Tarea 2

September 7, 2025

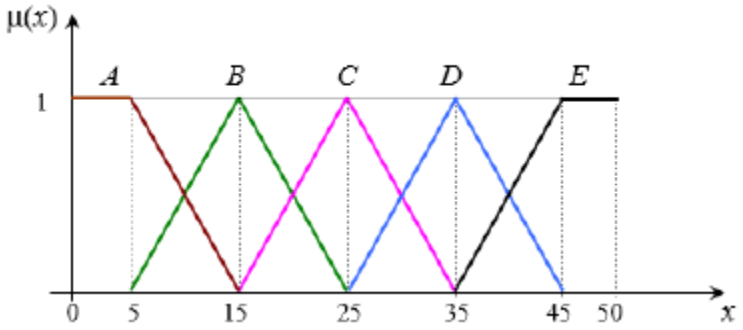
Sistemas Difusos

- *Maria Alejandra Bonilla Diaz - 20251595002*
- *Alvaro Alejandro Zarabanda Gutierrez – 20251595006*

[64]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Conjuntos difusos para temperatura de proceso industrial

Considere la siguiente función de pertenencia que describe 5 conjuntos fuzzy que representan la temperatura de un proceso industrial:



- **A:** Muy Baja
- **B:** Baja
- **C:** Media
- **D:** Alta
- **E:** Muy Alta

1.1 Expresiones analíticas para cada conjunto fuzzy

Cada conjunto fuzzy se describe mediante una función de membresía, que indica el grado de pertenencia de cada valor de temperatura al conjunto. Las expresiones se obtienen observando los puntos de intersección y los valores máximos de la gráfica.

- **A: Muy Baja**

$$\mu_A(x) = \begin{cases} 1 & 0 \leq x \leq 5 \\ \frac{15-x}{10} & 5 < x < 15 \\ 0 & x \geq 15 \end{cases}$$

Esta función es trapezoidal: es 1 en el intervalo [0,5], decrece linealmente hasta 0 en x=15.

- **B: Baja**

$$\mu_B(x) = \begin{cases} 0 & x \leq 5 \\ \frac{x-5}{10} & 5 < x < 15 \\ \frac{25-x}{10} & 15 \leq x < 25 \\ 0 & x \geq 25 \end{cases}$$

Es triangular, con máximo en x=15.

- **C: Media**

$$\mu_C(x) = \begin{cases} 0 & x \leq 15 \\ \frac{x-15}{10} & 15 < x < 25 \\ \frac{35-x}{10} & 25 \leq x < 35 \\ 0 & x \geq 35 \end{cases}$$

Es triangular, con máximo en x=25.

- **D: Alta**

$$\mu_D(x) = \begin{cases} 0 & x \leq 25 \\ \frac{x-25}{10} & 25 < x < 35 \\ \frac{45-x}{10} & 35 \leq x < 45 \\ 0 & x \geq 45 \end{cases}$$

Es triangular, con máximo en x=35.

- **E: Muy Alta**

$$\mu_E(x) = \begin{cases} 0 & x \leq 35 \\ \frac{x-35}{10} & 35 < x < 45 \\ 1 & 45 \leq x \leq 50 \end{cases}$$

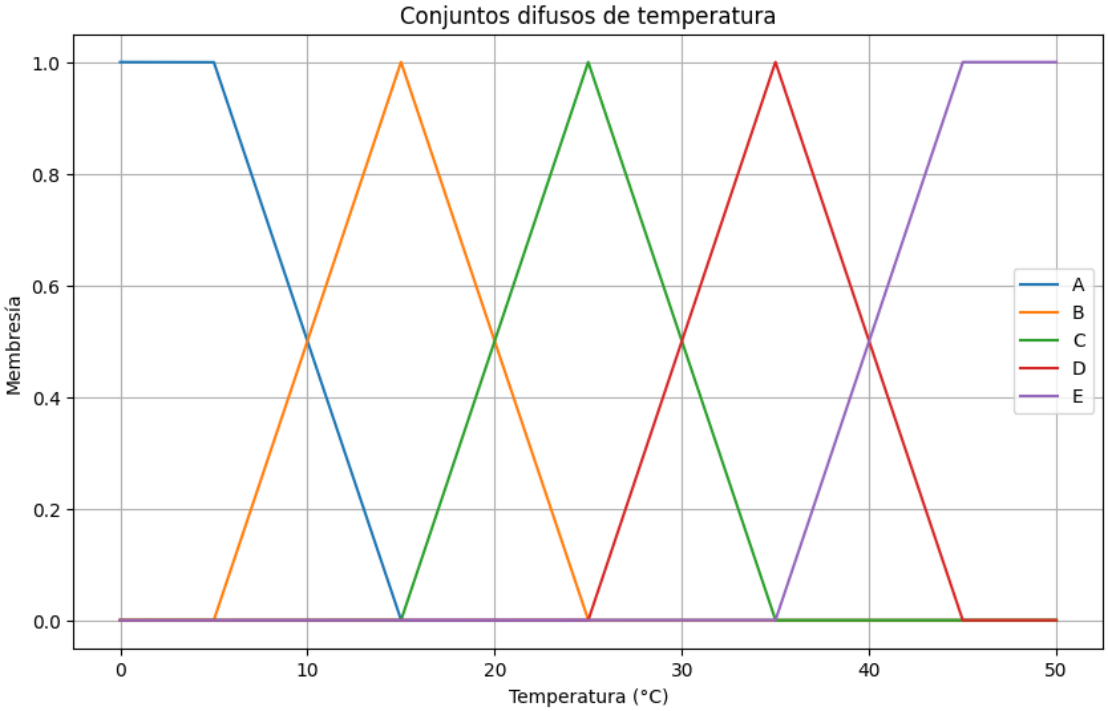
Es trapezoidal: crece linealmente entre 35 y 45, y es 1 en [45,50].

