Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica
EL-7038- Introducción a Teoría de Conjuntos Difusos y Sistemas Inteligentes

# Tarea de Avance 1

Resumen

Alumno: Juan Pablo Futalef

Profesores: Claudio Held
Auxiliar: Gabriel Orellana López
Ayudantes: Diego Jiménez

## 1. Acerca de

Este resumen contiene los resultados obtenidos de la implementación de una interfaz de difusión mediante un programa computacional. La interfaz de difusión ha sido implementada en Python 3.5 utilizando la librería Numpy para manejo de arreglos de forma eficiente, y la librería MatplotLib para la gráfica de resultados. El código está alojado en completitud y detallado en un repositorio público[1], pero además se incluye en este documento.

En forma general, el código opera como sigue: se define la función `trap_membership_degree` que permite calcular el grado de pertenencia de un sólo valor de acuerdo a una función tipo trapezoidal. Se define la función `trap_fuzzy_fun` para calcular el conjunto de pertenencia en un intervalo discretizado, esto es logrado utilizando `trap_membership_degree` para calcular el grado de pertenencia de cada uno de los valores dentro del intervalo. Las funciones `intersection`, `union` y `compliment` se encargan de retornar la intersección, unión y complemento de los conjuntos ingresados respectivamente.

## 2. Código implementado

El archivo `FuzzyInferenceSystem.py` contiene todo lo necesario para la correcta ejecución del programa. A continuación se muestra una copia exacta del código, comentado en inglés.

```
"""
This program implements a fuzzy interface, including computation of
   membership,fuzzy operations and plotting of membership sets.

It's necessary to count with numpy and matplotlib libraries.
"""

__author__ = 'Juan Pablo Futalef'
__email__ = "jpfutalef@gmail.com"

import numpy as np
import matplotlib.pyplot as plt
```

