en el otro conjunto, se calculan. También se incluyen los puntos en donde se intersectan las Funciones de Pertenencia de los conjuntos.
void BuscarAlfacortesAnalisis (fs_cont_r &C1,fs_cont_r &C2)	Función que busca los alfa-cortes de análisis de la operación. Estos *α-cortes* incluyen los de cada conjunto y si no están definidos en el otro conjunto, se calculan. Es responsabilidad del usuario verificar si cada conjunto es número difuso, pues únicamente se toma el primer intervalo de cada conjunto. Si el parámetro indica que no se emplean los α-cortes ya establecidos, se calculan unos nuevos de manera uniforme.
virtual fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)=0	Función que retorna un Conjunto Difuso como resultado de la operación.Implementada en **`minimo`** (p.70), **`maximo`** (p.71), **`suma`** (p.71), **`resta`** (p.71), **`multiplicacion`** (p.72), **`interv_min`** (p.72), y **`interv_max`** (p.73).
void RetornarCalculo (DoubleArray &auxPC)	Función que retorna el valor de los Puntos de Cálculo en la matriz auxPC.
void CambiarCantidadDeAlfacortes (int cant)	Función que cambia el valor de **CantidadDeAlfacortes**.
void ExtrapolarUniverso (bool extra=true)	Función que define si los cálculos se harán en la parte común del universo o en la suma de los universos de ambos.

Tabla 4.27. Métodos Públicos de la Clase `operación_difusa`

DoubleArray PuntosCalculo	Matriz que almacena los puntos de cálculo de la operación difusa. Tiene dimensión *n*3*. En donde cada punto *(i,f1,f2)* con *i* un punto dentro del universo a analizar, *f1* el valor de la *Función de Pertenencia* en ese punto *i* para el primer Conjunto Difuso y *f2* el valor de la *Función de Pertenencia* en el punto *i* para el segundo Conjunto Difuso.
ArregloDeAlfaCortes AlfacortesCalculo	Arreglo que almacena los *α-cortes* de cálculo de la operación difusa. Almacena el nivel y los intervalos para los Conjuntos Difusos con los que se va a Operar.
int CantidadDeAlfacortes	Define (en caso de que no se empleen los alfa-cortes ya establecidos) la cantidad de *α-cortes* con que se realizará el cálculo.
int TipoCalculo	Variable empleada para los cálculos de Tipo 1 (con alfa-cortes) en los que -1 implica que se usan los valores actuales de los alfa-cortes. Define qué tipo de Cálculo se va a realizar: 0 para Cálculo por *Función de Pertenencia* y 1 para Cálculo por *α-cortes*.

Tabla 4.28. Atributos Públicos de la Clase `operación_difusa`

bool extrapolar	Si *extrapolar=false* si se debe mantener el resultado de la operación difusa dentro de los límites en común de los dos conjuntos. En caso contrario se extrapolan los conjuntos difusos, con el valor del límite del universo.

Tabla 4.29. Atributos Privados de la Clase `operación_difusa`

4.4.2.2. Clase `minimo`

Clase para calcular el mínimo de dos conjuntos difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.30. muestra los métodos públicos.

minimo ()	Función Constructora. Define que la operación es calculada por los Puntos de la *Función de Pertenencia*.
~minimo ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "mínimo". Cada punto *(i,f1,f2)* de PuntosCalculo se examina y la *Función de Pertenencia* del conjunto resultante tiene las coordenadas *(i,min(f1,f2))*. Heredada de **`operacion_difusa`** (p.68).

Tabla 4.30. Métodos Públicos de la Clase `minimo`

4.4.2.3. Clase `maximo`

Clase para calcular el máximo de dos conjuntos difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.31. muestra los métodos públicos.

maximo ()	Función Constructora. Define que la operación escalculada por los Puntos de la *Función de Pertenencia*.
~maximo ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "máximo". Cada punto *(i,f1,f2)* de PuntosCalculo se examina y la *Función de Pertenencia* del conjunto resultante tiene las coordenadas *(i,max(f1,f2))*. Heredada de **`operacion_difusa`** (p. 68).

Tabla 4.31. Métodos Públicos de la Clase `maximo`

4.4.2.4. Clase `suma`

Suma de dos conjuntos difusos. Es responsabilidad del usuario determinar que los conjuntos empleados sean números difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.32. muestra los métodos públicos.

suma ()	Función Constructora. Define que la operación es calculada por los α-cortes de cálculo.
~suma ()	Función Destructora.
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "suma". Cada α-corte *(alfa,[xia,xib],[xic,xid])* de AlfacortesCalculo se examina y el α-corte del conjunto resultante tiene las coordenadas *(i ,[xia + xic, xib + xid])*. Heredada de **`operacion_difusa`** (p68).

Tabla 4.32. Métodos Públicos de la Clase `suma`

4.4.2.5. Clase `resta`

Resta de dos conjuntos difusos. Es responsabilidad del usuario determinar que los conjuntos empleados sean números difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.33. muestra los métodos públicos.

resta ()	Función Constructora. Define que la operación es calculada por los α-cortes de cálculo.
~resta ()	Función destructora.
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "resta". Cada α-corte *(alfa,[xia,xib],[xic,xid])* de AlfacortesCalculo se examina y el α-corte del conjunto resultante tiene las coordenadas *(i,[xia - xid, xib - xic)])*. Heredada de **operacion_difusa** (p. 68).

Tabla 4.33. Métodos Públicos de la Clase `resta`

4.4.2.6. Clase `multiplicacion`

Multiplicación de dos conjuntos difusos. Es responsabilidad del usuario determinar que los conjuntos empleados sean números difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.34. muestra los métodos públicos.

multiplicacion ()	Función Constructora. Define que la operación es calculada por los *α-cortes* de cálculo.
~multiplicacion ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "multiplicación". Cada α-corte *(alfa,[xia,xib],[xic,xid])* de AlfacortesCalculo se examina y la α-corte del conjunto resultante tiene las coordenadas *(i,[min(xia * xic, xia * xid, xib * xic, xib *xid),max((xia * xic, xia * xid, xib * xic, xib *xid))])*. Heredada de **operacion_difusa** (p. 68).

Tabla 4.34. Métodos Públicos de la Clase `multiplicación`

4.4.2.7. Clase `interv_min`

Mínimo de los intervalos de los *α-cortes* de dos conjuntos difusos. Es responsabilidad del usuario determinar que los conjuntos empleados sean números difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.35. muestra los métodos públicos.

interv_min ()	Función Constructora. Define que la operación es calculada por los *α-cortes* de cálculo.
~interv_min ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1,	Retorna el resultado de la operación "mínimo intervalar". Cada *α-corte (alfa,[xia,xib],[xic,xid])* de AlfacortesCalculo se examina

fs_cont_r &C2)	y la alfa-corte del conjunto resultante tiene las coordenadas *(i,[min(xia,xiac),min(xib,xid)])*. Heredada de **operacion_difusa** (p. 68).

Tabla 4.35. Métodos Públicos de la Clase `interv_min`

4.4.2.8. Clase `interv_max`

Máximo de los intervalos de los *α-cortes* de dos conjuntos difusos. Es responsabilidad del usuario determinar que los conjuntos empleados sean números difusos. Su implementación se encuentra en los archivos *Operaciones.h* y *Operaciones.cpp*. La Tabla 4.36 muestra los métodos públicos.

interv_max ()	Función Constructora. Define que la operación es calculada por los *α-cortes* de cálculo.
~interv_max ()	Función Destructora
fs_cont_r Calcular (fs_cont_r &C1, fs_cont_r &C2)	Retorna el resultado de la operación "máximo intervalar". Cada *α-corte (alfa,[xia,xib],[xic,xid])* de AlfacortesCalculo se examina y la alfa-corte del conjunto resultante tiene las coordenadas *(i,[max(xia, xiac),max(xib,xid)])*. Heredada de **operacion_difusa** (p. 68).

Tabla 4.36. Métodos Públicos de la Clase `interv_max`

4.4.3. Operaciones Difusas que Retornan Valores

4.4.3.1. Clase `valores`

Esta Clase define todas las operaciones difusas que retornan un valor flotante (*double*). Es una clase genérica de la que se heredan diversas operaciones. El diagrama de herencias se encuentra en la Figura 4.8. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.37. muestra los métodos públicos.

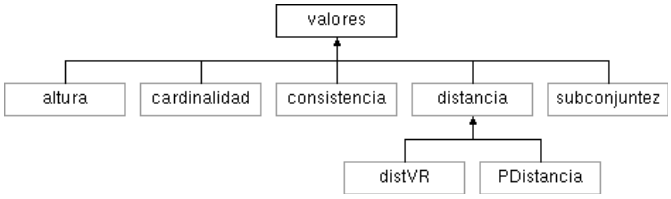


Figura 4.8. Diagrama de herencias de `valores`

valores()	Función Constructora.
~valores ()	Función Destructora.
virtual double Calcular (ArregloDeConjuntosDifusos &array)=0	Función para calcular el valor de la operación difusa. Pasa como parámetro un arreglo de Conjuntos Difusos y retorna un double. Implementada en **consistencia** (p.74), **cardinalidad** (p.74), **subconjuntez** (p.75), **altura** (p.75), **distancia** (p.75), **PDistancia** (p.76), y **distVR** (p.76).

Tabla 4.37. Métodos Públicos de la Clase `valores`

4.4.3.2. Clase `consistencia`

Clase para calcular la consistencia de varios Conjuntos Difusos. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.38 muestra los métodos públicos.

consistencia ()	Función Constructora
~consistencia ()	Función Destructora
double Calcular (ArregloDeConjuntosDifusos &array)	Función para calcular la Consistencia de un Arreglo de Conjuntos Difusos. Retorna -1 si no se introducen conjuntos. Si se introduce un sólo Conjunto, retorna el valor de la consistencia de él mismo con él mismo. La consistencia es la altura de los mínimos de todos los Conjuntos Difusos en el Arreglo. Heredada de **valores** (p.73).

Tabla 4.38. Métodos Públicos de la Clase `consistencia`

4.4.3.3. Clase `cardinalidad`

Clase para calcular la cardinalidad de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.39 muestra los métodos públicos.

cardinalidad ()	Función Constructora
~cardinalidad ()	Función Destructora
double Calcular (ArregloDeConjuntosDifusos &array)]	Función para calcular la Cardinalidad del primer Conjunto Difuso del arreglo definida como el área bajo la curva de la Función dePertenencia. Heredada de **valores** (p.73).

Tabla 4.39. Métodos Públicos de la Clase `cardinalidad`

4.4.3.4. Clase `subconjuntez`

Clase para calcular el Índice de Subconjuntez de un Conjunto Difuso con respecto a Otro. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.40 muestra los métodos públicos.

subconjuntez()	Función Constructora.
~subconjuntez ()	Función Destructora.
double Calcular (ArregloDeConjuntosDifusos &array)]	Función que retorna un valor que indica que tan subconjunto es *array[0]* de *array[1]*. El índice de subconjuntez de *(array[0],array[1])* está definido como la cardinalidad de la intersección de los dos conjuntos, dividido por la cardinalidad de *array[0]*. Sólo se deben ingresar dos Conjuntos Difusos en el arreglo, o retornará un valor de -1. Heredada de **`valores`** (p.73).

Tabla 4.40. Métodos Públicos de la Clase `subconjuntez`

4.4.3.5. Clase `altura`

Clase para calcular la altura de un Conjuntos Difusos. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.41 muestra los métodos públicos.

altura ()	Función Constructora.
~altura ()	Función Destructora.
double Calcular (ArregloDeConjuntosDifusos &array)	Función que retorna un el mayor valor de la *Función de Pertenencia* de de la primera posición del Arreglo. Heredada de **`valores`** (p.73).

Tabla 4.41. Métodos Públicos de la Clase `altura`

4.4.3.6. Clase `distancia`

Clase genérica que define las distancias entre Conjuntos Difusos. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.42 muestra los métodos públicos.

distancia ()	Función Constructora.
~distancia ()	Función Destructora.
virtual double Calcular (ArregloDeConjuntosDifusos &array)=0	Función para calcular el el valor de la distancia entre varios Conjuntos Difusos. Pasa como parámetro un arreglo de Conjuntos Difusos y retorna un double. Heredada de valores (p.73). Implementado en **`PDistancia`** (p.76), y **`distVR`** (p.76).

Tabla 4.42. Métodos Públicos de la Clase `distancia`

4.4.3.7. Clase `PDistancia`

Clase para calcular la P-distancia entre dos Conjuntos Difusos. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.43 muestra los métodos públicos, la Tabla 4.44 muestra los atributos privados.

PDistancia ()	Función Constructora
~PDistancia ()	Función Destructora
void DefinirP (double pe)	Función que define el valor de P.
double Calcular (ArregloDeConjuntosDifusos &array)	Función que Calcula la P-Distancia entre los dos primeros Conjuntos Difusos del Arreglo. Heredada de **`distancia`** (p.75).

Tabla 4.43. Métodos Públicos de la Clase `PDistancia`

double p	Valor del exponente P de la P-distancia.

Tabla 4.44. Atributos Privados de la Clase `PDistancia`

4.4.3.8. Clase `distVR`

Clase para calcular distancia a través del *Valor Representativo* entre dos Conjuntos Difusos. Su implementación se encuentra en los archivos *OperacionesValores.h* y *OperacionesValores.cpp*. La Tabla 4.45 muestra los métodos públicos; la Tabla 4.46 muestra los atributos privados.

distVR ()	Función Constructora.Define los parámetros de cálculo por defecto del *Valor Representativo*.
~distVR ()	Función Destructora
void DefinirParametros (double optimismo, double ere)	Define los parámetros de cálculo para el *Valor Representativo* de los Conjuntos.

double Calcular (ArregloDeConjuntosDifusos &array)	Función que Calcula la P-Distancia entre los dos primeros Conjuntos Difusos del Arreglo. Heredada de **`distancia`** (p.75).

Tabla 4.45. Métodos Públicos de la Clase `DistVR`

double r	Parámetro *r* para α' del *Valor Representativo*.
Double opt	Optimismo para el *Valor Representativo*.

Tabla 4.46. Atributos Privados de la Clase `DistVR`

4.4.4. Otras Operaciones Difusas

4.4.4.1. Clase `soporte`

Clase que retorna el soporte (los intervalos del α-corte cero) de un Conjunto Difuso. Su implementación se encuentra en los archivos *OperacionesOtras.h* y *OperacionesOtras.cpp*. La Tabla 4.45 muestra los métodos públicos; la Tabla 4.46 muestra los atributos privados.

soporte ()	Función Constructora.
~soporte ()	Función Destructora.
DoubleArray Calcular (fs_cont_r&C1)	Función que retorna el soporte (los intervalos del *α-corte* cero) de un conjunto Difuso.

Tabla 4.47. Métodos Públicos de la Clase `interv_max`

4.4.4.2. Clase `subconjunto`

Clase que retorna un booleano indicando si C1 es subconjunto de C2. Su implementación se encuentra en los archivos *OperacionesOtras.h* y *OperacionesOtras.cpp*. La Tabla 4.48 muestra los métodos públicos; la Tabla 4.49 muestra los atributos públicos.

subconjunto ()	Función Constructora.
~subconjunto ()	Función Destructora.
bool Calcular (fs_cont_r &C1, fs_cont_r &C2)	Función que retorna un booleano que indica si C1 es subconjunto de C2. Si en PuntosAnálisis si para cada punto de análisis *(i,f1,f2) f2<=f1*, entonces retorna *true*.
void BuscarPuntosAnalisis (fs_cont_r &C1, fs_cont_r &C2)	Función que busca los puntos en los que se evalúan C1 y C2. Emplea el mismo principio que la función con el mismo nombre en **`operación_difusa`**.

Tabla 4.48. Métodos Públicos de la Clase `subconjunto`

DoubleArray PuntosCalculo	Almacena los puntos en los que se evalúan los conjuntos C1 y C2.

Tabla 4.49. Atributos Públicos de la Clase `subconjunto`

4.5. Sistemas de Lógica Difusa

En esta sección se encuentran las clases que se emplean en los Sistemas de Lógica Difusa como las Reglas, los Concresores y los Difusores.

4.5.1. Difusores

4.5.1.1. Clase `difusor`

Clase genérica que define un Difusor que genera un conjunto difuso. Su implementación se encuentra en los archivos *Difusores.h* y *Difusores.cpp*. El diagrama de herencias se encuentra en la Figura 4.9. La Tabla 4.50 muestra los métodos públicos. Esta Clase define todas las operaciones difusas que retornan un valor flotante (*double*).

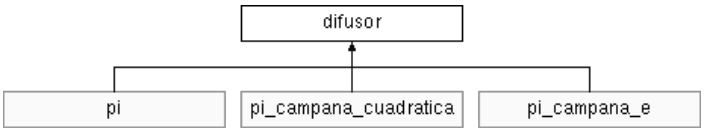


Figura 4.9. Diagrama de herencias de `difusor`

difusor ()	Función Constructora.
virtual ~difusor ()	Función Destructora.
virtual fs_cont_r Calcular (double crisp)	Función que genera un Conjunto Difuso a partir de un número flotante (crisp). Reimplementado en **`pi`** (p.79), **`pi_campana_cuadratica`** (p.79), y **`pi_campana_e`** (p.80).

Tabla 4.50. Métodos Públicos de la Clase `difusor`

4.5.1.2. Clase `pi`

Clase que define el difusor de *Pi*, *Triángulo* y *Singleton*. Su implementación se encuentra en los archivos *Difusores.h* y *Difusores.cpp*. La Tabla 4.51 muestra los métodos públicos; la Tabla 4.52 muestra los atributos privados.

pi ()	Función Constructora. Define los valores predeterminados de las amplitudes como 0.5.
~pi ()	Función Destructora
fs_cont_r Calcular (double crisp)	Función que genera un conjunto difuso tipo *pi* con los valores de **amp1**, **amp2** y **amp3**. A partir de un número double **crisp**. Reimplementado de **`difusor`** (p.78).
void DefinirAmplitudes (double amp1, double amp2, double amp3)	Función que define las amplitudes del trapecio.