1.2 Procedimiento computacional para mapear los conjuntos fuzzy

Para mapear los conjuntos fuzzy, se discretiza el universo de discurso (temperatura) en N puntos (por ejemplo, $N = 1000$). Para cada conjunto, se calcula el valor de la función de membresía en cada punto usando las expresiones analíticas anteriores. Esto permite visualizar y analizar los conjuntos fuzzy de manera computacional.

[65]:
```python
def trapezoidal(x, a, m, n, b):
    x = np.asarray(x)
    valor = np.zeros_like(x, dtype=float)
    # Segmento izquierdo
    if m != a:
        mascara_izq = (x >= a) & (x < m)
        valor[mascara_izq] = (x[mascara_izq] - a) / (m - a)
    else:
        mascara_izq = (x >= a) & (x <= m)
        valor[mascara_izq] = 1
    # Segmento plano
    mascara_medio = (x >= m) & (x <= n)
    valor[mascara_medio] = 1
    # Segmento derecho
    if b != n:
        mascara_der = (x > n) & (x <= b)
        valor[mascara_der] = (b - x[mascara_der]) / (b - n)
    else:
        mascara_der = (x >= n) & (x <= b)
        valor[mascara_der] = 1
    # Fuera de soporte
    valor[(x < a) | (x > b)] = 0
    valor = np.clip(valor, 0, 1)
    return valor

def triangular(x, a, m, b):
    x = np.asarray(x)
    valor = np.zeros_like(x, dtype=float)
    # Lado izquierdo
    if m != a:
        mascara_izq = (x >= a) & (x <= m)
        valor[mascara_izq] = (x[mascara_izq] - a) / (m - a)
    else:
        mascara_izq = (x == a)
        valor[mascara_izq] = 1
    # Lado derecho
    if b != m:
        mascara_der = (x > m) & (x <= b)
        valor[mascara_der] = (b - x[mascara_der]) / (b - m)
    else:
        mascara_der = (x == b)
        valor[mascara_der] = 1
    # Fuera de soporte
    valor[(x < a) | (x > b)] = 0
    valor = np.clip(valor, 0, 1)
    return valor
```

```python
# Parámetros de los conjuntos
conjuntos = {
    'A': {'tipo': 'trapezoidal', 'parametros': [0, 0, 5, 15]},
    'B': {'tipo': 'triangular', 'parametros': [5, 15, 25]},
    'C': {'tipo': 'triangular', 'parametros': [15, 25, 35]},
    'D': {'tipo': 'triangular', 'parametros': [25, 35, 45]},
    'E': {'tipo': 'trapezoidal', 'parametros': [35, 45, 50, 50]}
}

# Función general

def membresia_difusa(x, tipo, parametros):
    if tipo == 'triangular':
        return triangular(x, *parametros)
    elif tipo == 'trapezoidal':
        return trapezoidal(x, *parametros)
    else:
        raise ValueError('Tipo de función no soportado')

# b) Mapeo de conjuntos difusos

def mapear_conjuntos(n_puntos=1000):
    universo = np.linspace(0, 50, n_puntos)
    membresias = {nombre: membresia_difusa(universo, datos['tipo'],␣
 ↪datos['parametros']) for nombre, datos in conjuntos.items()}
    return universo, membresias

universo, membresias = mapear_conjuntos(100000)

# Graficar
plt.figure(figsize=(10,6))
for nombre, y in membresias.items():
    plt.plot(universo, y, label=nombre)
plt.xlabel('Temperatura (°C)')
plt.ylabel('Membresía')
plt.title('Conjuntos difusos de temperatura')
plt.legend()
plt.grid(True)
plt.show()
```



El código define funciones generales para calcular la membresía de cualquier conjunto difuso, ya sea triangular o trapezoidal, evitando divisiones por cero. Los parámetros de cada conjunto se almacenan en un diccionario llamado `conjuntos`, donde se especifica el tipo y los valores característicos.

La función `mapear_conjuntos` discretiza el universo de temperatura en 1000 puntos y calcula la membresía de cada conjunto en todos esos puntos, generando los vectores necesarios para graficar las funciones de membresía. Finalmente, se visualizan todos los conjuntos difusos en una sola gráfica para comparar sus formas y solapamientos.

Este procedimiento permite analizar y visualizar cómo se distribuyen los grados de pertenencia para cada conjunto fuzzy en el rango de temperaturas estudiado.

1.3 Procedimiento para identificar conjuntos activos

Un conjunto fuzzy está activo para un valor x si su función de membresía es mayor que cero en ese punto. El procedimiento consiste en evaluar todas las funciones de membresía en x y listar aquellos conjuntos cuyo valor es positivo. Esto permite saber, para cualquier temperatura, qué conjuntos fuzzy la representan parcialmente.