[1] https://github.com/jpfutalef/FCFM/tree/master/EL7038/FuzzyInferenceSystem

```
def trap_membership_degree(a, set_points):
    """
    Calculates membership degree of input according to trapezoidal
   function characterized by a size 4 array.
    :param a: number to evaluate in fuzzy set
    :param set_points: size 4 array with trapezoid definitions
    :return: membership degree of a
    Note: If points representing start and end of same support are equal,
   the membership degree is assumed to be one. For example [2,2,3,4], as
   both first and second values are 2, the membership degree of 2 is 1.
    """
    if len(set_points) > 4 or len(set_points) < 4:
        """Argument given is not a size 4 array."""
        raise NameError('nonTrapezoidalSet')
    else:
        if a < set_points[0] or a > set_points[3]:
            """Outside trapezoid case"""
            return 0
        elif a < set_points[1]:
            """Left support case"""
            d = set_points[1] - set_points[0]
            if d == 0:
                """Both values are equal"""
                return 1
            else:
                return (a - set_points[0]) / d
        elif a > set_points[2]:
            """Right support case"""
            d = set_points[2] - set_points[3]
            if d == 0:
                """both values are equal"""
                return 1
            else:
                return (a - set_points[3]) / d
        else:
            """Value is in core"""
            return 1

def trap_fuzzy_fun(u, set_points):
    """
    Allows to calculate membership values of input respect to trapezoidal
   function
    :param u: input vector
    :param set_points: size 4 array vector with trapezoid definitions
    :return: array vector with membership degrees vales for every input in
    u
    """
    return np.array([trap_membership_degree(i, set_points) for i in u])


def intersection(a, b):
    """
    Intersects a and b sets using T norm as minimum. a and b should be
   same length arrays.
```

```
    :param a: array containing first set
    :param b: array containing second set
    :return: intersection of both a and b sets.
    """
    return np.array([min(x) for x in list(zip(a, b))])


def union(a, b):
    """
    Intersects a and b sets using T norm as maximum. a and b should be
   same length arrays.
    :param a: array containing first set
    :param b: array containing second set
    :return: union of both a and b sets.
    """
    return np.array([max(x) for x in list(zip(a, b))])


def compliment(a):
    """
    Creates compliment set of a trough 1-a[i] operation of each element.
    :param a: target set
    :return: compliment of set
    """
    return np.array([1 - x for x in a])


# Define trapezoidal functions by size 4 arrays
A = np.array([-1.0, -1.0, -.9, -.2])
B = np.array([-.6, -.5, .0, .1])
C = np.array([-.3, .0, .2, .3])
D = np.array([.1, .2, .3, .8])
E = np.array([.4, .6, 1, 1])

# Define X axis values
x = np.linspace(-1, 1, 41)

# Create array containing sets definitions (size 4 arrays) an iterate
   trough them
sets = [A, B, C, D, E]
S = []
for i in sets:
    """
    Appends every fuzzy membership set in S. It gives S[0] as A set, S[1]
   as B set, and so on.
    Also, add graphs to plot object.
    """
    s = trap_fuzzy_fun(x, i)
    S.append(s)
    plt.plot(x, s)

# Plot membership functions and save result to file
plt.xlabel('Input value')
plt.ylabel('Membership degree')
```

```
plt.title('Membership functions')
plt.legend(['A', 'B', 'C', 'D'])
plt.savefig('membershipFuns.png')
#plt.show()
plt.close()


# Operacion 1
op1 = union(S[3], intersection(S[0], S[2]))
plt.subplot(2, 2, 1)
plt.xlabel('Input value')
plt.ylabel('Membership degree')
plt.plot(x, op1)
plt.title('Operación 1')


# Operacion 2
op2 = intersection(compliment(intersection(S[0], S[1])), intersection(S
    [3], S[4]))
plt.subplot(2, 2, 2)
plt.xlabel('Input value')
plt.ylabel('Membership degree')
plt.plot(x, op2)
plt.title('Operación 2')

# Operacion 3
op3 = intersection(compliment(S[3]), union(S[1], S[0]))
plt.subplot(2, 2, 3)
plt.xlabel('Input value')
plt.ylabel('Membership degree')
plt.plot(x, op3)
plt.title('Operación 3')

# Operacion 4
op4 = compliment(union(S[0], S[1]))
plt.subplot(2, 2, 4)
plt.xlabel('Input value')
plt.ylabel('Membership degree')
plt.plot(x, op4)
plt.title('Operación 4')

# Adjust plot layout for good fit and save to file
plt.tight_layout()
plt.savefig('operationResults.png')
#plt.show()
plt.close()

# Export to csv file sets and operations
setsCSV = np.transpose(np.array([x, S[0], S[1], S[2], S[3], S[4]]))
operationsCSV = np.transpose(np.array([x, op1, op2, op3, op4]))
np.savetxt('valuesSets.csv', setsCSV, delimiter=" ", fmt='%1.3f')
np.savetxt("valuesOperations.csv", operationsCSV, delimiter=" ", fmt='%1.3
    f')
```

# 3. Resultados

Se tienen 5 conjuntos difusos definidos por las siguientes funciones trapezoidales:

- A = [-1,-1,-0.9,-0.2]
- B = [-0.6, -0.5, 0, 0.1]
- C = [-0.3, 0, 0.2, 0.3]
- D = [0.1, 0.2, 0.3, 0.8]
- E = [0.4, 0.6, 1, 1]

La gráfica de los conjuntos se muestra en la figura 1 y los valores numéricos de cada punto graficado en la tabla 1. Claramente existe un traslape de las funciones, habiendo la presencia de núcleo, soporte y valores nulos para cada uno de los trazos, identificando los conjuntos como difusos.