Tabla 4.51. Métodos Públicos de la Clase `pi`

Para un trapecio típico *(a,b,c,d)* **amp1** es la distancia en x entre *a* y *b* **amp2** la distancia entre *b* y *c* y **amp3** la distancia entre *c* y *d*

double ampl1	Distancia *[a,b]*
double ampl2	Distancia *[b,c]*
double ampl3	Distancia *[c,d]*

Tabla 4.52. Atributos privadosde la Clase `pi`

4.5.1.3. Clase `pi_campana_cuadratica`

Clase que define el difusor Campana y Pi-Campana por medio de ecuaciones cuadráticas. Su implementación se encuentra en los archivos *Difusores.h* y *Difusores.cpp*. La Tabla 4.53 muestra los métodos públicos; la Tabla 4.54 muestra los atributos privados.

pi_campana_cuadratica()	Función Constructora. Define los valores predeterminados de las amplitudes como 0.5 y las alturas de las parábolas como 0.5 y la cantidad de *α-cortes* como 21.
~pi_campana_cuadratica()	Función Destructora.
fs_cont_r Calcular() (double crisp)	Función que genera un conjunto difuso tipo pi-campana a partir de 4 parábolas. Se generan las parábolas a partir de los *α-cortes* **cant_alfa**. Reimplementado de **`difusor`** (p.78).

void DefinirAmplitudes() (double dim1, double dim2, double dim3, double dim4, double dim5, double f1a, double f2a, double c_alfa)	Función que define las amplitudes de las curvas cuadráticas.

Tabla 4.53. Métodos Públicos de la Clase `pi_campana_cuadratica`

Para una campana definida por 4 porciones de parábolas cuyo inicio y fin está definido por los siguientes puntos: *[a,0], [b,f1], [c,1], [d,1], [e,f2], [f,0]*. Lo que se ha definido son las amplitudes que se emplean alrededor de un número crisp.	
double dim1a()	Distancia *[a,b]*
double dim2a()	Distancia *[b,c]*
double dim3a()	Distancia *[c,d]*
double dim4a()	Distancia *[d,e]*
double dim5a()	Distancia *[e,f]*
double f1aa()	Altura final de la primera parábola entre *[0,1]*
double f2aa()	Altura final de la última parábola entre *[0,1]*
double cant_alfa()	Cantidad de *α-cortes* con los que se va a definir la pi-campana.

Tabla 4.54. Atributos Privados de la Clase `pi_campana_cuadratica`

4.5.1.4. Clase `pi_campana_e`

Clase que define el difusor Campana y Pi-Campana por medio de campanas Gaussianas. Su implementación se encuentra en los archivos *Difusores.h* y *Difusores.cpp*. La Tabla 4.55 muestra los métodos públicos; la Tabla 4.56 muestra los atributos privados.

pi_campana_e ()	Función Constructora. Define los valores predeterminados de las σs como 1 y la amplitud de la porción recta como 1.
~pi_campana_e ()	Función Destructora.
fs_cont_r Calcular (double crisp)	Función que genera un conjunto difuso tipo pi-campana a partir de 2 campanas gaussianas. Se generan las campanas a partir de los *α-cortes* **cantidad_alfa**. Reimplementado de **`difusor`** (p. 78).
void DefinirAmplitudes (double sigma1a, double sigma2a, double dim1a, double cant_a)	Función que define los parámetros de las campanas gaussianas.

Tabla 4.55. Métodos Públicos de la Clase `pi_campana_e`

Para una campana definida por 2 porciones de campanas gaussianas cuyo inicio y fin está definido por la siguiente ecuación:

$$\mu(u) = e^{\left[-\frac{u-c}{\sigma}\right]^2} \tag{4.1}$$

Lo que se ha definido son las amplitudes que se emplean alrededor de un número crisp.

double sigma1	σ para la porción izquierda de la campana
double sigma2	σ para la porción derecha de la campana
double dim1	Amplitud de la meseta de la pi-campana
double cantidad_alfa	Cantidad de *α-cortes* con los que se va a definir la pi-campana.

Tabla 4.56. Métodos Públicos de la Clase `pi_campana_e`

4.5.2. Concresores

4.5.2.1. Clase `concresor`

Clase genérica que define un concresor y retorna un valor flotante (*double*). Su implementación se encuentra en los archivos *Concresores.h* y *Concresores.cpp*. El diagrama de herencias se encuentra en la Figura 4.10. La Tabla 4.57 muestra los métodos públicos. Esta Clase define todas las operaciones difusas que retornan un valor flotante (double).

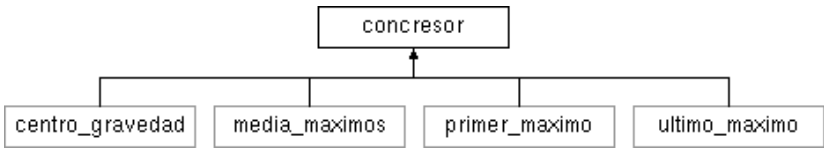


Figura 4.10. Diagrama de herencias de `concresor`

Concresor ()	Función Constructora.
virtual ~concresor ()	Función Destructora.
virtual double Calcular (fs_cont_r &C1)	Función para calcular el valor del concresor. Pasa como parámetro un Conjunto Difuso y retorna un *double*. Reimplementado en **`primer_maximo`** (p.82), **`ultimo_maximo`** (p.82), **`media_maximos`** (p.82), y **`centro_gravedad`** (p.83).

Tabla 4.57. Métodos Públicos de la Clase `concresor`

4.5.2.2. Clase `primer_maximo`

Clase para calcular el concresor "Primer Máximo". Su implementación se encuentra en los archivos *Concresores.h* y *Concresores.cpp*. La Tabla 4.58 muestra los métodos públicos.

primer_maximo ()	Función Constructora
~primer_maximo ()	Función Destructora
double Calcular (fs_cont_r &C1)	Función para calcular el Concresor Primer Máximo. Se define como el Primer punto del primer intervalo del *α-corte* mayor. Reimplementado de **`concresor`** (p. 81).

Tabla 4.58. Métodos Públicos de la Clase `primer_maximo`

4.5.2.3. Clase `ultimo_maximo`

Clase para calcular el concresor "último máximo". Su implementación se encuentra en los archivos *Concresores.h* y *Concresores.cpp*. La Tabla 4.59 muestra los métodos públicos.

ultimo_maximo ()	Función Constructora
~ultimo_maximo ()	Función Destructora
double Calcular (fs_cont_r &C1)	Función para calcular el Concresor Primer Máximo. Se define como el último punto del último intervalo del *α-corte* mayor. Reimplementado de **`concresor`** (p.81).

Tabla 4.59. Métodos Públicos de la Clase `ultimo_maximo`

4.5.2.4. Clase `media_maximos`

Clase para calcular el concresor "Media de Máximos". Su implementación se encuentra en los archivos *Concresores.h* y *Concresores.cpp*. La Tabla 4.60 muestra los métodos públicos.

media_maximos ()	Función Constructora
~media_maximos ()	Función Destructora
double Calcular (fs_cont_r &C1)	Función para calcular el Concresor Media de Máximos. Se define como el la media entre el primer punto del primer intervalo y el último punto del último intervalo del *α-corte* mayor. Reimplementado de **`concresor`** (p.81).

Tabla 4.60. Métodos Públicos de la Clase `media_maximos`

4.5.2.5. Clase `centro_gravedad`

Clase para calcular el concresor "Centro de Gravedad". Su implementación se encuentra en los archivos *Concresores.h* y *Concresores.cpp*. La Tabla 4.61 muestra los métodos públicos; la Tabla 4.62 muestra los atributos privados.

centro_gravedad ()	Función Constructora
~centro_gravedad ()	Función Destructora
double Calcular (fs_cont_r &C1)	Función para calcular el Concresor Centro de Gravedad.. Reimplementado de **`concresor`** (p.81). Se define como: $\frac{\int_X x \cdot \mu(x)}{\int_X \mu(x)}$
void SetCantPuntos (unsigned int pt)	Función para Fijar la cantidad de puntos que definirán el cálculo de las integrales.

Tabla 4.61. Métodos Públicos de la Clase `centro_gravedad`

unsigned int cant_puntos	Cantidad de puntos para calcular las integrales.

Tabla 4.62. Atributos Privados de la Clase `centro_gravedad`

4.5.3. Reglas

4.5.3.1. Clase `Regla`

Reglas tipo Mamdani compuestas por términos de antecedentes y consecuentes tipo clase *termino*. Su implementación se encuentra en los archivos *BaseReglas.h* y *BaseReglas.cpp*. La Tabla 4.63 muestra los métodos públicos; la Tabla 4.64 muestra los atributos privados.

Regla (Terminos &ant, Terminos &con)	Función Constructora. Crea una Regla Tipo Mamdani: SI antecedente ENTONCES consecuente.
virtual ~Regla ()	Función Destructora
Terminos RetornarAntecedente ()	Función que retorna el antecedente de la regla.
Terminos RetornarConsecuente ()	Función que retorna el consecuente de la regla.
termino RetornarTerminoA (int	Función que retorna el término del antecedente de la

posicion)	regla que se encuentra en **posicion**.
termino RetornarTerminoC (int posicion)	Función que retorna el *termino* del consecuente de la regla que se encuentra en **posicion**.
void AgregarTerminoA (termino &ter)	Función que agrega un *termino* al antecedente de la regla.
void AgregarTerminoC (termino &ter)	Función que agrega un *termino* al consecuente de la regla.

Tabla 4.63. Métodos Públicos de la Clase `regla`

Terminos Antecedente	Almacena los términos del antecedente y el consecuente. Cada arreglo representa una conjunción del tipo *T1 & T2 &...&Tn*. Donde *Ti* es un *termino*.
Terminos Consecuente	Almacena los términos del antecedente y el consecuente. Cada arreglo representa una conjunción del tipo *T1 & T2 &...&Tn*. Donde *Ti* es un *termino*.

Tabla 4.64. Atributos Privados de la Clase `regla`

4.5.3.2. Clase `ReglaTSK`

Reglas tipo TSK compuestas por términos de antecedentes tipo *Variable Lingüística+Modificador+Término* y consecuentes tipo polinomial (*f(entradas)*). Su implementación se encuentra en los archivos *BaseReglas.h* y *BaseReglas.cpp*. La Tabla 4.65 muestra los métodos públicos; la Tabla 4.66 muestra los atributos privados.

ReglaTSK (Terminos &ant, TerminosTSK &con)	Función Constructora. Crea una Regla Tipo TSK: SI antecedente ENTONCES *f(x)*. Donde *f(x)* es un *terminoTSK*.
virtual ~ReglaTSK ()	Función Destructora.
Terminos RetornarAntecedente ()	Función que retorna los términos del antecedente.
TerminosTSK RetornarConsecuente ()	Función que retorna los términos del consecuente.
termino RetornarTerminoA (int posicion)	Función que retorna el *termino* del antecedente de la regla que se encuentra en **posicion**.
terminoTSK RetornarTerminoC (int posicion)	Función que retorna el terminoTSK del consecuente de la regla que se encuentra en **posicion**.
void AgregarTerminoA (termino &ter)	Función que agrega un *termino* al antecedente de la regla.
void AgregarTerminoC (terminoTSK &ter)	Función que agrega un *termino* al consecuente de la regla.

Tabla 4.65. Métodos Públicos de la Clase `ReglaTSK`

Terminos Antecedente	Almacena los términos del antecedente.
TerminosTSK Consecuente	Almacena los términos del consecuente.

Tabla 4.66. Atributos Privados de la Clase `ReglaTSK`

4.5.4. Sistemas de Lógica Difusa

4.5.4.1. Clase `SLD_Mamdani`

Clase que define un Sistema de Lógica Difusa Tipo Mamdani. Su implementación se encuentra en los archivos *SLD.h* y *SLD.cpp*. La Tabla 4.67 muestra los métodos públicos; la Tabla 4.68 muestra los atributos privados.

SLD_Mamdani(wxArrayInt &in, wxArrayInt &out, ArregloDeVariablesL &arr)	Función Constructora. Define un Sistema de Lógica Difusa tipo Mamdani, un arreglo con las *Variables Lingüísticas* a emplear, y unos arreglos con las posiciones en el arreglo de las Variables de Entrada y las de Salida.
~SLD_Mamdani ()	Función Destructora
void AddRegla (Regla rule)	Función que añade una regla a la Base de Reglas.
Regla GetRegla (int posicion)	Función que retorna regla de la **posicion**.
void SetBaseReglas (BaseDeReglas Base)	Función que define una Base de Reglas para el SLD.
BaseDeReglas GetBaseReglas ()	Función que retorna la Base de Reglas.
bool SetDifusor (unsigned int pos_ent, difusor *dif)	Función que define el difusor a emplear en la Variable lingüística de entrada **pos_ent**.
bool SetConcresor (unsigned int pos_sal, concresor *conc)	Función que define el concresor a emplear en la Variable lingüística de salida **pos_sal**.
difusor *GetDifusor (unsigned int pos)	Función que retorna el Difusor de la posición pos.
concresor *GetConcresor (unsigned int pos)	Función que retorna el Concresor de la posición pos.
ArrayOfDoubles Calcular (ArrayOfDoubles ent)	Función que calcula las salidas para un conjunto de entradas. • Se realiza la fuzzyficación • Se calcula la consistencia entre las entradas y las reglas • Se agregan Arreglos de conjuntos difusos, tantos como salidas haya • Se calculan los conjuntos para luego calcular los recortes para cada regla • Si la Regla se Activa:

	Agregar al arreglo de salidas las variables • Se calcula la agregación • Se calcula la salida según el arreglo de Concresores
int VerificarIntegridad ()	Función que devuelve un entero que indica el número del error.
wxArrayInt RetornarEntradas ()	Función que retorna las *Variables Lingüísticas* de Entrada.
wxArrayInt RetornarSalidas ()	Función que retorna las *Variables Lingüísticas* de Salida.
ArregloDeVariablesL RetornarVL ()	Función que retorna el Arreglo de *Variables Lingüísticas* que se está empleando.
int RetornarPosDif (int var_ling)	Función que retorna la posición de la Variable lingüística en el Conjunto de Difusores. Si no está retorna -1.
int RetornarPosConc (int var_ling)	Función que retorna la posición de la Variable lingüística en el Conjunto de Concresores. Si no está retorna -1.

Tabla 4.67. Métodos Públicos de la Clase `SLD_Mamdani`

wxArrayInt Entradas	Arreglos que definen la posición de las entradas y salidas en el arreglo de *Variables Lingüísticas*.
wxArrayInt Salidas	Arreglos que definen la posición de las entradas y salidas en el arreglo de *Variables Lingüísticas*.
ArregloDeDifusores ArregloDifusores	Guardan los conjuntos de los difusores luego de entrar el valor Crisp de la variable.
ArregloDeConcresores ArregloConcresores	Guardan los conjuntos de los concresores luego de entrar el valor *crisp* de la variable.
BaseDeReglas BR	Base de Reglas del Sistema de Lógica Difusa.
ArregloDeVariablesL VL	Arreglo de *Variables Lingüísticas*.
ArrayOfDoubles entradas	Almacena los valores actuales de las variables de entrada y salida.
ArrayOfDoubles salidas	Almacena los valores actuales de las variables de entrada y salida.

Tabla 4.68. Atributos Privados de la Clase `SLD_Mamdani`

4.5.4.2. Clase `SLD_TSK`

Clase que define un Sistema de Lógica Difusa Tipo TSK. Su implementación se encuentra en los archivos *SLD.h* y *SLD.cpp*. La Tabla 4.69 muestra los métodos públicos; la Tabla 4.70 muestra los atributos privados.

SLD_TSK (wxArrayInt &in, wxArrayInt &out, ArregloDeVariablesL &arr)	Función Constructora. Define un Sistema de Lógica Difusa tipo TSK, un arreglo con las *Variables Lingüísticas* a emplear, y unos arreglos con las posiciones en el arreglo de las Variables de Entrada y las de Salida.
~SLD_TSK ()	Función Destructora
void AddRegla (ReglaTSK rule)	Función que añade una regla a la Base de Reglas a emplear.
ReglaTSK GetRegla (int posicion)	Función que retorna regla de ***posicion***.
void SetBaseReglas (BaseDeReglasTSK Base)	Función que define Base de Reglas.
BaseDeReglasTSK GetBaseReglas ()	Función que retorna Base de Reglas.
bool SetDifusor (unsigned int pos_ent, difusor *dif)	Función que define el difusor a emplear en la Variable lingüística de entrada *pos_ent*.
difusor *GetDifusor (unsigned int pos)	Función que retorna el Difusor de la posición pos.
ArrayOfDoubles Calcular (ArrayOfDoubles ent)	Función que calcula las salidas para un conjunto de entradas. Se efectúa la Fuzzyficación. • Se calcula la consistencia entre las entradas y las reglas. • Se agregan las salidas de acuerdo con un promedio ponderado de la consistencia: • Se calculan los consecuentes de acuerdo con las ecuaciones de las reglas • Se calculan las salidas para cada regla de acuerdo con los polinomios de los Términos tipo TSK y se ponderan por la consistencia de cada regla.
int VerificarIntegridad ()	Función que devuelve un entero que indica el número del error.
wxArrayInt RetornarEntradas ()	Función que retorna las *Variables Lingüísticas* de Entrada.
wxArrayInt RetornarSalidas ()	Función que retorna las *Variables Lingüísticas* de Salida.
ArregloDeVariablesL RetornarVL ()	Función que retorna el Arreglo de *Variables Lingüísticas* que se está empleando.
int RetornarPosDif (int var_ling)	Función que retorna la posición de la Variable lingüística en el Conjunto de Difusores. Si no está retorna -1.