[66]:
```
def conjuntos_activos(x):
    activos = []
    for nombre, datos in conjuntos.items():
        mu = membresia_difusa(np.array([x]), datos['tipo'],␣
↪datos['parametros'])[0]
        if mu > 0:
```

```
            activos.append((nombre, mu))
    return activos

# Ejemplo de uso
x_ejemplo = 10
activos = conjuntos_activos(x_ejemplo)
print(f'Conjuntos activos en x={x_ejemplo}:', activos)
```

```
Conjuntos activos en x=10: [('A', np.float64(0.5)), ('B', np.float64(0.5))]
```

El procedimiento para identificar los conjuntos difusos activos consiste en evaluar la función de membresía de cada conjunto para un valor específico de temperatura `x`. Si el resultado es mayor que cero, ese conjunto se considera activo para ese valor.

La función recorre todos los conjuntos definidos y retorna una lista con los nombres y el grado de pertenencia de los conjuntos que están activos en ese punto. Esto permite saber, para cualquier temperatura, qué etiquetas fuzzy la representan y con qué intensidad.

1.4 Procedimiento para calcular el nivel de pertenencia

El nivel de pertenencia de un conjunto fuzzy para un valor x es simplemente el valor de la función de membresía correspondiente en ese punto. Se evalúa la función analítica del conjunto en x y se retorna el resultado, que estará en el rango [0,1].

[67]:
```
def nivel_pertenencia(nombre, x):
    datos = conjuntos[nombre]
    return membresia_difusa(np.array([x]), datos['tipo'],␣
 ↪datos['parametros'])[0]

x_ejemplo = 25
nivel = nivel_pertenencia('C', x_ejemplo)
print(f'Nivel de pertenencia de C en x={x_ejemplo}:', nivel)
```

```
Nivel de pertenencia de C en x=25: 1.0
```

El procedimiento para calcular el nivel de pertenencia de un conjunto difuso dado un valor de temperatura `x` consiste en evaluar la función de membresía correspondiente en ese punto. El resultado es un valor entre 0 y 1 que indica el grado de pertenencia de `x` al conjunto seleccionado.

Este valor es útil para tomar decisiones basadas en lógica difusa, ya que permite cuantificar la pertenencia parcial de un valor a una etiqueta lingüística (por ejemplo, “media” o “alta”).

1.5 Procedimiento para obtener el conjunto crisp del α-corte

El α-corte de un conjunto fuzzy es el subconjunto de valores del universo de discurso donde la función de membresía es mayor o igual a α. Computacionalmente, se evalúa la función de membresía en todos los puntos discretizados y se seleccionan aquellos que cumplen la condición. El resultado es el conjunto crisp correspondiente al α-corte.

[68]:
```
def alpha_corte(nombre, alpha, universo):
    datos = conjuntos[nombre]
    mu = membresia_difusa(universo, datos['tipo'], datos['parametros'])
    return universo[mu >= alpha]

alpha = 0.999
corte = alpha_corte('C', alpha, universo)
print(f' -corte de C para  ={alpha}:', corte)
```

```
 -corte de C para  =0.999: [24.9902499  24.99074991 24.99124991 24.99174992
24.99224992 24.99274993
 24.99324993 24.99374994 24.99424994 24.99474995 24.99524995 24.99574996
 24.99624996 24.99674997 24.99724997 24.99774998 24.99824998 24.99874999
 24.99924999 24.99975    25.00025    25.00075001 25.00125001 25.00175002
 25.00225002 25.00275003 25.00325003 25.00375004 25.00425004 25.00475005
 25.00525005 25.00575006 25.00625006 25.00675007 25.00725007 25.00775008
 25.00825008 25.00875009 25.00925009 25.0097501 ]
```