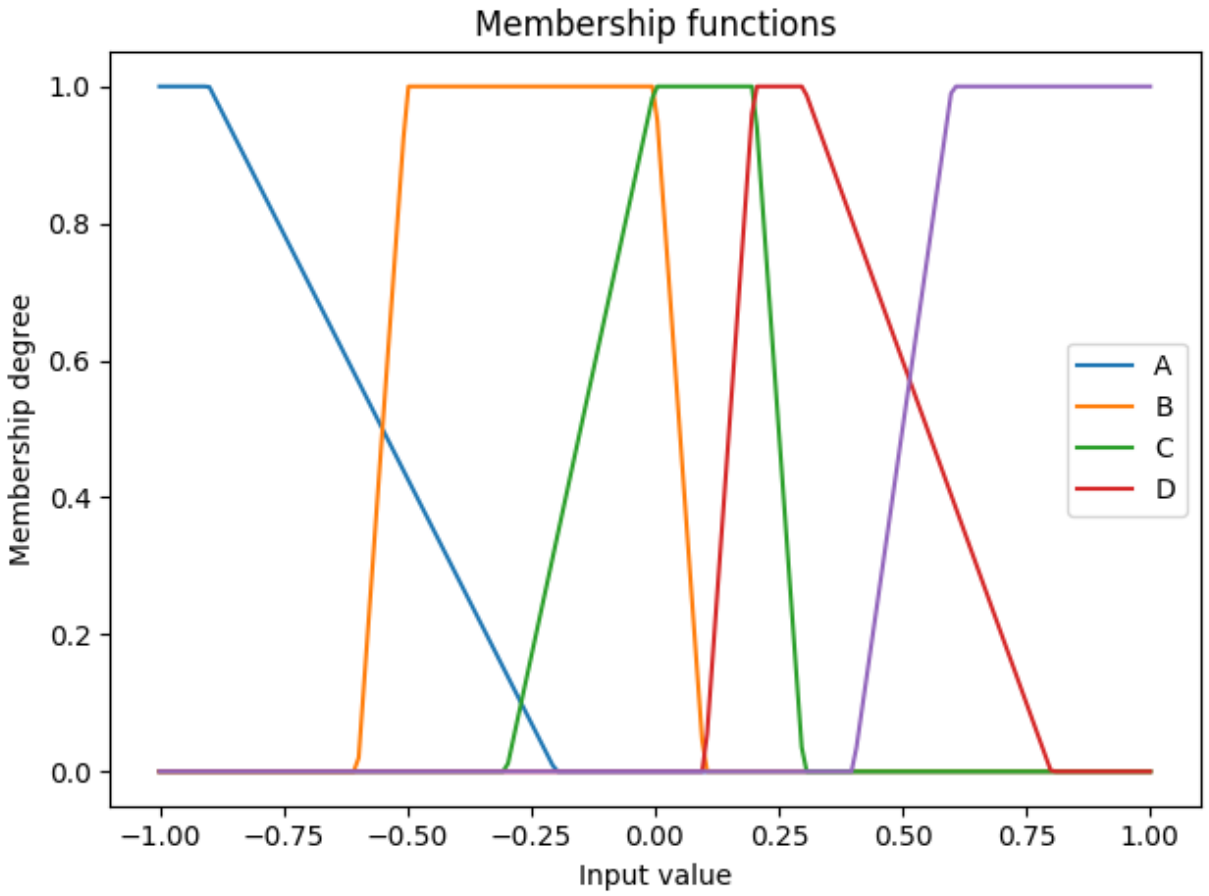


Figura 1: Funciones de pertenencia de los conjuntos A, B, C, D y E

Tabla 1: Valores tomados en conjuntos difusos.