Tabla 4.69. Métodos Públicos de la Clase `SLD_TSK`

wxArrayInt Entradas	Arreglos que definen la posición de las entradas y salidas en el arreglo de *Variables Lingüísticas*.

wxArrayInt Salidas	Arreglos que definen la posición de las entradas y salidas en el arreglo de *Variables Lingüísticas*.
ArregloDeDifusores ArregloDifusores	Guardan los conjuntos de los difusores luego de entrar el valor Crisp de la variable.
BaseDeReglasTSK BR	Base de Reglas del Sistema de Lógica Difusa.
ArregloDeVariablesLVL	Arreglo de *Variables Lingüísticas*.
ArrayOfDoubles entradas	Almacena los valores actuales de las variables de entrada y salida.
ArrayOfDoubles salidas	Almacena los valores actuales de las variables de entrada y salida.

Tabla 4.70. Atributos Privados de la Clase `SLD_TSK`

4.5.5. Agrupamiento

4.5.5.1. Clase `clustering`

Clase que define el Agrupamiento por el método *C-Means*. Su implementación se encuentra en los archivos *Clustering.h* y *Clustering.cpp*. La Figura 4.11 muestra el diagrama de herencias. La Tabla 4.71 muestra los métodos públicos; la Tabla 4.72muestra los atributos públicos; a Tabla 4.73 muestra los métodos privados; la Tabla 4.74 muestra los atributos privados.

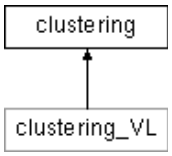


Figura 4.11. Diagrama de herencias de la clase `clustering`

clustering ()	Función Constructora. Se asignan valores por defecto a los nombres de los archivos y se asignan valores *false* para no emplear archivos medios ni finales.
virtual ~clustering ()	Función Destructora
bool Calcular (DoubleArray equis=1, int no_grupos=2, int CritPar=3, double tolU=0.001, double tolV=0.001, double tolJ=0.001, int itermax=50, int m=2)	Función para Calcular los Valores de los Centros y las matrices. Retorna false si no se pudo abrir alguno de los archivos. • Si se emplean archivos, se limpian y se extracta el valor de la matriz X, si no, se toma el valor del Arreglo pasado como parámetro. • Se inicializa U como matriz aleatoria.

	• Se calculan U, V y J. Y se repite esta operación hasta que se cumpla el criterio de parada ó hasta que se alcance le número máximo de iteraciones del ciclo.
DoubleArray RetornaX ()	Función para retornar X.
DoubleArray RetornaU ()	Función para retornar U.
double RetornaJ ()	Función para retornar J.
DoubleArray RetornaV ()	Función para retornar V.
void EmplearArchivos (bool arch=false)	Función para seleccionar si se desea emplear archivos o no.
void NombreEntrada (wxString ent)	Función que nombra el archivo de entrada de X.
void NombreArchivoU (wxString ent)	Función que nombra el archivo de salida de U.
void NombreArchivoV (wxString ent)	Función que nombra el archivo de salida de V.
void NombreArchivoJ (wxString ent)	Función que nombra el archivo de salida de J.
void EmplearArchivosMedios (bool arch=false)	Función para seleccionar si se desea emplear archivos medios o no.
void NombreArchivoUMedio (wxString ent)	Función que nombra el archivo medio de salida de U.
void NombreArchivoVMedio (wxString ent)	Función que nombra el archivo medio de salida de V.
void NombreArchivoJMedio (wxString ent)	Función que nombra el archivo medio de salida de J.
bool GuardarMatriz (DoubleArray &mat, wxString arch, bool append=0)	Función Intermedia que Guarda una matriz en un archivo determinado.
void LimpiarArchivo (wxString arch)	Función Intermedia que limpia los archivos.

Tabla 4.71. Métodos Públicos de la Clase `clustering`

unsigned int c	Cantidad de Grupos (Clusters) para cada variable.
DoubleArray X	Matriz X de los casos a agrupar *(p*n)*. *n* es el número de casos, *p* es la cantidad de variables.
DoubleArray V	Matriz V de los centros de los grupos *(p*c)*.
int M	Valor del exponente de cálculo para la *Función de Pertenencia*.
bool calculado	Indica si ya se ha ejecutado la función calcular.
bool archivos	Indica si se desea emplear archivos para extractar y guardar las matrices.
bool archivos_medios	Indica si se desea emplear archivos para guardar las matrices medias de cálculo.

Tabla 4.72. Atributos Públicos de la Clase `clustering`

void CalcularU ()	Función para calcular el valor de la matriz U.
void CalcularV ()	Función para calcular el valor de la matriz V.
void CalcularJ ()	Función para calcular el valor de J.
bool parada ()	Función que evalúa si el criterio de parada se cumple. Evalúa que el valor anterior de U, V , J o U, V y J menos el actual sea menor o igual que la tolerancia.

Tabla 4.73. Métodos Privados de la Clase `clustering`

wxString ArchivoX	Variables para los nombres de los archivos Finales de resultado de las Matrices.
wxString ArchivoU	Variables para los nombres de los archivos Finales de resultado de las Matrices.
wxString ArchivoV	Variables para los nombres de los archivos Finales de resultado de las Matrices.
wxString ArchivoJ	Variables para los nombres de los archivos Finales de resultado de las Matrices.
wxString ArchivoUMedio	Variables para los nombres de los archivos Medios de resultado de las Matrices.
wxString ArchivoVMedio	Variables para los nombres de los archivos Medios de resultado de las Matrices.
wxString ArchivoJMedio	Variables para los nombres de los archivos Medios de resultado de las Matrices.
int cant_itera	Variable para preservar la cantidad actual de iteraciones.
double J	Valor de J.
int CriterioParada	Valor del Criterio de Parada a Emplear. 0: Tolerancia de U. 1: Tolerancia de V. 2: Tolerancia de J. 3: Tolerancias de U, V y J.
int max_iteraciones	Variable para preservar la cantidad máxima de iteraciones.
double toleranciaU	Tolerancia para U, V y J.
double toleranciaV	Tolerancia para U, V y J.
double toleranciaJ	Tolerancia para U, V y J.
DoubleArray U	Matriz U.
DoubleArray Uant	Matriz U del paso anterior.
DoubleArray Vant	Matriz V del paso anterior.
double JantV	Valor J del paso anterior.

Tabla 4.74. Atributos Privados de la Clase `clustering`

4.5.5.2. Clase `clustering_VL`

Clase heredada que define la creación de *Variables Lingüísticas* agrupadas por C-Means. Su implementación se encuentra en los archivos *Clustering.h* y *Clustering.cpp*. La Tabla 4.75 muestra los métodos públicos; la Tabla 4.76 muestra los atributos privados.

clustering_VL ()	Función Constructora. Se asignan valores por defecto al nombre del archivo para almacenar las *Variables Lingüísticas*, cantidad de puntos de evaluación (30) y se asigna valor de *false* para no normalizar los términos.
~clustering_VL ()	Función Destructora
bool CrearVL ()	Función para Crear las *Variables Lingüísticas*. Retorna falso si no se había realizado el Cálculo de *C-Means*. • Se busca el menor y mayor valor de xi para cada variable y se toman como límites del universo. • Se calcular una matriz auxiliar *Xaux* de acuerdo con la cantidad de puntos de cálculo definida. • Se crean las *Variables Lingüísticas* cada una con c términos con centro en v_i. La *Función de Pertenencia* se calcula de acuerdo con la ecuación $$\mu_{ik} = \sum_{j=1}^{cant\_puntos} \left[\frac{d(x_k, v_i)}{d(x_k, v_j)} \right]^{\frac{2}{m-1}}$$
ArregloDeVariablesL RetornaVL ()	Función para retornar las *Variables Lingüísticas*.
void SetCantidadIntervalos (int cant)	Función que fija la cantidad de intervalos para calcular las *Variables Lingüísticas*.
void Normalizar (bool norm=true)	Función que fija si se deben normalizar o no los términos.
void NombreArchivoVL (wxString ent)	Función que nombra el el archivo de salida de *Variables Lingüísticas*.

Tabla 4.75. Métodos Públicos de la Clase `clustering_VL`

ArregloDeVariablesL VarLin	Arreglo de *Variables Lingüísticas* que se crea.
int cant_puntos	Cantidad de puntos de evaluación en el universo para crear la *Variable Lingüística*.
wxString ArchivoVL	Variables para los nombres de los archivos donde se almacenan las *Variables Lingüísticas* creadas.
bool normal	Indica si los Términos de las variables Lingüísticas creadas se deben normalizar o no.

Tabla 4.76. Atributos Privados de la Clase `clustering_VL`

4.5.6. Ordenamiento

4.5.6.1. Clase `ranking`

Clase abstracta que define ordenamiento. Se pasa un Arreglo de conjuntos Difusos y retorna un arreglo ordenado. Su implementación se encuentra en los archivos *Ranking.h* y *Ranking.cpp*. La Figura 4.12 muestra el diagrama de herencias. La Tabla 4.76 muestra los métodos públicos.

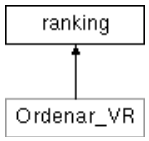


Figura 4.12. Diagrama de herencias de la clase `ranking`

ranking ()	Función Constructora.
~ranking ()	Función Destructora.
virtual ArregloDeConjuntosDifusos Ordenar (ArregloDeConjuntosDifusos &array) =0	Operación que ordena un Arreglo De Conjuntos Difusos Ordenado y lo retorna como otro.

Tabla 4.77. Métodos Públicos de la Clase `ranking`

4.5.6.2. Clase `Ordenar_VR`

Ranking por el *Valor Representativo* de los conjuntos difusos. Su implementación se encuentra en los archivos *Ranking.h* y *Ranking.cpp*. La Tabla 4.78 muestra los métodos públicos; la Tabla 4.79 muestra los atributos privados.

Ordenar_VR ()	Función Constructora. Fija el valor del optimismo β y de *r* de α^r del *Valor Representativo*.
void FijarParametros (double optimismo, double ere)	Función para fijar las variables de cálculo del *Valor Representativo*.

Tabla 4.78. Métodos Públicos de la Clase `Ordenar_VR`

double opt	Valores de las variables de cálculo del *Valor Representativo*.
double r	Valores de las variables de cálculo del *Valor Representativo*.

Tabla 4.79. Atributos Privados de la Clase `Ordenar_VR`

5. EJEMPLOS

En esta sección se encuentran algunos ejemplos para la creación de Conjuntos Difusos y Operaciones. Desde la sección 5.1 hasta la sección 5.4 se encuentran los ejemplos de cómo crear y manejar las clases auxiliares básicas que se emplean para manipular, operar y modelar Sistemas de Lógica Difusa, así como Conjuntos Difusos.

A continuación se encuentra una breve descripción de las potencialidades que la librería permite, no todas ellas tienen un ejemplo en esta sección:

- Conjuntos Difusos: Creación, Modificación, y funciones para manipular y retornar la *Función de Pertenencia*, los *α-cortes*, el *Nombre*, y el *Universo*. Creación de *α-cortes* a partir de unos puntos determinados de la *Función de Pertenencia*, creación de la *Función de Pertenencia* a partir de los *α-cortes*, determinar si el conjunto Difuso es Convexo, Normal o Número Difuso, hallar el *Valor Representativo* del Conjunto Difuso, determinar si el *Universo* es Válido.
- Operar Conjuntos Difusos: Calcular operaciones uniarias en las que se retorna un Conjunto Difuso, calcular operaciones binarias en las que se retorna un Conjunto Difuso, calcular operaciones entre Conjuntos Difusos en las que se retorna un valor *double* o número *crisp*, calcular el soporte de un Conjunto Difuso en el que se retorna una matriz de *doubles* (*DoubleArray*) que contiene los intevalos del *α-corte* cero, calcular si un Conjunto Difuso es sub-conjunto de otro y ordenar un arreglo de Conjuntos Difusos de acuerdo con su *Valor Representativo*.
- Variables Lingüísticas: Creación, y funciones para manipular y retornar *Función de Pertenencia*, los *α-cortes*, el *Nombre*, y el *Universo*. Determinar si el *Universo* es Válido. Para modificar los *Términos de Variable* se modifica cada conjunto en el arreglo de Conjuntos Difusos de la Variable.

- Reglas: Crear, agregar y modificar Reglas tipo Mamdani y TSK. Agregación de Términos de Conjunción de las reglas.
- Base de Reglas: Crear, agregar y modificar Bases de reglas tipo Mamdani y TSK. Términos de Conjunción de las reglas.
- Sistemas de Lógica Difusa: Creación y Modificación de Sistemas de Lógica Difusa tipo Mamdani y TSK. Agregación de Variables Lingüísticas, Bases de Reglas, elección de concresores y difusores según el caso, Cálculo de salidas según determinadas entradas.
- Agrupamiento: Selección de cantidad de grupo, ingreso de matriz de datos para una cantidad definida de Variables como parámeto o mediante el empleo de archivos, seleccionar si se van a emplear de archivos de origen y destino, seleccionar si se van a emplar archivos que almacenen los estados intermedios, modificación de nombres de los archivos.

5.1. Manejo de elementos de tipo Double

Aquí se pueden ver ejemplos para el manejo de objetos de la clase `Double` que es empleada como auxiliar para crear wxArray de números *doubles*.

5.1.1. Crear y modificar un `Double`

```
//Se declara un Double numero con valor 2.644
Double numero(2.644);
//Se declara un Double numero2 con valor 2.644
Double numero2=2.644;
//Se cambia el valor de numero a 3.77
numero=Double(3.77);
```

5.1.2. Recuperar el valor de un Double

```
//Se asigna el valor de un Double numero al double valor
double valor=numero.Get();
```

5.2. Manejo de elementos de tipo ArrayOfDoubles

A continuación se encuentran ejemplos para un vector de números Doubles.

5.2.1. Crear y agregar elementos a un ArrayOfDoubles

```
//Se crea un ArrayOfDoubles arreglo
ArrayOfDoubles arreglo;
//Se agrega un elemento de valor 2.5 a arreglo
arreglo.Add(Double(2.5));
```

5.2.2. Borrar elementos de un ArrayOfDoubles

```
//Se borra un elemento en la posición 5 de arreglo
arreglo.RemoveAt(5);
```

5.2.3. Limpiar todos los elementos de un ArrayOfDoubles

```
//Se borran todos los elementos de arreglo
arreglo.Clear();
```

5.2.4. Recuperar elementos de un ArrayOfDoubles

```
//Se asigna el valor de la posición 5 de arreglo al double num
double num=arreglo.Item(5).Get();
```

5.3. Agregar elementos a un DoubleArray

Este es un ejemplo de cómo crear y manejar matrices de números *doubles*.

5.3.1. Crear y agregar elementos a un DoubleArray

```
//Se crea un DoubleArray arreglo
DoubleArray arreglo=3;
//Se agregan tres elementos a un ArrayOfDoubles auxiliar aux para agregar
posteriormente una columna a arreglo
arr.Add(Double(2.5));
arr.Add(Double(3.2));
arr.Add(Double(4.3));
arreglo.Add(arr);
```

5.3.2. Borrar elementos de un DoubleArray

```
//Se borran un elemento en la columna 5 de arreglo
arreglo.RemoveAt(5);
```

5.3.3. Limpiar todos los elementos de un DoubleArray

```
//Se borran todos los elementos de arreglo
arreglo.Clear();
```

5.3.4. Recuperar elementos de un DoubleArray

```
//Se asigna el valor de la columna 5, fila 2 de arreglo al double num
double num=arreglo.Get(5,2).Get();
//Se asigna el valor de la columna 5 de arreglo al ArrayOfDoubles arr
ArrayOfDoubles arr=arreglo.GetColumn(5);
//Se asigna el valor de la fila 2 de arreglo al ArrayOfDoubles arr
ArrayOfDoubles arr=arreglo.GetRow(2);
```

5.4. Crear un α-corte

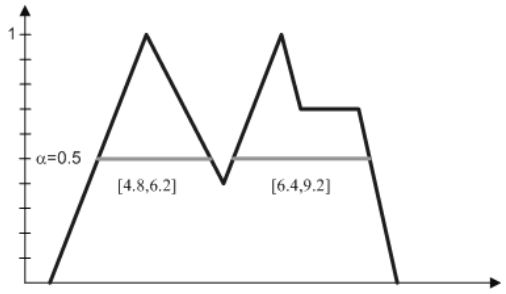


Figura 5.1. Ejemplo α-corte

La Figura 3.2. muestra el α-corte de nivel 0.5 cuyos intervalos son [4.8,6.2] y [6.4,9.2]. A continuación se encuentran ejemplos de cómo manipular y crear α-cortes.