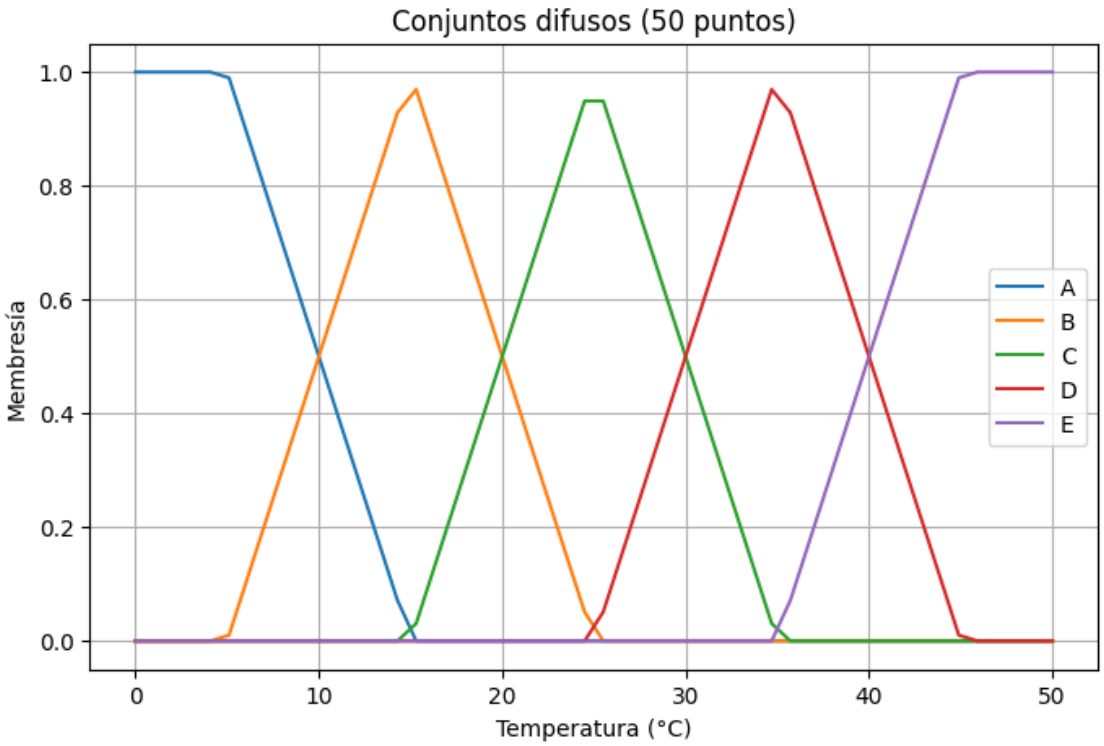
2 Operaciones entre conjuntos difusos

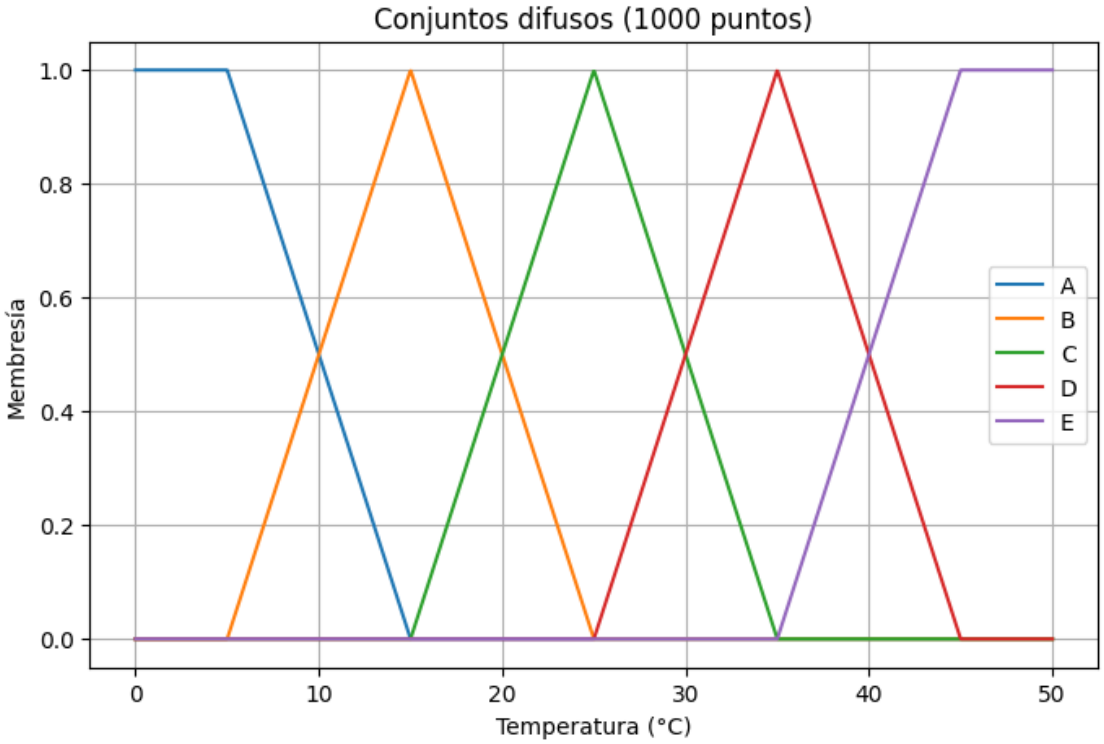
Basado en los procedimientos computacionales realizados anteriormente:

2.1 Graficar los cinco conjuntos fuzzy con 50 y 1000 puntos de discretización

Se grafican los cinco conjuntos difusos utilizando 50 y 1000 puntos de discretización. Con pocos puntos, las curvas se ven escalonadas y menos precisas, mientras que con muchos puntos se obtiene una representación suave y fiel a la función analítica. Una mayor discretización mejora la visualización y precisión en cálculos posteriores, aunque requiere más recursos computacionales.

[69]:
```
# Funciones para operaciones difusas
def graficar_conjuntos(n_puntos):
    universo, membresias = mapear_conjuntos(n_puntos)
    plt.figure(figsize=(8,5))
    for nombre, y in membresias.items():
        plt.plot(universo, y, label=nombre)
    plt.title(f'Conjuntos difusos ({n_puntos} puntos)')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
    return universo, membresias


# a) Graficar los cinco conjuntos fuzzy con 50 y 1000 puntos de discretización
_=graficar_conjuntos(50)
_=graficar_conjuntos(1000)
```