| **X** | **A** | **B** | **C** | **D** | **E** |
|---|---|---|---|---|---|
| -1.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.950 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.900 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.850 | 0.929 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.800 | 0.857 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.750 | 0.786 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.700 | 0.714 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.650 | 0.643 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.600 | 0.571 | 0.000 | 0.000 | 0.000 | 0.000 |
| -0.550 | 0.500 | 0.500 | 0.000 | 0.000 | 0.000 |
| -0.500 | 0.429 | 1.000 | 0.000 | 0.000 | 0.000 |
| -0.450 | 0.357 | 1.000 | 0.000 | 0.000 | 0.000 |
| -0.400 | 0.286 | 1.000 | 0.000 | 0.000 | 0.000 |
| -0.350 | 0.214 | 1.000 | 0.000 | 0.000 | 0.000 |
| -0.300 | 0.143 | 1.000 | 0.000 | 0.000 | 0.000 |
| -0.250 | 0.071 | 1.000 | 0.167 | 0.000 | 0.000 |
| -0.200 | 0.000 | 1.000 | 0.333 | 0.000 | 0.000 |
| -0.150 | 0.000 | 1.000 | 0.500 | 0.000 | 0.000 |
| -0.100 | 0.000 | 1.000 | 0.667 | 0.000 | 0.000 |
| -0.050 | 0.000 | 1.000 | 0.833 | 0.000 | 0.000 |
| 0.000 | 0.000 | 1.000 | 1.000 | 0.000 | 0.000 |
| 0.050 | 0.000 | 0.500 | 1.000 | 0.000 | 0.000 |
| 0.100 | 0.000 | 0.000 | 1.000 | 0.000 | 0.000 |
| 0.150 | 0.000 | 0.000 | 1.000 | 0.500 | 0.000 |
| 0.200 | 0.000 | 0.000 | 1.000 | 1.000 | 0.000 |
| 0.250 | 0.000 | 0.000 | 0.500 | 1.000 | 0.000 |
| 0.300 | 0.000 | 0.000 | 0.000 | 1.000 | 0.000 |
| 0.350 | 0.000 | 0.000 | 0.000 | 0.900 | 0.000 |
| 0.400 | 0.000 | 0.000 | 0.000 | 0.800 | 0.000 |
| 0.450 | 0.000 | 0.000 | 0.000 | 0.700 | 0.250 |
| 0.500 | 0.000 | 0.000 | 0.000 | 0.600 | 0.500 |
| 0.550 | 0.000 | 0.000 | 0.000 | 0.500 | 0.750 |
| 0.600 | 0.000 | 0.000 | 0.000 | 0.400 | 1.000 |
| 0.650 | 0.000 | 0.000 | 0.000 | 0.300 | 1.000 |
| 0.700 | 0.000 | 0.000 | 0.000 | 0.200 | 1.000 |
| 0.750 | 0.000 | 0.000 | 0.000 | 0.100 | 1.000 |
| 0.800 | 0.000 | 0.000 | 0.000 | -0.000 | 1.000 |
| 0.850 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| 0.900 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| 0.950 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |
| 1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 1.000 |

Se han realizado múltiples operaciones utilizando las funciones definidas en el programa y los conjuntos A, B, C, D y E, manteniendo la cantidad de puntos. Estas son como sigue:

- Operación I: $D \cup (A \cap C)$
- Operación II: $\overline{(A \cap B)} \cap (D \cap E)$
- Operación III: $\overline{D} \cap (B \cup A)$
- Operación IV: $\overline{A \cup B}$

El resultado de estas se muestra en la figura 2, correspondiente a la gráfica continua de un arreglo de puntos, los que son mostrados en la tabla 2.