5.4.1. Crear y agregar elementos a un α-corte

```
//Se declara un arreglo de alfa-corte arregloAC
ArregloDeAlfacortes arregloAC;
//Se declara un alfa-corte alfa
alfa_corte alfa;
//Se asigna el nivel de alfa=0.5
alfa.DefinirNivel(0.5);
//Se agregan dos intervalos a alfa: [4.8,6.2] y [6.4,9.2]
alfa.AgregarIntervalo(4.8,6.2);
alfa.AgregarIntervalo(6.4,9.2);
```

5.4.2. Recuperar elementos de un α-corte

```
//Se asigna la cantidad de intervalos en el alfa-corte alfa al double num
double num=alfa.RetornarCantidadIntervalos();
//Se eliminan los intervalos repetidos en el alfa-corte alfa
alfa.QuitarRepetido();
//Se asigna el arreglo de intervalos del alfa-corte alfa al DoubleArray
arr
DoubleArray arr; alfa.RetornarIntervalos(arr);
```

5.5. Crear un Conjunto Difuso

5.5.1. Crear un Conjunto Difuso introduciendo la Función de Pertenencia

El siguiente ejemplo muestra cómo crear un conjunto difuso tipo triángulo, cuyas cotas del Universo son [2,3] y las coordenadas de la *Función de Pertenencia* son {(2,0),(2.3,1),(3,0)}

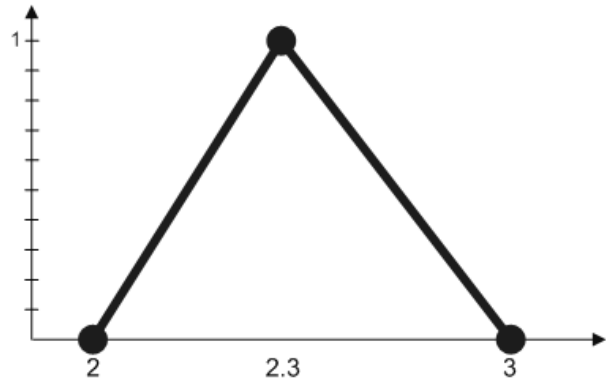


Figura 5.2. Ejemplo Conjunto Difuso (f.p.)

```
//Se declara un arreglo de Conjuntos Difusos arregloCD
ArregloDeConjuntosDifusos arregloCD;
//Se declara el Conjunto Difuso auxiliar
fs_cont_r auxiliar;
//Se declara una matriz arraux de 2 filas
DoubleArray arraux=2;
//Se declara un vector arr
ArrayOfDoubles arr;
//Se nombra el Conjunto Difuso con el wxString "Triangulo 1"
auxiliar.fsNombrarUniverso("Triangulo 1");
//Se ingresan las cotas del Universo: 2 es el límite inferior, 6 el
superior.
auxiliar.fsDefinirUniverso(2,3);
//Se agregan las coordenadas del punto (2,0) a arr
arr.Add(Double(2));
arr.Add(Double(0));
//Se agrega la columna con coordenadas del punto (2,0) a arraux
arraux.Add(arr);
```

```
//Se borra el contenido del arreglo arr
arr.Clear();
//Se agregan las coordenadas del punto (2.3,1) a arr
arr.Add(Double(2.3));
arr.Add(Double(1));
//Se agrega la columna con coordenadas del punto (2.3,1) a arraux
arraux.Add(arr);
//Se borra el contenido del arreglo arr
arr.Clear();
//Se agregan las coordenadas del punto (3,0) a arr
arr.Add(Double(3));
arr.Add(Double(0));
//Se agrega la columna con coordenadas del punto (3,0) a arraux
arraux.Add(arr);
//Se borra el contenido del arreglo arr
arr.Clear();
//Se asigna el contenido de la matriz arraux a la Función de Pertenencia
del Conjunto
auxiliar.fpDefinir(arraux);
//Se calculan los alfa-cortes de acuerdo con la Función de Pertenencia
del Conjunto
auxiliar.acCalcularAC();
//Se añade el Conjunto auxiliar al Arreglo de Conjuntos Difusos ArregloCD
ArregloCD.Add(auxiliar);
//Se borran los alfa-cortes del conjunto para poder usar auxiliar para
crear un nuevo conjunto difuso
auxiliar.acBorrarTodos();
```

5.5.2. Crear un Conjunto Difuso introduciendo los α-cortes

El siguiente ejemplo muestra cómo crear un conjunto difuso, cuyas cotas del Universo son [0,9] y los *α-cortes* son: *{α=0,{(0,9)}; α=0.1,{(0.25,8.85)}; α=0.2,{(0.49,8.71)}; α=0.3,{(0.74,8.57)}; α=0.4,{(0.99,8.42)}; α=0.5,{(1.25,4.16), (4.74,8.28)}; α=0.6,{(1.5,3.83), (4.99,8.14)}; α=0.7,{(1.75,3.5), (5.25,6.5)}; α=0.8,{(2,3.16), (5.49,6.33)}; α=0.9,{(2.25,2.83), (5.75,6.16)}, α=1,{(2.5,2.5), (6, 6)}}*

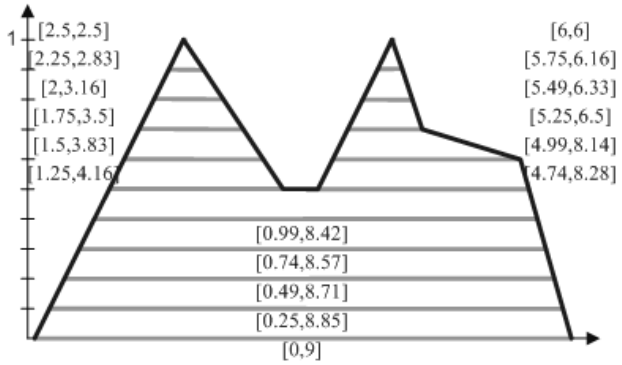


Figura 5.3. Ejemplo Conjunto Difuso (α-cortes)

```
//Se declara un arreglo de Conjuntos Difusos arregloCD
ArregloDeConjuntosDifusos arregloCD;
//Se declara el Conjunto Difuso auxiliar
fs_cont_r auxiliar;
//Se declara un vector arr
ArrayOfDoubles arr;
//Se nombra el Conjunto Difuso con el wxString "Conjunto"
auxiliar.fsNombrarUniverso("Conjunto");
//Se ingresan las cotas del Universo: 2 es el límite inferior, 6 el
superior.
auxiliar.fsDefinirUniverso(0,9);
//Se crea un ciclo para agregar alfa-cortes cada 0.1
for(unsigned int i=0; i<=1; i=i+0.1)
{
   //Se declara un alfa-corte auxiliar alfaux
   alfa_corte alfaux;
   //Se define el nivel del alfa-corte auxiliar alfaux
   alfaux.DefinirNivel(i);
   //De acuerdo con el alfa-corte i se agrega(n) el(los)
   intervalo(s)
   switch(i)
   {
      case 0:
         alfaux.AgregarIntervalo(0,9);
         break;
      case 0.1:
         alfaux.AgregarIntervalo(0.25,8.85);
         break;
      case 0.2:
         alfaux.AgregarIntervalo(0.49,8.71);
         break;
      case 0.3:
         alfaux.AgregarIntervalo(0.74,8.57);
         break;
      case 0.4:
         alfaux.AgregarIntervalo(0.99,8.42);
         break;
      case 0.5:
         alfaux.AgregarIntervalo(1.25,4.16);
         alfaux.AgregarIntervalo(4.74,8.28);
         break;
      case 0.6:
         alfaux.AgregarIntervalo(1.5,3.83);
         alfaux.AgregarIntervalo(4.99,8.14);
         break;
      case 0.7:
         alfaux.AgregarIntervalo(1.75,3.5);
         alfaux.AgregarIntervalo(5.25,6.5);
         break;
      case 0.8:
         alfaux.AgregarIntervalo(2,3.16);
         alfaux.AgregarIntervalo(5.49,6.33);
         break;
      case 0.9:
         alfaux.AgregarIntervalo(2.25,2.83);
         alfaux.AgregarIntervalo(5.75,6.16);
         break;
```

```
      case 1:
         alfaux.AgregarIntervalo(2.5,2.5);
         alfaux.AgregarIntervalo(6,6);
         break;
   }
   //Se agrega el alfa-corte i al Conjunto Difuso auxiliar
   auxiliar.acAgregar(alfaux);
}
//Se genera la Función de Pertenencia del Conjunto Difuso auxiliar
auxiliar.acCalcularFP();
//Se añade el Conjunto auxiliar al Arreglo de Conjuntos Difusos ArregloCD
ArregloCD.Add(auxiliar);
```

5.5.3. Recuperar Características de un Conjunto Difuso por α-cortes

```
//Retorna true si el alfa-corte de nivel 0.2 existe para el Conjunto
Difuso auxiliar
auxiliar.acNivelExiste(0.2);
//Retorna los alfa-cortes del Conjunto Difuso auxiliar en el
ArregloDeAlfaCortes ArrAC
auxiliar.RetornarAlfacortes(ArrAC);
//Retorna la posición del alfa-corte de la posición a del
ArregloDeAlfaCortes
int a=auxiliar.acRetornarPosicion(0.2);
//Retorna el alfa-corte de la posición del ArregloDeAlfaCortes en el
DoubleArray da
//Se almacena en noAC la cantidad de alfa-cortes
int noAC=auxiliar.acRetornarCantidad();
```

5.5.4. Recuperar Características de un Conjunto Difuso por Función de Pertenencia

```
//Se almacena en noFP la cantidad de puntos de la Función de Pertenencia
int noPT=auxiliar.fpLeerNoPt();
//Retorna los alfa-cortes del Conjunto Difuso auxiliar en el
ArregloDeAlfaCortes ArrAC
//Función que retorna los puntos de la Función de Pertenencia en el
DoubleArray ArrFP.
DoubleArray ArrFP=auxiliar.fpLeer();
//Función que borra todos los puntos de la Función de Pertenencia en el
DoubleArray ArrFP.
auxiliar.fpBorrarTodos();
```

5.6. Creación de una Variable Lingüística

La Figura 5.4. ilustra el siguiente ejemplo para la creación de una *Variable Lingüística*. Las cotas del Universo son [0,9]. Tiene tres términos de Variable: Bajo, Medio y Alto.

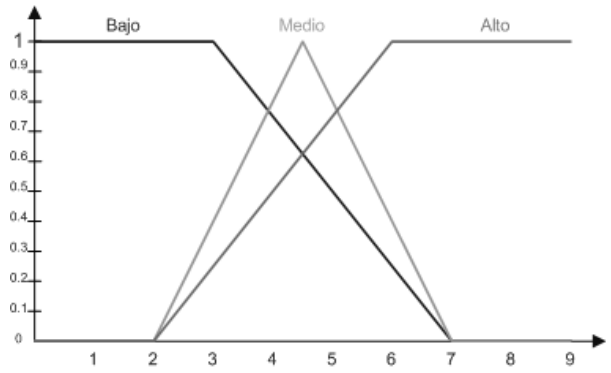


Figura 5.4. Ejemplo `Variable Lingüística`

```
//Se declara una Variable Lingüística var
VariableLinguistica var;
//Se declara un Arreglo de Conjuntos Difusos Auxiliar arrCD
ArregloDeConjuntosDifusos arrCD;
//Se declara un DoubleArray  univ en el que se almacenarán las cotas del
universo.
DoubleArray univ=2;
//Se declara un Conjunto Difuso auxiliar
fs_cont_r auxiliar;
//Se declara un DoubleArray  arraux para definir las Funciones de
Pertenencia.
DoubleArray arraux=2;
//Se declara un ArrayOfDoubles arr para agregar valores a arraux.
ArrayOfDoubles arr;
//Se nombra la Variable Lingüística Temperatura
var.NombrarVL(wxString("Temperatura"));
//Se define el universo [0,9]
arr.Add(Double(0));
arr.Add(Double(9));
univ.Add(arr);
arr.Clear();
//Se nombra el término de Variable Bajo
auxiliar.fsNombrarUniverso(wxString("Bajo"));
//Se define el universo para Bajo
auxiliar.fsDefinirUniverso(univ.Get(0,0).Get(),univ.Get(0,1).Get());
//Se ingresa la Función de Pertenencia de Bajo
arr.Add(Double(0));
arr.Add(Double(1));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(3));
arr.Add(Double(1));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(7));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(9));
arr.Add(Double(0));
arraux.Add(arr);
```

```
arr.Clear();
//Se agrega la Función de Pertenencia de Bajo
auxiliar.fpDefinir(arraux);
//Se Crean los alfa-cortes para la Función de Pertenencia de Bajo
auxiliar.acCalcularAC();
//Se agrega al Arreglo de Conjuntos Difusos Bajo
auxiliar.Add(arrCD);
//Se limpian los alfa-cortes para emplear de nuevo el Conjunto Difuso
auxiliar
auxiliar.acBorrarTodos();
arraux.Clear();
//Se nombra el término de Variable Medio
auxiliar.fsNombrarUniverso(wxString("Medio"));
//Se define el universo para Medio
auxiliar.fsDefinirUniverso(univ.Get(0,0).Get(),univ.Get(0,1).Get());
//Se ingresa la Función de Pertenencia de Medio
arr.Add(Double(0));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(2));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(4.5));
arr.Add(Double(1));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(7));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(9));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
//Se agrega la Función de Pertenencia de Medio
auxiliar.fpDefinir(arraux);
//Se Crean los alfa-cortes para la Función de Pertenencia de Medio
auxiliar.acCalcularAC();
//Se agrega al Arreglo de Conjuntos Difusos Medio
auxiliar.Add(arrCD);
//Se limpian los alfa-cortes para emplear de nuevo el Conjunto Difuso
auxiliar
auxiliar.acBorrarTodos();
arraux.Clear();
//Se nombra el término de Variable Alto
auxiliar.fsNombrarUniverso(wxString("Alto"));
//Se define el universo para Alto
auxiliar.fsDefinirUniverso(univ.Get(0,0).Get(),univ.Get(0,1).Get());
//Se ingresa la Función de Pertenencia de Alto
arr.Add(Double(0));
arr.Add(Double(0));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(2));
arr.Add(Double(0));
```

```
arraux.Add(arr);
arr.Clear();
arr.Add(Double(6));
arr.Add(Double(1));
arraux.Add(arr);
arr.Clear();
arr.Add(Double(9));
arr.Add(Double(1));
arraux.Add(arr);
arr.Clear();
//Se agrega la Función de Pertenencia de Alto
auxiliar.fpDefinir(arraux);
//Se Crean los alfa-cortes para la Función de Pertenencia de Alto
auxiliar.acCalcularAC();
//Se agrega al Arreglo de Conjuntos Difusos Alto
auxiliar.Add(arrCD);
//Se limpian los alfa-cortes para emplear
de nuevo el Conjunto Difuso auxiliar
auxiliar.acBorrarTodos();
arraux.Clear();
//Se agregan los Términos del Arreglo de Conjuntos Difusos arrCD a la
Variable Lingüística. Otra forma de hacerlo es al final de la creación de
cada Conjunto Difuso emplear var.AgregarTermino(auxiliar);
var.AgregarNTerminos(arrCD);
```

5.7. Operaciones Difusas

5.7.1. Cálculo de Operaciones Binarias

En este ejemplo se realiza el mínimo entre los Conjuntos Difusos *CD1* y *CD2* y se almacena en *CD3*. Se define como válidas las porciones del Universo que tienen en común y las que no.

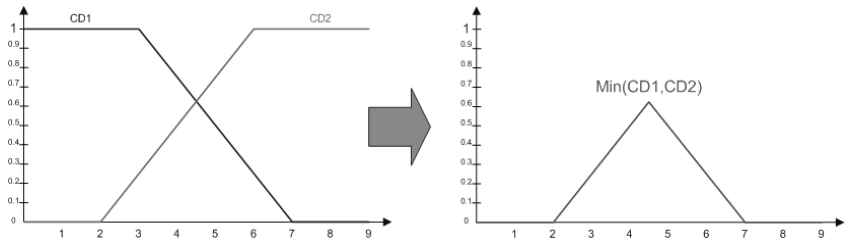


Figura 5.5. Ejemplo Operación Difusa Mínimo

```
//Se declara min de tipo minimo. Estas declaraciones y el cálculo de las
operaciones se realiza de la misma manera para maximo, suma, resta,
multiplicacion, interv_min (mínimo intervalar) e interv_max (máximo
intervalar).
minimo min;
//Se define la extrapolación de la operación de manera que se cubran los
universos de ambos Conjuntos Difusos
min.ExtrapolarUniverso(true);
```

```
//Se calcula el mínimo entre los Conjuntos Difusos CD1 y CD2 y el
resultado se almacena en el Conjunto Difuso CD3.
CD3=min.Calcular(CD1,CD2);
```

5.7.2. Cálculo de Operaciones Uniarias

En este ejemplo se calcula el complemento del Conjunto Difuso *CD1* y se almacena en *CD3*.

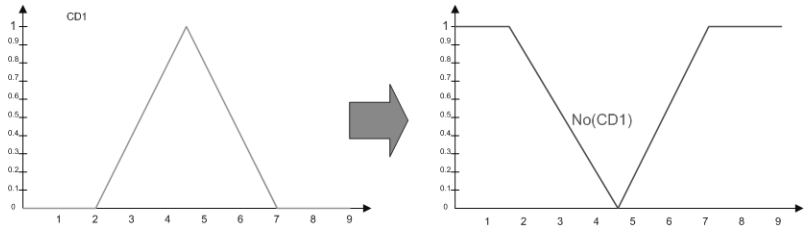


Figura 5.6. Ejemplo Operación Difusa Complemento

```
//Se declara complemento de tipo no. Estas declaraciones y el cálculo de
las operaciones se realiza de la misma manera para muy, mas_o_menos, mas,
algo, extremadamente, no_muy y normalizar.
no complemento;
//Se calcula el complemento para el Conjunto Difuso CD1 y el resultado se
almacena en el Conjunto
Difuso CD3.
CD3=complemento.Calcular(CD1);
//Se declara mult de tipo multiplicador.
multiplicador mult;
//Se define el factor a multiplicar por el Conjunto Difuso (0.2).
mult.Set(0.2);
//Se calcula el Conjunto Difuso CD3 modificando el Conjunto Difuso CD1
por el factor.
CD3=mult.Calcular(CD1);
```

5.7.3. Cálculo de Operaciones que retornan Valores

En este ejemplo se calcula la consistencia para el arreglo de Conjuntos Difusos *ArregloCD* y se almacena en *resultado*.