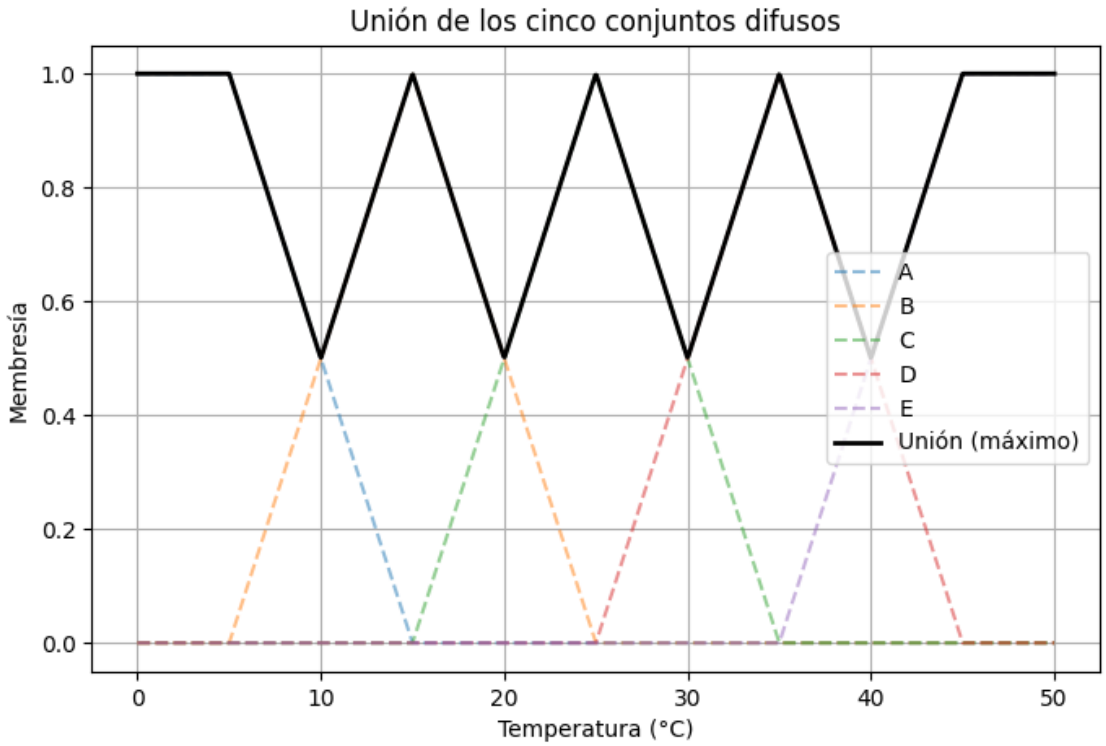

Conjuntos difusos (50 puntos)
Membresía
Temperatura (°C)
A
B
C
D
E


Conjuntos difusos (1000 puntos)
Membresía
Temperatura (°C)
A
B
C
D
E

2.2 Graficar la Unión de los cinco conjuntos fuzzy

La unión de los cinco conjuntos difusos se calcula usando el operador máximo. Para cada valor de temperatura, se toma el mayor grado de pertenencia entre todos los conjuntos. Esto permite identificar los rangos donde al menos uno de los conjuntos está activo y con qué intensidad. Es útil para modelar situaciones donde basta con que un conjunto esté activo para considerar la pertenencia.

[70]:
```
# b) Graficar la unión de los cinco conjuntos fuzzy (1000 puntos)

def union_max(membresias):
    return np.maximum.reduce(list(membresias.values()))

def graficar_union(n_puntos):
    universo, membresias = mapear_conjuntos(n_puntos)
    union = union_max(membresias)
    plt.figure(figsize=(8,5))
    for nombre, y in membresias.items():
        plt.plot(universo, y, '--', alpha=0.5, label=nombre)
    plt.plot(universo, union, 'k', linewidth=2, label='Unión (máximo)')
    plt.title('Unión de los cinco conjuntos difusos')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
    return universo, union

_=graficar_union(1000)
```