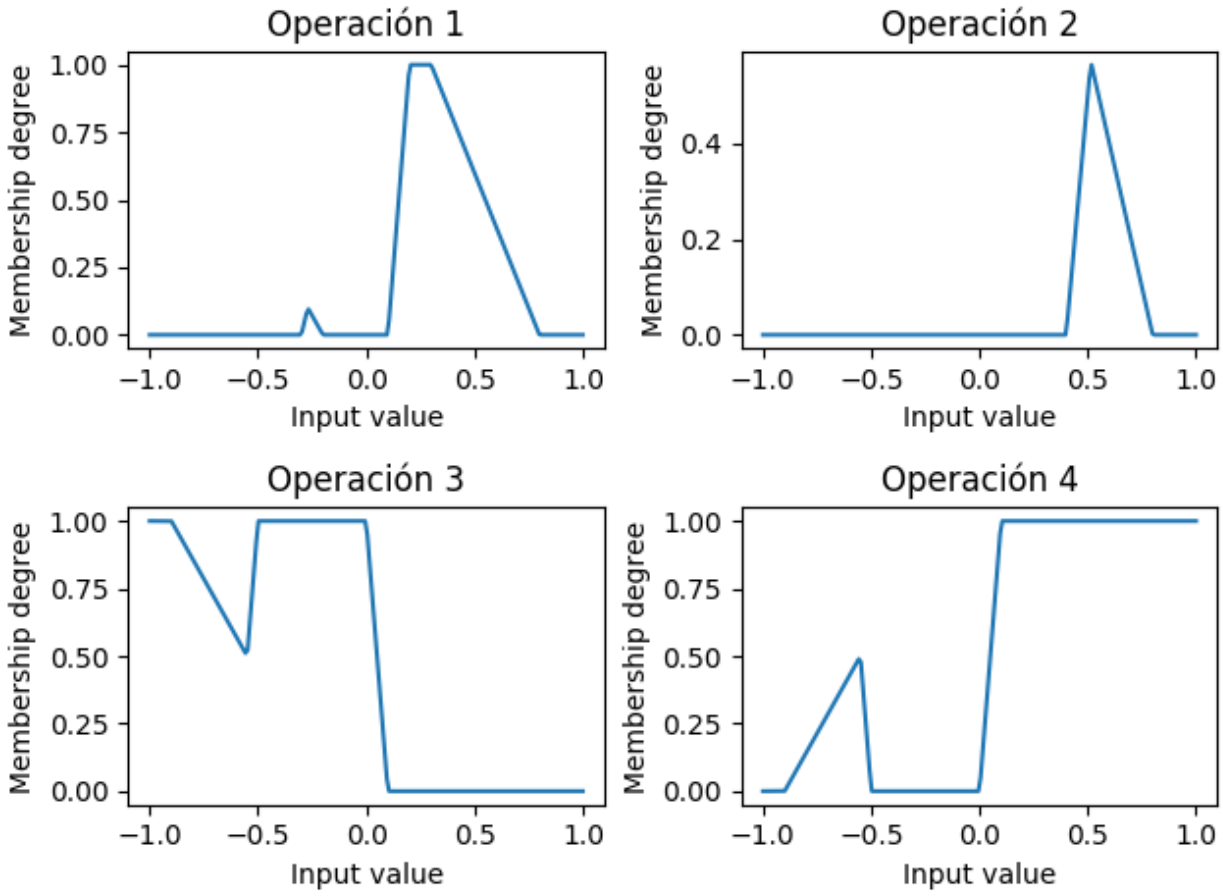


Figura 2: Resultados de operaciones realizadas.

Tabla 2: Resultado de operaciones.

| **X** | **OP 1** | **OP 2** | **OP 3** | **OP 4** |
|---|---|---|---|---|
| -1.000 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.950 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.900 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.850 | 0.000 | 0.000 | 0.929 | 0.071 |
| -0.800 | 0.000 | 0.000 | 0.857 | 0.143 |
| -0.750 | 0.000 | 0.000 | 0.786 | 0.214 |
| -0.700 | 0.000 | 0.000 | 0.714 | 0.286 |
| -0.650 | 0.000 | 0.000 | 0.643 | 0.357 |
| -0.600 | 0.000 | 0.000 | 0.571 | 0.429 |
| -0.550 | 0.000 | 0.000 | 0.500 | 0.500 |
| -0.500 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.450 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.400 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.350 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.300 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.250 | 0.071 | 0.000 | 1.000 | 0.000 |
| -0.200 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.150 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.100 | 0.000 | 0.000 | 1.000 | 0.000 |
| -0.050 | 0.000 | 0.000 | 1.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 1.000 | 0.000 |
| 0.050 | 0.000 | 0.000 | 0.500 | 0.500 |
| 0.100 | 0.000 | 0.000 | 0.000 | 1.000 |
| 0.150 | 0.500 | 0.000 | 0.000 | 1.000 |
| 0.200 | 1.000 | 0.000 | 0.000 | 1.000 |
| 0.250 | 1.000 | 0.000 | 0.000 | 1.000 |
| 0.300 | 1.000 | 0.000 | 0.000 | 1.000 |
| 0.350 | 0.900 | 0.000 | 0.000 | 1.000 |
| 0.400 | 0.800 | 0.000 | 0.000 | 1.000 |
| 0.450 | 0.700 | 0.250 | 0.000 | 1.000 |
| 0.500 | 0.600 | 0.500 | 0.000 | 1.000 |
| 0.550 | 0.500 | 0.500 | 0.000 | 1.000 |
| 0.600 | 0.400 | 0.400 | 0.000 | 1.000 |
| 0.650 | 0.300 | 0.300 | 0.000 | 1.000 |
| 0.700 | 0.200 | 0.200 | 0.000 | 1.000 |
| 0.750 | 0.100 | 0.100 | 0.000 | 1.000 |
| 0.800 | -0.000 | -0.000 | 0.000 | 1.000 |
| 0.850 | 0.000 | 0.000 | 0.000 | 1.000 |
| 0.900 | 0.000 | 0.000 | 0.000 | 1.000 |
| 0.950 | 0.000 | 0.000 | 0.000 | 1.000 |
| 1.000 | 0.000 | 0.000 | 0.000 | 1.000 |