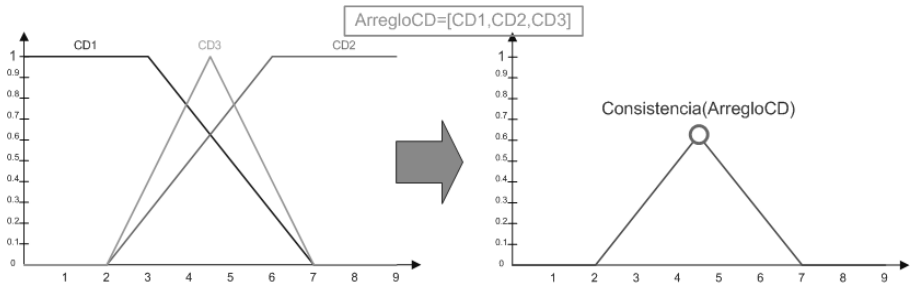


Figura 5.7. Ejemplo Operación Difusa Consistencia

```
//Se declara cons de tipo consistencia. Estas declaraciones y el cálculo
de las operaciones se realiza de la misma manera para cardinalidad,
subconjuntez y altura.
consistencia cons;
//Se calcula la consistencia para el Arreglo de Conjuntos Difusos
ArregloCD y el resultado se almacena en el double resultado.
double resultado=cons.Calcular(ArregloCD);
//Se declara pdist de tipo PDistancia.
PDistancia pdist;
//Se define el valor de P=2.
pdist.DefinirP(2);
//Se calcula la P-Distancia entre los dos primeros conjuntos del Arreglo
de Conjuntos Difusos ArregloCD y el resultado se almacena en el double
resultado.
double resultado=pdist.Calcular(ArregloCD);
//Se declara dist de tipo distVR.
distVR dist;
//Se define el valor de beta (optimismo)=0.5 y de r=2
dist.DefinirParametros(0.5,2);
//Se calcula la Distancia por medio del Valor Representativo entre los
dos primeros conjuntos del Arreglo de Conjuntos Difusos ArregloCD y el
resultado se almacena en el double resultado.
double resultado=dist.Calcular(ArregloCD);
```

5.7.4. Cálculo de otras Operaciones

5.7.4.1. Cálculo de Soporte

En este ejemplo se calcula el soporte para el Conjunto Difuso *CD1* y se almacena en *resultado*.

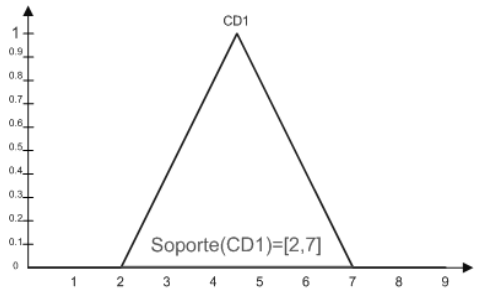


Figura 5.8. Ejemplo Operación Difusa Soporte

```
//Se declara sop de tipo soporte.
soporte sop;
//Se calcula el soporte para el Conjunto Difuso CD1 y el resultado se
almacena en el DoubleArray resultado.
DoubleArray resultado=sop.Calcular(CD1);
```

5.7.4.2. Cálculo de Subconjunto

En este ejemplo se indica si el Conjunto Difuso *CD1* es subconjunto de *CD2* y se almacena en el booleano *resultado*.

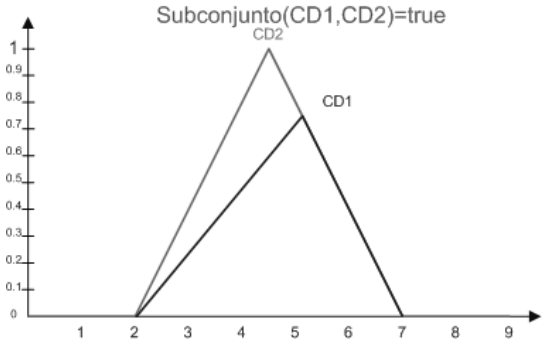


Figura 5.9. Ejemplo Operación Difusa Subconjunto

```
//Se declara sub de tipo subconjunto.
subconjunto sub;
//Se retorna true si el Conjunto Difuso CD1 es subconjunto de CD2 y false
en caso contrario.
bool resultado=sub.Calcular(CD1,CD2);
```

5.8. Empleo de Valor Representativo para Ordenar un Arreglo de Conjuntos Difusos

En este ejemplo se ordena el Arreglo de Conjuntos Difusos *ConjuntosDifusos* según el *Valor Representativo* calculado con los parámetros *opt (β)* y *ere (r para α)*.

```
//Se declara aux del tipo Ordenar_VR
Ordenar_VR aux;
//Sean las variables tipo double opt y ere definidas como el optimismo y
el exponente de alfa^r respectivamente. Se fijan los parámetros del Valor
Representativo
aux.FijarParametros(opt,ere);
//Se Ordena el Arreglo de Conjuntos Difusos ConjuntosDifusos
ConjuntosDifusos=aux.Ordenar(ConjuntosDifusos);
```

5.9. Creación de Difusores

5.9.1. Pi

En el siguiente ejemplo se crea un Conjunto Difuso tipo Pi para el número crisp 9.

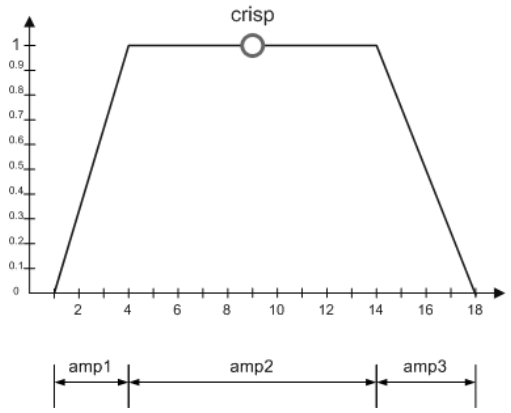


Figura 5.10. Ejemplo Difusor Pi

```
//Se declara un difusor pi. Para un singleton se recomienda que la
amplitud central sea 1*e-8
pi tr1;
//Se definen las amplitudes del trapecio pi como 3 para el segmento
izquierdo, 10 para el central y 4 para el derecho
tr1.DefinirAmplitudes(3,10,4);
//Se calcula un conjunto difuso a partir del numero crisp 9 y se almacena
en el Conjunto Difuso cd
fs_cont_r cd=tr1.Calcular(9);
```

5.9.2. Pi-Campana Cuadrática

En el siguiente ejemplo se crea un Conjunto Difuso tipo Pi-Campana con curvas cuadráticas para el número crisp 8. Según la Figura 5.12. amp1=2, amp2=3, amp3=4, amp4=2, amp5=2, f1=0.6 y f2=0.4, el número de α-cortes con que se calculará es 21.

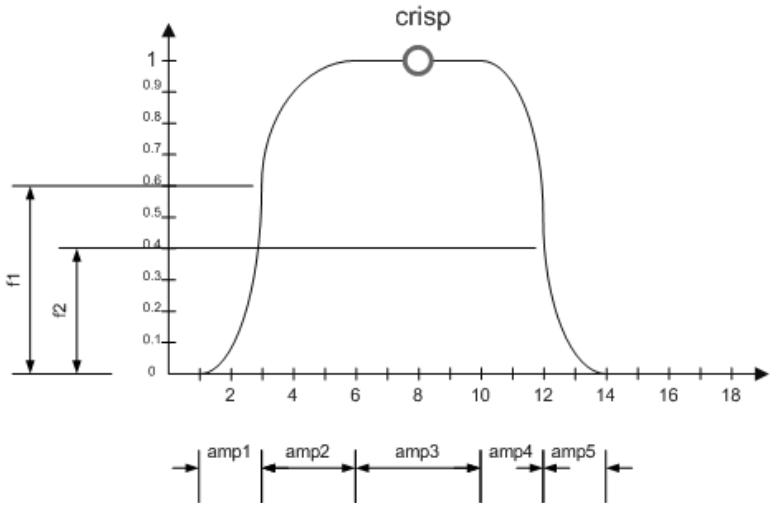


Figura 5.11. Ejemplo Difusor Pi-Campana Cuadrática

```
//Se declara un difusor pi_campana_cuadratica
pi_campana_cuadratica pi1;
//Se definen las amplitudes de las curvas cuadráticas
pi1.DefinirAmplitudes(2,3,4,2,2,.6,.4,21);
//Se calcula un conjunto difuso a partir del numero crisp 8 y se almacena
en el Conjunto Difuso cd
fs_cont_r cd=tr1.Calcular(8);
```

5.9.3. Pi-Campana Gaussiana

En el siguiente ejemplo se crea un Conjunto Difuso tipo Pi-Campana con curvas cuadráticas para el número crisp 8. Según la Figura 5.11, σ1=5. σ2=4 y amp3=4, el número de α-cortes con que se calculará es 21.

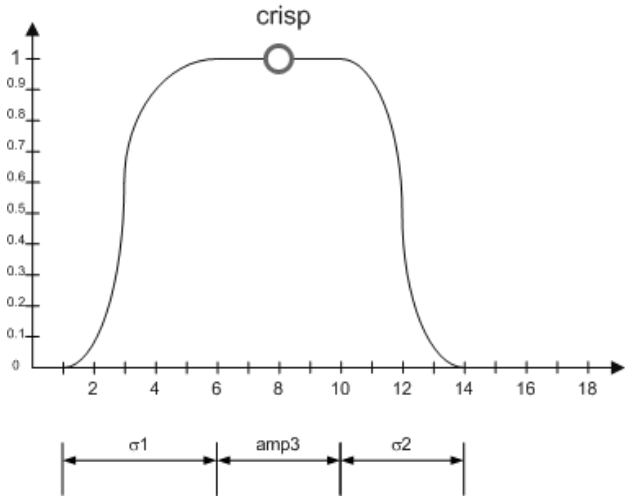


Figura 5.12. Ejemplo Difusor Pi-Campana Gaussiana

```
//Se declara un difusor pi_campana_e
pi_campana_e pi1;
//Se definen las amplitudes de las curvas cuadráticas
pi1.DefinirAmplitudes(5,4,4,21);
//Se calcula un conjunto difuso a partir del numero crisp 8 y se almacena
en el Conjunto Difuso cd
fs_cont_r cd=tr1.Calcular(8);
```

5.10. Creación de Concresores

Este es un ejemplo para la creación de los concresores `primer_maximo`, `ultimo_maximo`, `media_maximos` y `centro_gravedad`.

```
//Se crea el concresor primer máximo pm y se calcula un número crisp para
el Conjunto Difuso CD1 que se almacena en el double aux
primer_maximo pm;
double aux=pm.Calcular(CD1);
//Se crea el concresor último máximo um y se calcula un número crisp para
el Conjunto Difuso CD1 que se almacena en el double aux
ultimo_maximo um;
double aux=um.Calcular(CD1);
//Se crea el concresor media de máximos mm y se calcula un número crisp
para el Conjunto Difuso CD1 que se almacena en el double aux
media_maximos mm;
double aux=mm.Calcular(CD1);
//Se crea el concresor centro de gravedad cg y se calcula un número crisp
para el Conjunto Difuso CD1 que se almacena en el double aux
centro_gravedad cg;
double aux=cg.Calcular(CD1);
```

5.11. Creación de Reglas

5.11.1. Creación de Términos

Este ejemplo muestra cómo se crean los términos de conjunción del tipo ***A es αa***.

```
//Se crea un termino de tipo  SI Temperatura ES no Baja. Se introducen números enteros que representan la posición de Temperatura en un ArregloDeVariablesL (1) y  la posición del término Baja en el ArregloDeConjuntosDifusos de términos en Temperatura(1)
//Se define un apuntador neg a un modificador de tipo no
no *neg;
neg=new no;
//Se crea el término Temperatura es no Baja
termino uno=termino(1,neg,1);
//Se crea el término de variable de posición 2, sin modificador y con término de variable de posición 0.
termino dos=termino(2,NULL,0);
```

5.11.2. Creación de Términos TSK

Este ejemplo muestra cómo se crean los términos de conjunción del tipo ***B es f(entradas)***.

```
//Se crea un termino de tipo TSK:  SI Temperatura ES no Baja Y Presión es Alta ENTONCES f(Temperatura). f(Temperatura,Presión) es del tipo A0+A1* temp+A2*                                    temp^2+B0+B1*pres+B2*pres^2,con A0=2,A1=3,A2=2.2,B0=0,B1=0.44,B2=2.55
ArrayOfDoubles aod;
DoubleArray arr=3;
aod.Add(Double(2));
aod.Add(Double(3));
aod.Add(Double(2.2));
arr.Add(aod);
aod.Add(Double(0));
aod.Add(Double(0.44));
aod.Add(Double(2.55));
arr.Add(aod);
terminoTSK uno=terminoTSK(arr);
//Se crea el término TSK.
```

5.11.3. Creación de Reglas tipo Mamdani

En el siguiente ejemplo se definen Reglas tipo Mandami con Antecedentes y Consecuentes de tipo términos de Conjunción.

```
//Se definen Ant y Cons como los términos auxiliares tipo Mamdani para
antecedente y el consecuente.
Terminos Ant,Cons;
//Se crea una Base de reglas tipo Mamdani BR
BaseDeReglas BR;
//Se agregan dos términos a Ant y dos términos a Cons
Ant.Add(termino(0,NULL,0));
Ant.Add(termino(1,NULL,1));
Cons.Add(termino(3,NULL,2));
Cons.Add(termino(4,NULL,1));
//Se agregan los términos de Ant y Cons a los correspondientes
antecedente y consecuente de una Regla tipo Mamdani rule
Regla rule=Regla(Ant,Cons);
//Se agrega la regla rule a la BaseDeReglas BR
BR.Add(rule);
//Se limpian el Antecedente y el Consecuente para reutilizarlos creando
la regla que se muestra a continuación
Ant.Clear();
Cons.Clear();
Ant.Add(termino(2,NULL,1));
Ant.Add(termino(0,NULL,2));
Cons.Add(termino(3,NULL,1));
Cons.Add(termino(4,NULL,2));
//Se agrega la regla rule a la BaseDeReglas BR
rule=Regla(Ant,Cons);
BR.Add(rule);
//Se limpian el Antecedente y el Consecuente para reutilizarlos creando
la regla que se muestra a continuación
Ant.Clear();
Cons.Clear();
Ant.Add(termino(0,NULL,1));
Ant.Add(termino(1,NULL,0));
Ant.Add(termino(2,NULL,2));
Cons.Add(termino(3,NULL,0));
Cons.Add(termino(4,NULL,0));
//Se agrega la regla rule a la BaseDeReglas BR
rule=Regla(Ant,Cons);
BR.Add(rule);
//Se limpian el Antecedente y el Consecuente para reutilizarlos creando
otra regla
Ant.Clear();
Cons.Clear();
```

5.11.4. Creación de Reglas tipo TSK

En el siguiente ejemplo se definen Reglas tipo Mandami con Antecedentes de tipo términos de Conjunción y Consecuentes tipo términos TSK.

```
//Se crea una Base de reglas tipo TSK BR
BaseDeReglasTSK BR;
//Se definen Ant y Cons como los términos auxiliares tipo Mamdani y TSK
para antecedente y el consecuente, respectivamente.
```

```
Terminos Ant;
TerminosTSK Cons;
//Se crea un DoubleArray arr para la insersión de los términos TSK
DoubleArray arr=2;
//Se crea un ArrayOfDoubles aod para la insersión de las columnas de arr
ArrayOfDoubles aod;
//Se agregan dos términos a Ant
Ant.Add(termino(10,NULL,0));
Ant.Add(termino(11,NULL,1));
//Se agregan un término TSK de polinomios de grado cuatro para dos
variables de entrada y una variable de salida Ant
aod.Add(Double(-1.8));aod.Add(Double(0));
arr.Add(aod);
aod.Clear();
aod.Add(Double(-0.37));aod.Add(Double(-0.9));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.185));aod.Add(Double(0.8));
arr.Add(aod);
aod.Clear();
aod.Add(Double(-0.5));aod.Add(Double(0.001));
arr.Add(aod);
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
aod.Add(Double(0.1));aod.Add(Double(0.5));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.2));aod.Add(Double(0.8));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.3));aod.Add(Double(-0.3));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.4));aod.Add(Double(0.1));
arr.Add(aod);
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
//Se agregan los términos de Ant y Cons a los correspondientes
antecedente y consecuente de una Regla tipo TSK rule
rule=ReglaTSK(Ant,Cons);
//Se agrega la regla rule a la BaseDeReglas BR
BR.Add(rule);
Ant.Clear();
Cons.Clear();
Ant.Add(termino(10,NULL,2));
Ant.Add(termino(11,NULL,0));
//Se agregan un término TSK de polinomios de grado cuatro para dos
variables de entrada y una variable de salida Ant
aod.Add(Double(0));aod.Add(Double(-0.3));
arr.Add(aod);
aod.Clear();
aod.Add(Double(-0.1));aod.Add(Double(0.45));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.2));aod.Add(Double(1));
```

```
arr.Add(aod);
aod.Clear();
aod.Add(Double(-2));aod.Add(Double(1));
arr.Add(aod);
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
aod.Add(Double(0));aod.Add(Double(-1));
arr.Add(aod);
aod.Clear();
aod.Add(Double(1));aod.Add(Double(-1));
arr.Add(aod);
aod.Clear();
aod.Add(Double(1));aod.Add(Double(-1));
arr.Add(aod);
aod.Clear();
aod.Add(Double(1));aod.Add(Double(-1));
arr.Add(aod);
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
//Se agregan los términos de Ant y Cons a los correspondientes
antecedente y consecuente de una Regla tipo TSK rule
rule=ReglaTSK(Ant,Cons);
//Se agrega la regla rule a la BaseDeReglas BR
BR.Add(rule);
Ant.Clear();
Cons.Clear();
Ant.Add(termino(10,NULL,1));
Ant.Add(termino(11,NULL,2));
//Se agregan un término TSK de polinomios de grado cuatro para dos
variables de entrada y una variable de salida Ant
aod.Add(Double(0));aod.Add(Double(.95));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.7));aod.Add(Double(1));
arr.Add(aod);
aod.Clear();
aod.Add(Double(-0.3));aod.Add(Double(0.33));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.4));aod.Add(Double(0.44));
arr.Add(aod);
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
aod.Add(Double(-9));aod.Add(Double(9));
arr.Add(aod);
aod.Clear();
aod.Add(Double(9));aod.Add(Double(-9));
arr.Add(aod);
aod.Clear();
aod.Add(Double(8));aod.Add(Double(-8));
arr.Add(aod);
aod.Clear();
aod.Add(Double(0.55));aod.Add(Double(1));
arr.Add(aod);
```

```
aod.Clear();
Cons.Add(terminoTSK(arr));
arr.Clear();
//Se agregan los términos de Ant y Cons a los correspondientes
antecedente y consecuente de una Regla tipo TSK rule
rule=ReglaTSK(Ant,Cons);
//Se agrega la regla rule a la BaseDeReglas BR
BR.Add(rule);
Ant.Clear();
Cons.Clear();
```