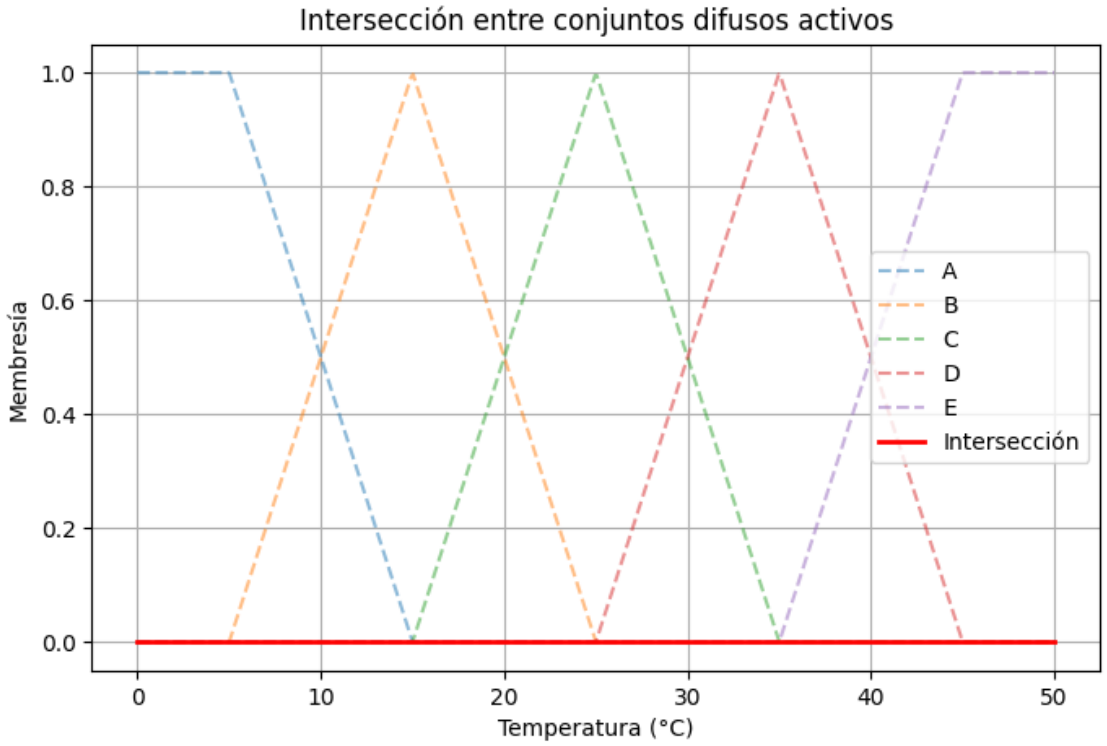


2.3 Graficar la Intersección de los cinco conjuntos fuzzy

La intersección se calcula tomando el mínimo entre los grados de pertenencia de los conjuntos en cada punto. Esto significa que la intersección solo es positiva en los intervalos donde al menos dos conjuntos se solapan. Es útil para modelar situaciones donde se requiere coincidencia entre etiquetas difusas.

[71]:
```
# c) Graficar la intersección entre conjuntos (500 puntos)

def interseccion_min(membresias):
    arr = np.array(list(membresias.values()))
    resultado = np.zeros(arr.shape[1])
    for i in range(arr.shape[1]):
        activos = arr[:, i][arr[:, i] >= 0]
        resultado[i] = np.min(activos) if len(activos) > 0 else 0
    return resultado

def graficar_interseccion_activos(n_puntos):
    universo, membresias = mapear_conjuntos(n_puntos)
    interseccion = interseccion_min(membresias)
    plt.figure(figsize=(8,5))
    for nombre, y in membresias.items():
        plt.plot(universo, y, '--', alpha=0.5, label=nombre)
```

```
    plt.plot(universo, interseccion, 'r', linewidth=2, label='Intersección')
    plt.title('Intersección entre conjuntos difusos activos')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
    return universo, interseccion


_=graficar_interseccion_activos(500)
```

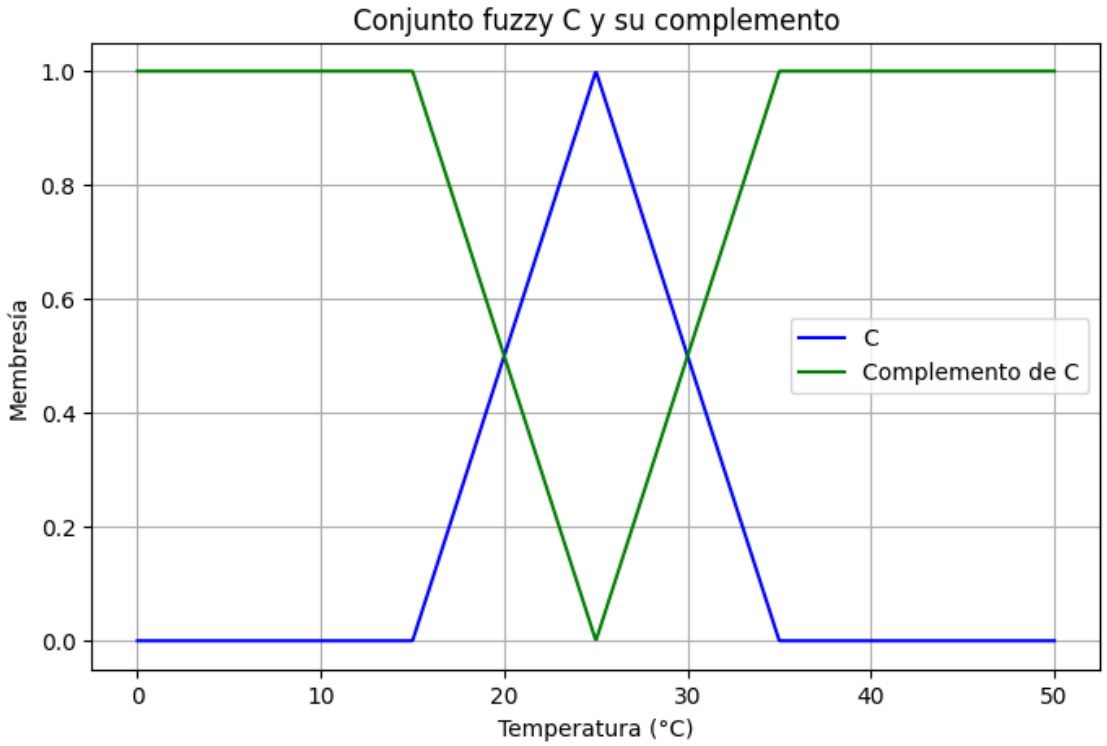


2.4 Graficar el Complemento sobre el conjunto fuzzy C

El complemento de un conjunto difuso indica el grado en que un valor NO pertenece al conjunto. Se calcula como 1 menos el grado de pertenencia del conjunto C en cada punto. Es útil para definir restricciones, exclusiones o reglas en sistemas difusos.