5.12. Creación de Sistemas de Lógica Difusa

5.12.1. Creación de SLD tipo Mamdani

En el siguiente ejemplo se define un Sistema de Lógica Difusa tipo Mandami con entradas y salidas en el Arreglo de Variables Lingüísticas (*VariablesLinguisticas*) en las posiciones 0, 1 y 2, y 3 y 4 respectivamente, difusores tipo Pi, con amplitudes 0.4, 0 y 0.4 (Triángulo), concresores tipo primer máximo, la base de reglas a emplear se encuentra en el Arreglo de Bases de Reglas *BasesReglas* en la posición 2.

```
//Se declara un Arreglo de Sistemas de Lógica Difusa tipo Mamdani SLDM
ArraySLDM SLDM;
//Se declaran arreglos de enteros para almacenar las posiciones de las
Variables Lingüísticas de entrada y salida
wxArrayInt ent,sal;
//Se agregan las posiciones de las Variables Lingüísticas de entrada a
ent
ent.Add(0);ent.Add(1);ent.Add(2);
//Se agregan las posiciones de las Variables Lingüísticas de salida a sal
sal.Add(13);sal.Add(14);
//Se agregan un nuevo SLD tipo Mamdani al arreglo SLDM con las las
posiciones de las Variables Lingüísticas de entrada ent y de salida a sal,
además de el Arreglo de Variables Lingüísticas VariablesLinguisticas
SLDM.Add(SLD_Mamdani(ent,sal,VariablesLinguisticas));
//Se define la Base de Reglas para el SLD
recién agregado al arreglo SLDM en la posición 0 con la Base de
Reglas de la posición 2 del ArregloDeBaseDeReglas BasesReglas
SLDM.Item(0).SetBaseReglas(BasesReglas.Item(2));
//Se define un apuntador a un difusor tr1 tipo pi
pi *tr1;
//Se genera un nuevo difusor tr1
tr1=new pi;
//Se definen las amplitudes del difusor tr1
tr1->DefinirAmplitudes(0.4,0,0.4);
//Se define el difusor tr1 para la entrada 0
SLDM.Item(0).SetDifusor(0,tr1);
//Se genera un nuevo difusor tr1
tr1=new pi;
```

```
//Se definen las amplitudes del difusor tr1
tr1->DefinirAmplitudes(0.4,0,0.4);
//Se define el difusor tr1 para la entrada 1
SLDM.Item(0).SetDifusor(1,tr1);
//Se genera un nuevo difusor tr1
tr1=new pi;
//Se definen las amplitudes del difusor tr1
tr1->DefinirAmplitudes(0.4,0,0.4);
//Se define el difusor tr1 para la entrada 2
SLDM.Item(0).SetDifusor(2,tr1);
//Se define un apuntador a un concresor pm tipo primer_maximo
primer_maximo *pm;
//Se genera un nuevo concresor pm
pm=new primer_maximo;
//Se define el concresor tr1 para la salida 0
SLDM.Item(0).SetConcresor(0,pm);
//Se genera un nuevo concresor pm
pm=new primer_maximo;
//Se define el concresor tr1 para la salida 1
SLDM.Item(0).SetConcresor(1,pm);
ent.Clear();sal.Clear();
```

5.12.2. Creación de SLD tipo TSK

En el siguiente ejemplo se define un Sistema de Lógica Difusa tipo Mandami con entradas y salidas en el Arreglo de Variables Lingüísticas (*VariablesLinguisticas*) en las posiciones 0, 1 y 2, y 3 y 4 respectivamente, difusores tipo Pi, con amplitudes 1, 0 y 1 (Triángulo), la base de reglas a emplear se encuentra en el Arreglo de Bases de Reglas TSK *BasesReglasTSK* en la posición 0.

```
//Se declara un Arreglo de Sistemas de Lógica Difusa tipo TSK SLDTSK
ArraySLDTSK SLDTSK;
//Se declaran arreglos de enteros para almacenar las posiciones de las
Variables Lingüísticas de entrada y salida
wxArrayInt ent,sal;
//Se agregan las posiciones de las Variables Lingüísticas de entrada a
ent
ent.Add(0);ent.Add(1);ent.Add(2);
//Se agregan las posiciones de las Variables Lingüísticas de salida a sal
sal.Add(3);sal.Add(4);
//Se agregan un nuevo SLD tipo TSK al arreglo SLDTSK con las las
posiciones de las Variables Lingüísticas de entrada ent y de salida a sal,
además de el Arreglo de Variables Lingüísticas VariablesLinguisticas
SLDTSK.Add(SLD_TSK(ent,sal,VariablesLinguisticas));
//Se define la Base de Reglas para el SLD recién agregado al arreglo
SLDTSK en la posición 0 con la Base de Reglas de la posición 0 del
ArregloDeBaseDeReglasTSK BasesReglasTSK
SLDTSK.Item(0).SetBaseReglas(BasesReglasTSK.Item(0));
//Se define un apuntador a un difusor tr1 tipo pi
pi *tr1;
```

```
//Se genera un nuevo difusor tr1
tr1=new pi;
//Se definen las amplitudes del difusor tr1
tr1->DefinirAmplitudes(1,0,1);
//Se define el difusor tr1 para la entrada 0
SLDTSK.Item(0).SetDifusor(0,tr1);
//Se genera un nuevo difusor tr1
tr1=new pi;
//Se definen las amplitudes del difusor
tr1
tr1->DefinirAmplitudes(1,0,1);
//Se define el difusor tr1 para la entrada 1
SLDTSK.Item(0).SetDifusor(1,tr1);
//Se genera un nuevo difusor tr1
tr1=new pi;
//Se definen las amplitudes del difusor tr1
tr1->DefinirAmplitudes(1,0,1);
//Se define el difusor tr1 para la entrada 1
SLDTSK.Item(0).SetDifusor(2,tr1);
```

5.13. Agrupamiento

```
//Se emplearán los archivos por defecto y archivos medios y el número de
puntos por defecto para crear las Variables Lingüísticas. Se normalizarán
los conjuntos resultado y se crearán 3 grupos para cada variable. Se crea
clust de tipo clustering_VL
Clustering_VL clust;
clust.Normalizar();
clust.EmplearArchivos (true);
clust.EmplearArchivosMedios(true);
//Se calcula el Agrupamiento por el método C-Means
clust.Calcular(DoubleArray(1),3);
//Se crean las Variables Lingüísticas
clust.CrearVL();
//Se retornan las Variables Lingüísticas en el Arreglo De Variables
Lingüísticas array
ArregloDeVariablesL array=clust.RetornarVL();
```

5.14. Ejemplo de Aplicación de un SLD Mamdani y Algunos Resultados

En el siguiente ejemplo se muestran algunos resultados obtenidos para un Sistema de Lógica Difusa de 3 entradas y 2 salidas.

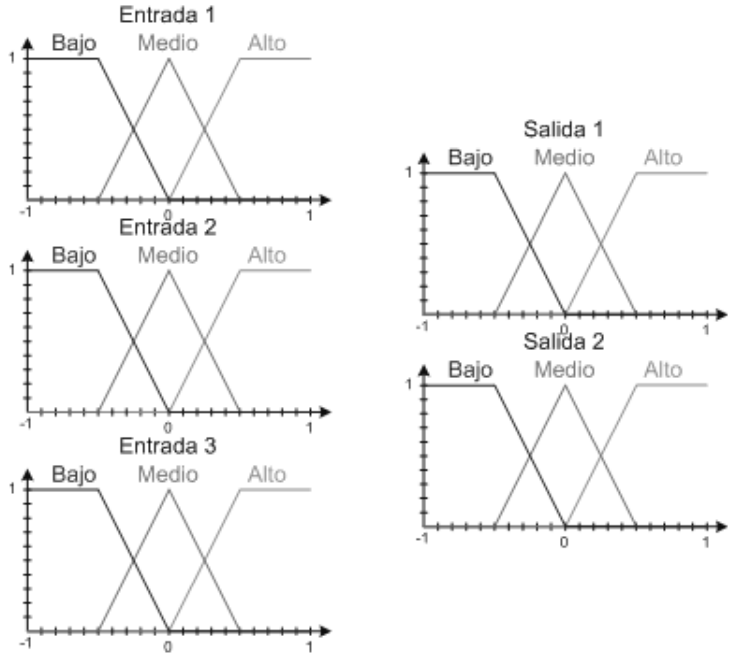


Figura 5.13. Ejemplo de un Sistema de Lógica Difusa tipo Mamdani

ENTRADAS:

- Entrada 1: *Universo*: [-1,1]. Difusor: Pi(0.4,0,0.4)
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}
- Entrada 2: *Universo*: [-1,1]. Difusor: Pi(0.4,0,0.4)
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}
- Entrada 3: *Universo*: [-1,1]. Difusor: Pi(0.4,0,0.4)
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}

SALIDAS:

- Salida 1: *Universo*: [-1,1]. Concresor: Primer Máximo
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}

- Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
- Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}

- Salida 2: *Universo*: [-1,1]. Concresor: Primer Máximo
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}

5.14.1. Reglas

1. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Bajo*
2. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Bajo*
3. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Bajo*
4. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Medio*
5. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Medio*
6. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Medio*
7. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Alto*
8. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Alto*
9. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Alto*
10. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Bajo*
11. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Bajo*
12. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Bajo*
13. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Medio*
14. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Medio*
15. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Medio*
16. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Alto*

17. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Alto*
18. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Alto*
19. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Bajo*
20. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Bajo*
21. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Bajo* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Bajo*
22. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Medio*
23. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Medio*
24. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Medio* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Medio*
25. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Bajo* THEN *Salida 1* es *Bajo* AND *Salida 2* es *Alto*
26. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Medio* THEN *Salida 1* es *Medio* AND *Salida 2* es *Alto*
27. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Alto* AND *Entrada 3* es *Alto* THEN *Salida 1* es *Alto* AND *Salida 2* es *Alto*

5.14.2. Resultados

Para calcular con las entradas (-1,0.4,-0.6) se emplea el código que se presenta a continuación. La Figura 5.14. muestra las reglas que se activaron para este conjunto de entradas (R4, R7 y R8). En la Tabla 5.1. se encuentran algunos resultados para combinaciones diferentes en las entradas.

```
ArrayOfDoubles ent,sal;
ent.Add(Double(-1));
ent.Add(Double(0.4));
ent.Add(Double(-0.6));
sal=SLDM.Item(0).Calcular(ent);
```

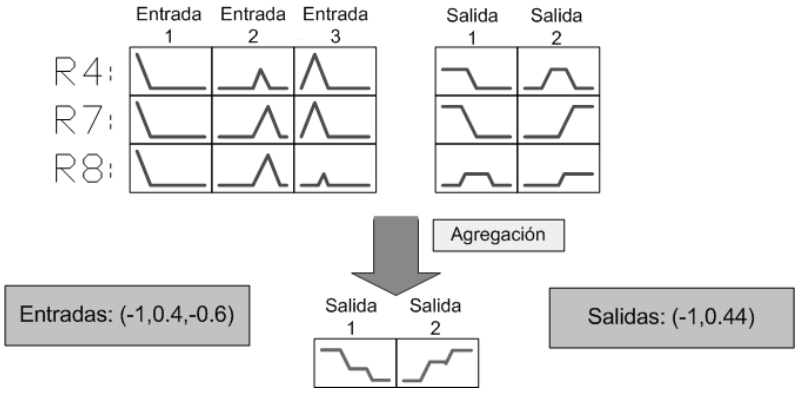


Figura 5.14. Cálculo de un Sistema de Lógica Difusa tipo Mamdani

Entrada 1	Entrada 2	Entrada 3	Salida 1	Salida 2
-1.000000	-1.000000	-1.000000	-1.000000	-1.000000
-1.000000	-1.000000	0.500000	0.500000	-1.000000
-1.000000	-0.800000	1.000000	0.500000	-1.000000
-1.000000	-0.200000	-1.000000	-1.000000	-0.111111
-1.000000	0.400000	-0.600000	-1.000000	0.444444
0.600000	0.500000	0.900000	0.500000	0.500000
1.000000	1.000000	1.000000	0.500000	0.500000

Tabla 5.1. Algunos Resultados de un Sistema de Lógica Difusa

5.15. Ejemplo de Aplicación de un SLD TSK y Algunos Resultados

En el siguiente ejemplo se muestran algunos resultados obtenidos para un Sistema de Lógica Difusa de 2 entradas y 2 salidas regidos por las reglas de la Figura 5.15.

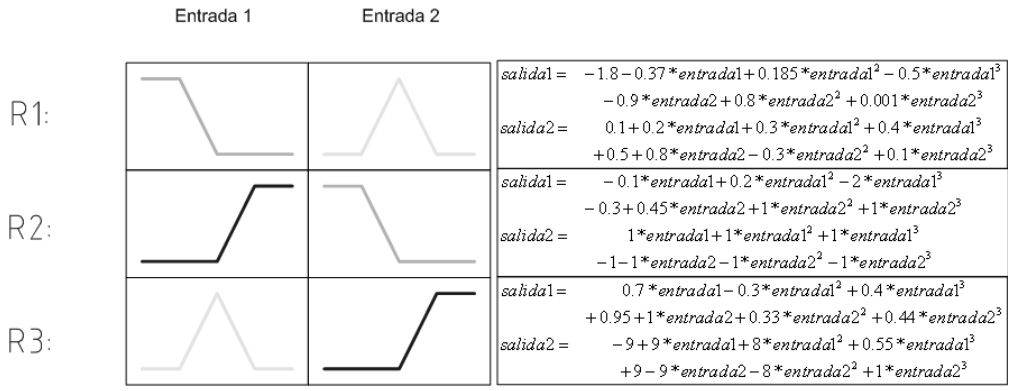


Figura 5.15. Ejemplo de un Sistema de Lógica Difusa tipo TSK

ENTRADAS:

- Entrada 1: *Universo*: [-1,1]. Difusor: Pi(0.4,0,0.4)
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}
- Entrada 2: *Universo*: [-1,1]. Difusor: Pi(0.4,0,0.4)
  - Bajo: Función de Pertenencia: {(-1,1),(-0.5,1),(0,0),(1,0)}
  - Medio: Función de Pertenencia: {(-1,0),(-0.5,0),(0,1),(0.5,0),(1,0)}
  - Alto: Función de Pertenencia: {(-1,0),(0,0),(0.5,1),(1,1)}

SALIDAS:

- Salida 1: *Universo*: [-1,1].
- Salida 2: *Universo*: [-1,1].

5.15.1. Reglas

1. IF *Entrada 1* es *Bajo* AND *Entrada 2* es *Medio* THEN *Salida 1* es -1.8-0.37(Entrada 1) + 0.185(Entrada 1)^2 - 0.5(Entrada 1)^3 - 0.9(Entrada 2) + 0.8(Entrada 2)^2 + 0.001(Entrada 2)^3 AND *Salida 2* es 0.1 + 0.2(Entrada 1) + 0.3(Entrada 1)^2 + 0.4(Entrada 1)^3 + 0.5 + 0.8(Entrada 2) - 0.3(Entrada 2)^2 + 0.1(Entrada 2)^3
2. IF *Entrada 1* es *Alto* AND *Entrada 2* es *Bajo* THEN *Salida 1* es -0.1(Entrada 1) - 2(Entrada 1)^2 - 0.3 + 0.45(Entrada 2) + 1(Entrada 2)^2 AND *Salida 2* es 1(Entrada 1) + 1(Entrada 1)^2 + 1(Entrada 1)^3 - 1 - 1(Entrada 2) - 1(Entrada 2)^2 - 1(Entrada 2)^3
3. IF *Entrada 1* es *Medio* AND *Entrada 2* es *Alto* THEN *Salida 1* es 0.7(Entrada 1) - 0.3(Entrada 1)^2 + 0.4(Entrada 1)^3 + 0.95 + 1(Entrada 2) + 0.33(Entrada 2)^2 + 0.44(Entrada 2)^3 AND *Salida 2* es -9 + 9(Entrada 1) + 8(Entrada 1)^2 + 0.55(Entrada 1)^3 + 9 - 9(Entrada 2) - 8(Entrada 2)^2 + 1(Entrada 2)^3

5.15.2. Resultados

Para calcular con las entradas (-1,0.4) se emplea el código que se presenta a continuación. La Figura 5.16. muestra la regla que se activó para este conjunto de entradas (R1). En la Tabla 5.2. se encuentran algunos resultados para combinaciones diferentes en las entradas.

```
ArrayOfDoubles ent,sal;
ent.Add(Double(-1));
ent.Add(Double(0.4));
sal=Tsk.Calcular(ent);
```

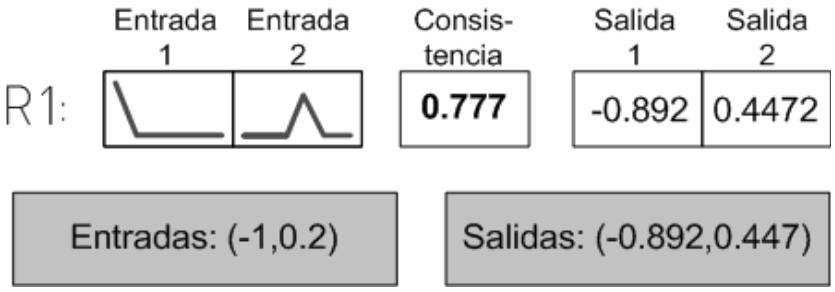


Figura 5.16. Cálculo de un Sistema de Lógica Difusa tipo TSK

Entrada 1	Entrada 2	Salida 1	Salida 2
0.60	0.3	0.250000	0.250000
0.44	0.25	0.245902	0.245902
-1.00	0.20	-0.892992	0.448800
-0.10	0.10	-0.0606384	-0.361654
0.55	0.22	0.3396230	0.339623
-1.00	-1.00	-nan	-nan
0.00	0.52	-0.296875	0.281717

Tabla 5.2. Algunos Resultados de un Sistema de Lógica Difusa TSK

Debido a que no existe una regla definida para *Entrada 1* es *Bajo* y *Entrada 2* es *Bajo* se obtuvo como resultado *-nan*.

5.16. Ejemplo de utilización para la librería

En el CD Anexo se encuentra el código fuente para un programa de consola que agrupa una colección de ejemplos y que fue desarrollado para mostrar el funcionamiento de la librería. El nombre del programa es `FuzzySet.cpp`.

5.16.1. Librerías Incluidas

Este es el listado de las librerías incluídas en el programa de ejemplo.

```
#include <iostream.h>
#include <stdio.h>
#include <wx/wx.h>
#include "fs_cont_r.h"
#include "DoubleArray.h"
#include "alfa-cortes.h"
#include "Operaciones.h"
#include "OperacionesUniarias.h"
#include "OperacionesValores.h"
#include "OperacionesOtras.h"
#include "Ranking.h"
#include "Concresores.h"
#include "Difusores.h"
#include "VariableLinguistica.h"
#include "BaseReglas.h"
#include "SLD.h"
#include "Clustering.h"
```

5.16.2. Funciones

La Tabla 5.3. muestra las funciones del programa y la Tabla 5.4. muestra las variables globales del programa.

int main (int argc, char *argv[])	Función Principal de ejecución. Crea los Conjuntos, Variables, Bases de Reglas, Sistemas de Lógica Difusa de Ejemplo y muestra el siguiente menú. 1. CDCR (Conjunto Difuso Continuo en R): Llama a la función menú_CDCR() 2. CDCR (Conjunto Difuso Continuo en R): Llama a la función menu_CDCR() 3. Operaciones Difusas: Llama a la función *menu_operaciones* 4. Inferencia: Llama a la función *menu_inferencia* 5. *Variables Lingüísticas*: Llama a la función *menu_variable_linguistica* 6. Base de Reglas: Llama a la función *menu_base_reglas* 7. Sistema de Lógica Difusa: Llama a la función *menu_sld* 8. Agrupamiento Difuso: Llama a la función *menu_clustering* 9. Acerca De...: Informa los datos de creación del programa 10. Salir: Sale del programa
void menu_CDCR()	Menú de selección de operaciones sobre un Conjunto Difuso. 1. Crear CDCR: Llama a la función *menu_crear_CDCR* 2. Leer CDCR: Llama a la función *menu_leer_CDCR*

	3. Modificar CDCR: Llama a la función *menu_modificar_CDCR* 4. Borrar CDCR: Borra un Conjunto Difuso definido por el Usuario. 5. Comprobar si es Numero Difuso: Llama a la función *menu_comprobar_nd* 6. Retornar al menú Principal
void menu_crear_CDCR ()	Menú para crear un Conjunto Difuso 1. Crear CDCR por Interpolación Lineal: Ingresa el *Universo* y Llama a la función *OnCrearCDCR* 2. Crear CDCR por Alfa-cortes distribuidos Uniformemente: Ingresa el *Universo* y Llama a la función *OnACUniformeCDCR* 3. Crear CDCR por Alfa-cortes introduciendo los Niveles: Ingresa el *Universo* y Llama a la función OnACNivelesCDCR 4. Retornar al menú Anterior
void menu_leer_CDCR ()	Menú para leer un Conjunto Difuso 1. Leer *Función de Pertenencia* de un CDCR: Selecciona el Conjunto Difuso y Llama a la función *OnLeerCDCR* 2. Leer Alfa-Cortes de un CDCR: Selecciona el Conjunto Difuso y Llama a la función *OnACLeerCDCR* 3. Retornar al menú Anterior
void menu_modificar_CDCR ()	Menú para modificar un Conjunto Difuso 1. Modificar *Universo*: Llama a la función *OnModificarUniversoCDCR* 2. Modificar Nombre: Selecciona el Conjunto Difuso y Llama a la función *OnModificarNombreCDCR* 3. Modificar *Función de Pertenencia*: Llama a la función *menu_modificar_CDCR_FP* *4.* Modificar Alfa-cortes: Llama a la función *menu_modificar_CDCR_AC* 5. Definir Alfa-cortes Uniformemente para una *Función de Pertenencia*: Selecciona el Conjunto Difuso y Llama a la función *OnModificarUniformeFP* 6. Retornar al menú Anterior
void menu_comprobar_nd()	Menú para comprobar si un Conjunto Difuso es Número Difuso 4. Comprobar si el CDCR es Normal: Selecciona el Conjunto Difuso y comprueba si es normal. 5. Comprobar si el CDCR es Convexo: Selecciona el Conjunto Difuso y comprueba si es convexo. 6. Comprobar si el CDCR es Semi-continuo Superior: Indica que se asume semi-continuidad para todos los conjuntos. 7. Comprobar si el CDCR es Numero Difuso: Selecciona el Conjunto Difuso y comprueba si es un número difuso. 8. Retornar al menú Anterior
void menu_operaciones ()	Menú para seleccionar operaciones a efectuar sobre Conjuntos Difusos. Se seleccionan los conjuntos, se realiza la operación indicada y si el resultado es un Conjunto Difuso lo agrega a ConjuntosDifusos. -BINARIAS- Mínimo Máximo

	Suma Resta Multiplicación Mínimo Intervalar Máximo Intervalar -UNIARIAS- 9. Calcular *Valor Representativo* de un Numero Difuso 10. Complemento 11. Muy 12. Más o menos 13. Más 14. No 15. Algo 16. Extremadamente 17. No muy 18. Altura 19. Cardinalidad 20. Normalizar -RETORNO DE FLOTANTES- 21. Subconjuntez 22. P-Distancia 23. Distancia por *Valor Representativo* 24. Consistencia -BOOLEANAS- 25. Subconjunto 26. Soporte de un Conjunto -ORDENAMIENTO- 27. Ordenar Arreglo de CD por *Valor Representativo* 28. Retornar al menú Anterior
void menu_inferencia()	Menú para seleccionar si se operará un difusor o un concresor 1. Difusores: Llama a la función menu_difusores 2. Concresores: Llama a la función menu_concresores 3. Retornar al menu Anterior
void menu_difusores()	Menú para seleccionar un difusor a operar sobre un Número Crisp. 1. Triangulo: Crea un triángulo a partir de un número crisp solicitando los parámetros de distancias (a,b) y (c,d) para un trapecio (a,b,c,d). La distancia (b,c) es igual a cero. 2. Trapecio: Crea un trapecio a partir de un número crisp solicitando los parámetros de distancias (a,b),(b,c) y (c,d) para un trapecio (a,b,c,d). 3. Singleton: Crea un singleton a partir de un número crisp solicitando los parámetros de distancias (b,c) (se sugiere 1e-8) para un trapecio (a,b,c,d). Las distancias (a,b) y (c,d) son iguales a cero. 4. Campana Cuadrática: Crea una campana cuadrática a partir de un número crisp solicitando los parámetros de las curvas a,b,c,e,f) (c=d) para una pi-campana con 4 parábolas (a,b,c,d,e,f,f1,f2). 5. Pi-Campana Cuadrática: Crea una campana cuadrática a partir de un número crisp solicitando los parámetros de las curvas pi-campana con 4 parábolas (a,b,c,d,e,f,f1,f2). 6. Campana Euler: Crea una campana gaussiana a partir de un número crisp solicitando los parámetros de las campanas

	(sigma1,a,sigma2)(a=0). 7. Pi-Campana Euler: Crea una campana gaussiana a partir de un número crisp solicitando los parámetros de las campanas (sigma1,a,sigma2). 8. Salir: Retorna al menú anterior.
void menu_concresores ()	Menú para seleccionar un concresor a operar sobre un Conjunto Difuso 1. Primer Máximo: Selecciona un Conjunto Difuso y le aplica el concresor Primer Máximo. Muestra el resultado. 2. Ultimo Máximo: Selecciona un Conjunto Difuso y le aplica el concresor Último Máximo. Muestra el resultado. 3. Media de Máximos: Selecciona un Conjunto Difuso y le aplica el concresor Media de Máximos. Muestra el resultado. 4. Centro de Gravedad: Selecciona un Conjunto Difuso y le aplica el concresor Centro de Gravedad. Muestra el resultado. 5. Retornar al menú Anterior
void menu_modificar_CDCR_FP ()	Menú para seleccionar cómo modificar un Conjunto Difuso a través de su *Función de Pertenencia* 1. Modificar *Universo*: Llama a la función *OnModificarUniversoCDCR* 2. Modificar Nombre: Selecciona el Conjunto Difuso y Llama a la función *OnModificarNombreCDCR* 3. Modificar *Función de Pertenencia*: Llama a la función *menu_modificar_CDCR_FP* 4. Modificar Alfa-cortes: Llama a la función *menu_modificar_CDCR_AC* 5. Definir Alfa-cortes Uniformemente para una *Función de Pertenencia*: Selecciona el Conjunto Difuso y Llama a la función *OnModificarUniformeFP* 6. Retornar al menú Anterior
void menu_modificar_CDCR_AC ()	Menú para seleccionar cómo modificar un Conjunto Difuso a través de sus Alfa-cortes 1. Modificar Definición Uniforme de Alfa-cortes: Selecciona el Conjunto Difuso y Llama a la función *OnModificarUniforme* 2. Modificar Definición de Niveles de Alfa-cortes: Selecciona el Conjunto Difuso y Llama a la función *OnModificarNiveles* 3. Agregar un Alfa-corte de un Nivel: Selecciona el Conjunto Difuso y Llama a la función *OnModificarAgregarAC* 4. Retornar al menu Anterior
Void menu_variable_linguistica ()	Menú de selección de operaciones sobre un Conjunto Difuso 1. Crear *Variable Lingüística*: Llama a la función *menu_crear_variable_linguistica* 2. Leer *Variable Lingüística*: Llama a la función *menu_leer_variable_linguistica* 3. Modificar *Variable Lingüística*: Llama a la función *menu_modificar_variable_linguistica* 4. Borrar *Variable Lingüística*: Borra un Conjunto Difuso definido por el Usuario. 5. Retornar al Menú Principal

void menu_crear_variable_linguistica ()	Menú para crear una *Variable Lingüística* 1. Crear VL predefinida (Alto,Medio,Bajo): Crea una *Variable Lingüística* con tres términos de variable solicitando al usuario las cotas del universo: Medio es una función triángulo, alto es una función Gamma y Bajo es una función L. 2. Crear VL por *Función de Pertenencia*: Solicita al usuario la cantidad de términos y las cotas del universo, y crea los términos mediante la función *OnCrearCDCR* 3. Crear VL por Alfa-cortes Uniformemente distribuidos: Solicita al usuario la cantidad de términos y las cotas del universo, y crea los términos mediante la función *OnACUniformeCDCR* 4. Crear VL por Alfa-cortes introduciendo Niveles: Solicita al usuario la cantidad de términos y las cotas del universo, y crea los términos mediante la función *OnACNivelesCDCR* 5. Retornar al Menú Principal
void menu_leer_variable_linguistica ()	Menú para leer una *Variable Lingüística* 1. Leer VL por *Función de Pertenencia*: Selecciona la *Variable Lingüística* y muestra cada término mediante la unción *OnLeerCDCR* 2. Leer VL por Alfa-cortes: Selecciona la *Variable Lingüística* y muestra cada término mediante la función *OnACLeerCDCR* 3. Retornar al Menu Principal
void menu_modificar_variable_linguistica ()	Menú para modificar una *Variable Lingüística* 1. Modificar Nombre de la Variable: Selecciona la *Variable Lingüística* y le cambia el nombre 2. Modificar Nombre de un Termino: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre ---*FUNCIÓN DE PERTENENCIA*--- 3. Modificar Numero de Puntos de *Función de Pertenencia*: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarNoPtFP* 4. Modificar Puntos de *Función de Pertenencia*: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarPtFP* 5. Agregar Punto a *Función de Pertenencia*: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarAgregarPtFP* ---ALFA-CORTES--- 6. Modificar Definición Uniforme de Alfa cortes: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarUniforme* 7. Modificar Definición de Niveles de Alfa cortes: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarNiveles* 8. Agregar Alfa corte de un Nivel: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarAgregarAC* 9. Modificar Definición de Niveles de Alfa cortes: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarNiveles*

	10. Definir Alfa cortes Uniformemente para una *Función de Pertenencia*: Selecciona la *Variable Lingüística* y su término de variable y le cambia el nombre mediante la función *OnModificarUniformeFP* 11. Borrar Conjunto Difuso de una VL: Selecciona la *Variable Lingüística* y su término de variable y se elimina de la variable. 12. Retornar al Menú Principal
void menu_base_reglas()	Menú para operar Bases de Reglas 1. Crear Base de Reglas: Solicita al usuario la cantidad de reglas tipo Mamdani y para cada regla la cantidad de términos del antecedente y del consecuente. 2. Agrega la Base de Reglas a *BasesReglas*. 3. Agregar Regla: Solicita al usuario el número de Base de Reglas a agregar la regla y la cantidad de términos del antecedente y del consecuente. 4. Ver Base de Reglas: Solicita al usuario el número de Base de Reglas e imprime en pantalla las reglas de esta Base. 5. Borrar Regla: Solicita al usuario el número de Base de Reglas y la regla de esta base y la borra de la Base de Reglas 6. Borrar Base de Reglas: Solicita al usuario el número de Base de Reglas y lo borra de *BasesReglas*. ---*** BASE DE REGLAS TIPO TSK ***--- 7. Crear Base de Reglas: Solicita al usuario la cantidad de reglas tipo TSK y para cada regla la cantidad de términos del antecedente y las funciones del consecuente. Agrega la Base de Reglas a *BasesReglasTSK*. 8. Agregar Regla: Solicita al usuario el número de Base de Reglas a agregar la regla y la cantidad de términos del antecedente y las funciones del consecuente. 9. Ver Base de Reglas: Solicita al usuario el número de Base de Reglas TSK e imprime en pantalla las reglas de esta Base. 10. Borrar Regla: Solicita al usuario el número de Base de Reglas y la regla de esta base y la borra de la Base de Reglas 11. Borrar Base de Reglas: Solicita al usuario el número de Base de Reglas y lo borra de *BasesReglasTSK*. 12. Retornar al Menu Principal
void menu_SLD()	Menú para seleccionar Sistemas de Lógica Difusa 1. SLD tipo Mamdani: Si no existe un Sistema de Lógica Difusa, crea uno indicando el número de entradas y salidas, si no se selecciona uno del arreglo SLDM. Luego de esto se llama la función menu_SLD_Mamdani para el Sistema seleccionado. 2. SLD TSK: Si no existe un Sistema de Lógica Difusa, crea uno indicando el número de entradas y salidas, si no se selecciona uno del arreglo SLDTSK. Luego de esto se llama la función menu_SLD_TSK para el Sistema seleccionado. 3. Retornar al Menu Principal
void menu_SLD_Mamdani (SLD_Mamdani &Mam)	Menú de selección de un Sistema de Lógica Difusa tipo Mamdani 1. Agregar Base de Reglas: Indica que la Base de Reglas del