[72]:
```
# d) Graficar el complemento del conjunto fuzzy C (1000 puntos)

def complemento_conjunto(membresias, nombre):
    return 1 - membresias[nombre]
```

```python
def graficar_complemento_C(n_puntos):
    universo, membresias = mapear_conjuntos(n_puntos)
    complemento = complemento_conjunto(membresias, 'C')
    plt.figure(figsize=(8,5))
    plt.plot(universo, membresias['C'], 'b', label='C')
    plt.plot(universo, complemento, 'g', label='Complemento de C')
    plt.title('Conjunto fuzzy C y su complemento')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
    return universo, complemento
_=graficar_complemento_C(1000)
```



3 Operaciones entre conjuntos difusos activos

Basado en los procedimientos computacionales realizados anteriormente y considerando tan solo los conjuntos activos para una determinada temperatura, realice los siguientes gráficos:

3.1 Unión de los conjuntos activos en x=16.75.

3.2 Unión de los conjuntos activos en x=37.29.

3.3 Intersección de los conjuntos activos en x=20.

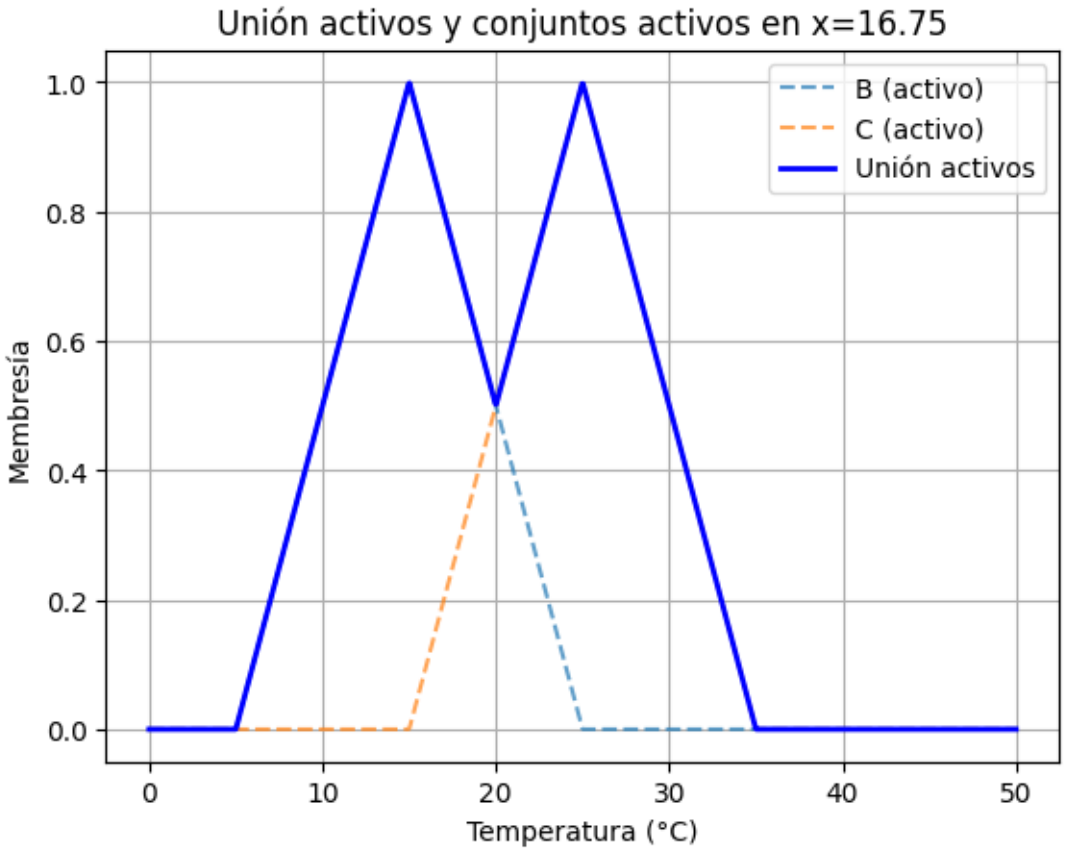
3.4 Intersección de los conjuntos activos en x=40.