	SLD se toma del arreglo *BasesReglas*. 2. Agregar Regla: Agrega una regla a la Base de Reglas del SLD. 3. Borrar Regla: Borra un regla de la Base de Reglas del SLD. 4. Definir Difusores: Se definen los difusores para todas las entradas del SLD. 5. Definir Concresores: Se definen los concresores para todas las salidas del SLD. 6. Verificar Integridad: Función que no se encuentra completamente implementada para verificar errores del SLD, pero verifica que el número de entradas sea coincidente. 7. Calcular: Calcula las salidas para determinados valores de las entradas. 8. Ver Base de Reglas: Muestra las Reglas del SLD. 9. Retornar al Menú Principal
void menu_SLD_TSK (SLD_TSK &Tsk)	Menú de selección de un Sistema de Lógica Difusa tipo TSK 1. Agregar Base de Reglas: Indica que la Base de Reglas del SLD se toma del arreglo *BasesReglasTSK*. 2. Agregar Regla: Agrega una regla TSK a la Base de Reglas del SLD. 3. Borrar Regla: Borra un regla de la Base de Reglas del SLD. 4. Definir Difusores: Se definen los difusores para todas las entradas del SLD. 5. Verificar Integridad: Función que no se encuentra completamente implementada para verificar errores del SLD, pero verifica que el número de entradas sea coincidente. 6. Calcular: Calcula las salidas para determinados valores de las entradas. 7. Ver Base de Reglas: Muestra las Reglas TSK del SLD. 8. Retornar al Menú Principal.
void menu_Clustering()	Menú de selección de Agrupamiento por el método C-Means 1. Crear X y Calcular V: Crea la Matriz X con c casos y p Variables x_i. Calcula la matriz de Centros V 2. Emplear Archivos para Calcular V: Realiza los mismos cálculos, pero emplea archivos para almacenar las matrices. También da la posibilidad de empleo de archivos para almacenar las matrices de cálculos intermedios. ---VARIABLES LINGÜÍSTICAS--- 3. Crear X, Calcular V, crear *Variables Lingüísticas*: Crea la Matriz X con n casos y p Variables x_i. Calcula la matriz de Centros V con c centros y con V crea c términos para p *Variables Lingüísticas*. 4. Emplear Archivos para Calcular V y Crear *Variables Lingüísticas*: Realiza los mismos cálculos que el ítem anterior, pero emplea archivos para almacenar las matrices y variables. También da la posibilidad de empleo de archivos para almacenar las matrices de cálculos intermedios. 5. Retornar al Menú Principal

void OnCrearCDCR (ArregloDeConjuntosDifusos &ACD, DoubleArray &univ)	Función que Crea, nombra, define el *Universo* mediante *univ*, ingresa los puntos de la *Función de Pertenencia* y agrega el Conjunto Difuso al Arreglo de Conjuntos Difusos *ACD*.
void OnLeerCDCR (ArregloDeConjuntosDifusos &arreglo, int conjunto)	Función que imprime en pantalla el nombre, universo y *Función de Pertenencia* el Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnACUniformeCDCR (ArregloDeConjuntosDifusos &arreglo, DoubleArray &univ)	Función que Crea, nombra, define el *Universo* mediante *univ*, ingresa los intervalos de los alfa-cortes distribuidos a distancias uniformes y agrega el Conjunto Difuso al Arreglo de Conjuntos Difusos *arreglo*.
void OnACNivelesCDCR (ArregloDeConjuntosDifusos &arreglo, DoubleArray &univ)	Función que Crea, nombra, define el *Universo* mediante *univ*, ingresa los intervalos de los alfa-cortes distribuidos a distancias solicitadas al usuario y agrega el Conjunto Difuso al Arreglo de Conjuntos Difusos *arreglo*.
void OnACLeerCDCR (ArregloDeConjuntosDifusos &arreglo, int conjunto)	Función que imprime en pantalla el nombre, universo y Alfa-cortes del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarUniversoCDCR ()	Función que retorna un mensaje de error que indica que es necesario crear el conjunto de nuevo. Resulta más práctico Borrarlo y crearlo nuevamente.
void OnModificarNombreCDCR (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que cambia el nombre del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarNoPtFP (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que modifica el número de puntos y los puntos de la *Función de Pertenencia* del Conjunto Difuso de la posición *posicion*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarPtFP (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que modifica los puntos de la *Función de Pertenencia* del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarAgregarPtFP (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que agrega un punto a la *Función de Pertenencia* del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarUniforme (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que redefine la distribución y los intervalos de los Alfa-cortes del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarAgregarAC	Función que agrega un Alfa-corte al Conjunto

(ArregloDeConjuntosDifusos &arreglo, int posicion)	Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarNiveles (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que modifica los intervalos de los Alfa-cortes del Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnModificarUniformeFP (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que modifica los alfa-cortes definiéndolos uniformemente para la *Función de Pertenencia* establecida de un Conjunto Difuso de la posición *conjunto*, en el Arreglo de Conjuntos Difusos *arreglo*.
void OnBorrarCDCR (ArregloDeConjuntosDifusos &arreglo, int posicion)	Función que elimina el Conjunto difuso de la posición *posicion* en el Arreglo de Conjuntos Difusos *arreglo*.
DoubleArray IngresarUniverso ()	Función Intermedia que solicita al usuario las cotas del universo, las almacena en un DoubleArray que es retornado por la función.
DoubleArray IngresarPtFP (int nopt, double x, double y)	Función Intermedia que solicita al usuario los *nopt* puntos de la *Función de Pertenencia* asegurándose de que se encuentren dentro de las cotas del universo [*x*,*y*], los almacena en un DoubleArray que es retornado por la función.
void NuevoCDCR (fs_cont_r &nuevoCDCR, ArregloDeConjuntosDifusos &arreglo)	Función Intermedia que agrega un nuevo Conjunto Difuso al Arreglo de Conjuntos Difusos *arreglo*.
int SeleccionarCDCR (ArregloDeConjuntosDifusos &arreglo)	Función Intermedia que imprime en pantalla los nombres de los Conjuntos Difusos en el Arreglo de Conjuntos Difusos *arreglo*, espera una entrada el usuario indicando el conjunto seleccionado y devuelve la posición de este en *arreglo*.
int SeleccionarVL (ArregloDeVariablesL &arreglo)	Función Intermedia que imprime en pantalla los nombres de las *Variables Lingüísticas* en el Arreglo de *Variables Lingüísticas arreglo*, espera una entrada el usuario indicando la variable seleccionada y devuelve la posición de este en *arreglo*.
int SeleccionarBR (ArregloDeBaseDeReglas &arreglo)	Función Intermedia que imprime en pantalla los números de las Bases de Reglas tipo Mamdani existentes en el Arreglo de Bases de reglas *arreglo*, espera una entrada el usuario indicando la Base seleccionada y devuelve la posición de esta en *arreglo*.
int SeleccionarBRTSK (ArregloDeBaseDeReglasTSK &arreglo)	Función Intermedia que imprime en pantalla los números de las Bases de Reglas tipo TSK existentes en el Arreglo de Bases de reglas *arreglo*, espera una entrada el usuario indicando la Base seleccionada y devuelve la posición de esta en *arreglo*.

int SeleccionarSLDM (ArraySLDM &arreglo)	Función Intermedia que imprime en pantalla los números de los Sistemas de Lógica Difusa tipo Mamdani existentes en el Arreglo de Sistemas de Lógica Difusa *arreglo*, espera una entrada el usuario indicando el Sistema seleccionada y devuelve la posición de esta en *arreglo*.
int SeleccionarSLDTSK (ArraySLDTSK &arreglo)	Función Intermedia que imprime en pantalla los números de los Sistemas de Lógica Difusa tipo Mamdani existentes en el Arreglo de Sistemas de Lógica Difusa *arreglo*, espera una entrada el usuario indicando el Sistema seleccionada y devuelve la posición de esta en *arreglo*.
void LeerBaseReglas (BaseDeReglas &arreglo,ArregloDeVariablesL &varlin)	Función Intermedia que imprime en pantalla la base de reglas *arreglo* con las *Variables Lingüísticas varlin*.
void LeerBaseReglasTSK (BaseDeReglasTSK &arreglo,ArregloDeVariablesL &varlin)	Función Intermedia que imprime en pantalla la base de reglas TSK *arreglo* con las *Variables Lingüísticas varlin*.
void AgregarRegla (BaseDeReglas &arreglo,ArregloDeVariablesL &varlin)	Función Intermedia que solicita al usuario la entrada de una regla para la Base de Reglas *arreglo* con las *Variables Lingüísticas varlin*.
void AgregarReglaTSK (BaseDeReglasTSK &arreglo,ArregloDeVariablesL &varlin)	Función Intermedia que solicita al usuario la entrada de una regla para la Base de Reglas tipo TSK *arreglo* con las *Variables Lingüísticas varlin*.
void COUTwxString (wxString Total)	Función Intermedia que imprime en pantalla el wxString *Total*.
wxString LeerwxString ()	Función Intermedia que lee por teclado y retorna un wxString.
void CrearConjuntosEjemplo ()	Función que crea unos Conjuntos Difusos de Ejemplo y los agrega al Arreglo de Conjuntos Difusos *ConjuntosDifusos*.
void CrearVariablesEjemplo ()	Función que crea unas *Variables Lingüísticas* de Ejemplo y las agrega al Arreglo de *Variables Lingüísticas VariablesLinguisticas*.
void CrearBaseReglasEjemplo ()	Función que crea unas Bases de Reglas tipo Mamdani de Ejemplo y las agrega al Arreglo de Bases de Reglas *BasesReglas*.
void CrearBaseReglasTSKEjemplo ()	Función que crea unas Bases de Reglas tipo TSK de Ejemplo y las agrega al Arreglo de Bases de Reglas *BasesReglasTSK*.
void CrearSLDMEjemplo ()	Función que crea unos Sistemas de Lógica Difusa tipo Mamdani de Ejemplo y los agrega al Arreglo de

	Sistemas de Lógica Difusa *SLDM*.
void CrearSLDTSKEjemplo()	Función que crea unos Sistemas de Lógica Difusa tipo TSK de Ejemplo y los agrega al Arreglo de Sistemas de Lógica Difusa *SLDTSK*.

Tabla 5.3. Funciones del programa FuzzySet.cpp

5.16.3. Variables

ArregloDeConjuntosDifusos ConjuntosDifusos	Arreglo global de Conjuntos Difusos.
ArregloDeVariablesL VariablesLinguisticas	Arreglo global de *Variables Lingüísticas*.
ArregloDeBaseDeReglas BasesReglas	Arreglo global de Bases de Reglas tipo Mamdani.
ArregloDeBaseDeReglasTSK BasesReglasTSK	Arreglo global de Bases de Reglas tipo TSK.
ArraySLDM SLDM	Arreglo global de Sistemas de Lógica Difusa tipo Mamdani.
ArraySLDTSK SLDTSK	Arreglo global de Sistemas de Lógica Difusa tipo TSK.
ArregloDoubleArray CasosCluster	Arreglo global de los Casos creados por Clustering.

Tabla 5.4.Variables Globales del Programa FuzzySet.cpp

6. CONCLUSIONES

Durante el desarrollo de la presente tesis se elaboró una librería en C++ que puede incluirse en diversas plataformas. La *Librería para Técnicas Difusas* generó una herramienta de *Libre Distribución* unificada que agrupa diversas caracterizaciones antes sólo presente en muy pocas herramientas comerciales de alto costo. También reunió algunas Técnicas Difusas que se habían desarrollado de manera independiente en la Universidad. La librería tiene las siguientes características:

- Creación de conjuntos Difusos descritos de forma dual: α-cortes y Función de Pertenencia
- Creación de Variables Lingüísticas
- Operaciones Difusas:
  - Operaciones Uniarias: Complemento y modificadores.
  - Mínimo y Máximo
  - Aritmética Difusa: Suma, Resta, Multiplicación.
  - Operaciones Intervalares: Mínimo Intervalar, Máximo Intervalar.
  - Subconjunto y Soporte.
- Modelamiento, diseño e implementación de Sistemas Basados en Reglas
  - Sistemas tipo Mamdani
  - Sistemas tipo Takagi-Sugeno-Kang (TSK)
- Agrupamiento Difuso según el método Fuzzy C-Means
  - Creación (modelamiento) de Variables Lingüísticas a partir de los datos agrupados por el método Fuzzy C-Means
- Por estar elaborada en C++ está Orientada a Objetos lo que facilita una posterior expansibilidad y modularidad a través de la herencia, además facilita la creación de una interfaz gráfica amigable por estar basado en wxWindows.

Se sugiere la ampliación de la librería en el futuro con otras Técnicas Difusas como Modelamiento por Universos Fijos y por Universos Variables, implementación de Técnicas Neurodifusas, además de otras operaciones y técnicas que fueron tenidas en cuenta en esta Tesis, pero que por su desarrollo modular son de facil implementación. Se sugiere además realizar la depuración de la librería, por ejemplo en lo que se refiere a la validación de datos de entrada a las funciones. También es posible emplear la librería para la creación de una herramienta gráfica que permita el análisis y graficación de los Sistemas de Lógica Difusa y que pueda implementar una Calculadora Difusa. Dicha herramienta facilitaría notablemente la visualización y comprensión de los Conjuntos Difusos. Otra posibilidad es la creación de una Herramienta CAD, una extensión de la herramienta gráfica, que facilite la creación y manejo de Conjuntos, Operaciones y Modelamiento.

En la Figura 2.1. se esbozan algunas de los trabajos sugeridos para futuras expansiones. SAD son los sistemas de aritmética Difusa. Además se espera que se empleen estas técnicas difusas sobre $\Re^n$.

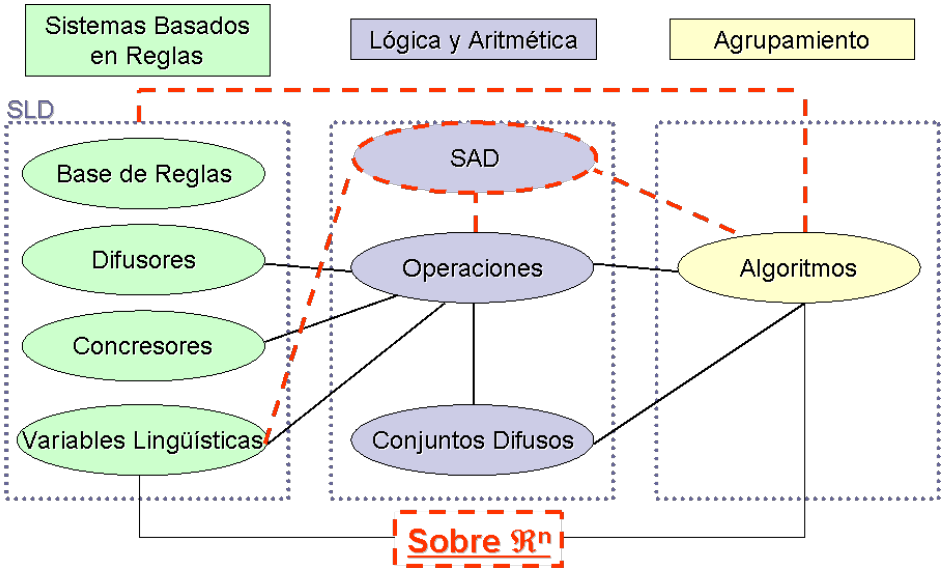


Figura 6.1. Trabajos Futuros

BIBLIOGRAFÍA

[1] ALVARADO, M. y otros Metodología para determinar la factibilidad de implementación de la tecnología de banda ancha LMDS(local multipoint distribution system), en una organización. Bogotá. 2004.

[2] ARROYO, J. M. y otros. Sistema experto basado computación flexible para la selección de parámetros de maquinado. Bogotá. 2003.

[3] CÁRDENAS, E. y otros. Editor gráfico para el prototipo de un sistema experto como herramienta en la negociación de energía eléctrica en la bolsa de Santafé de Bogotá. Bogotá. 1999.

[4] DELGADO Rangel, L.J. y otros. Desarrollo de una librería genérica en C++ para el procesamiento de imágenes con lógica difusa. Bogotá. 2002.

[5] DUARTE, O. Aplicaciones de la Lógica Difusa. En: Ingeniería e Investigación. Número 45. pp. 5-12. 2000.

[6] DUARTE, O. Agrupamiento Difuso. Sin publicar. 2000.

[7] DUARTE, O. UNFUZZY - Software para el diseño, análisis, simulación e implementación de sistemas de lógica difusa. Bogotá. 1997.

[8] ESPEJO, J. H. y otros. Implantación de un prototipo para control borroso. "Análisis, diseño y construcción de una biblioteca en C++ para crear y manejar conjuntos difusos. Bogotá. 1996.

[9] FERNÁNDEZ, O. P. y otros. Diseño de un sistema para el control del proceso de pasteurización de la planta piloto de leches del ICTA-UN. Bogotá. 2002.

[10] GALLEGO, L. E. y otros. Metodología para al análisis de riesgo de daño por rayos en Colombia utilizando técnicas difusas. Bogotá. 2002.

[11] LEIVA, O. E. y otros. Módulo de implementación de controladores difusos de entradas/salidas de un P.L.C. Bogotá. 1998.

[12] LÓPEZ, E. M. Nuevas alternativas de secado mecánico de granos. Bogotá. 2002.

[13] MARIÑO, Daniel y otros. Control de un secador rotatorio con sistema de lógica difusa. Bogotá. 1999.

[14] MARIÑO, J. E. y otros. Diseño y control de un control difuso basado en micro-controladores.

[15] MARTÍNEZ, J. A. y otros. Sistema experto para negociación de energía en sistemas hidrotérmicos. Bogotá. 2002.

[16] MOLANO , L. Y. y otros. Modelamiento evolutivo en sistemas de lógica difusa. Bogotá. 2002.

[17] ROBAYO, R. I. y otros. Modelo de PSS y AVR basado en lógica difusa para un generador sincrónico en sistemas de potencia. Bogotá. 1999.

[18] ROJAS, O. y otros. Segmentación de imágenes usando algoritmos fuzzyBogotá. 1999.

[19] RONCANCIO, L. F. y otros. Control difuso de posición para un motor de corriente continua. Bogotá. 1995.

[20] BAEZ, A. D. Métodos y Modelos de Programación Lineal Difusa. Bogotá. 2003.

[21] wxWidgets. En: http://www.wxwindows.org/

ANEXOS

Anexo A. Bibliografía de Referencia para Bases de Lógica Difusa

[1] DUARTE, O. Sistemas de Lógica Difusa – Fundamentos. En: Ingeniería e Investigación. Número 42. pp. 22-30. 1999

[2] PÉREZ, G. Fundamentos de Lógica Difusa. Sin publicar. 2001.

[3] KANTROWITZ, M. FAQ: Fuzzy Logic and Fuzzy Expert Systems. En: http://www-2.cs.cmu.edu/Groups/AI/html/faqs/ai/fuzzy/part1/faq.html. 1997.

[4] LUKE, B. Fuzzy Sets and Fuzzy Logic. En: http://members.aol.com/btluke/fuzzy01.htm

ÍNDICE