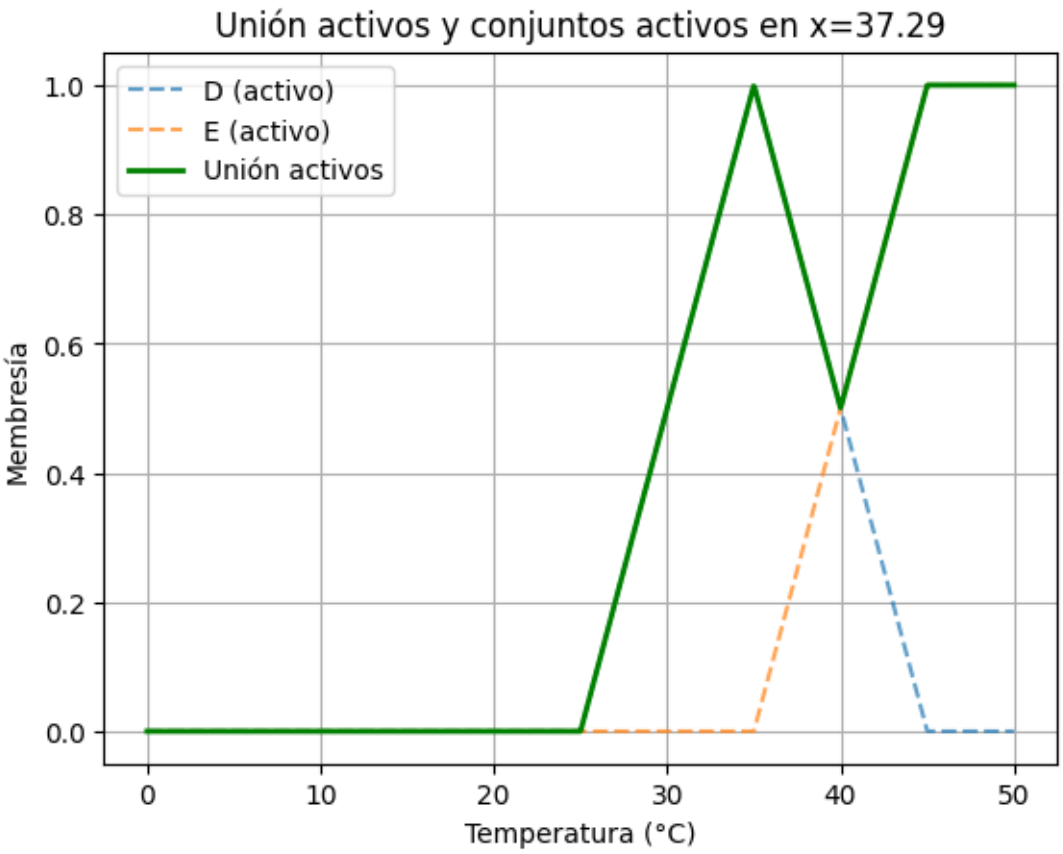
Imprima todos los gráficos de este ejercicio en la misma hoja, utilizando los operadores Máximo (Unión), Mínimo (Intersección) con 1000 puntos de discretización.

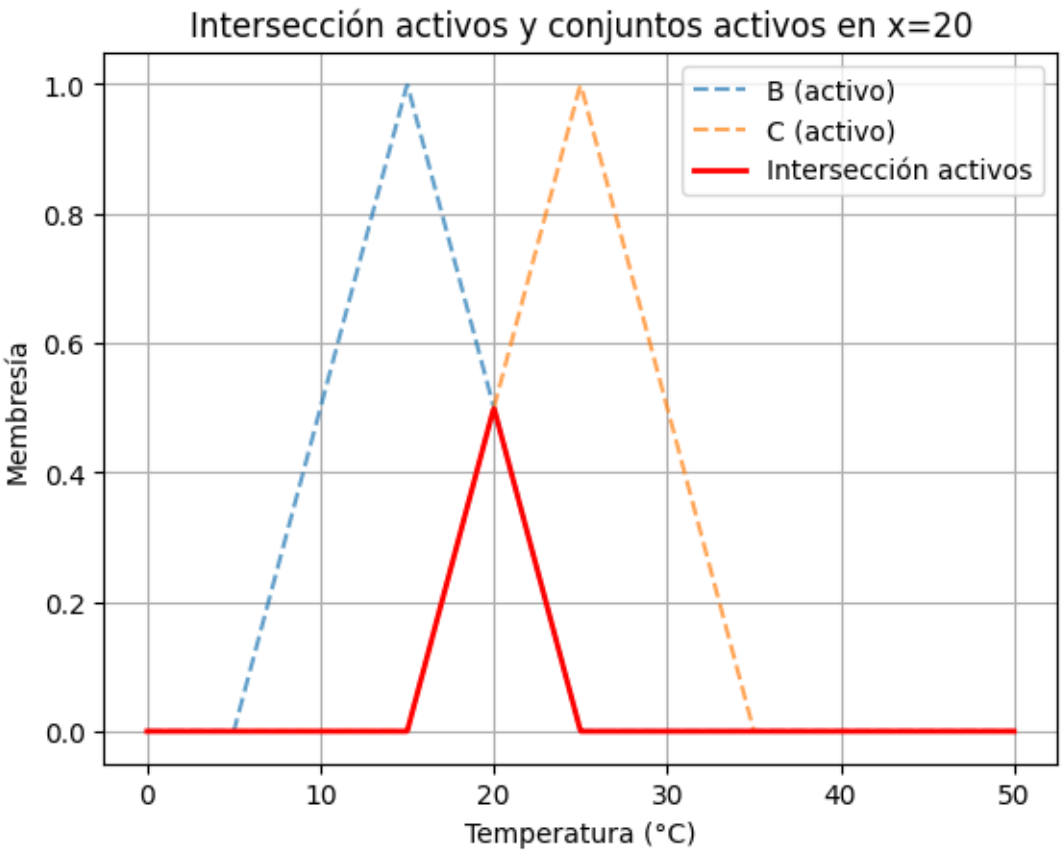
[73]:
```
universo = np.linspace(0, 50, 1000)

def membresias_activos(x, universo):
    activos = conjuntos_activos(x)
    return {nombre: membresia_difusa(universo, conjuntos[nombre]['tipo'],␣
 ↪conjuntos[nombre]['parametros']) for nombre, _ in activos}

# a) Unión de conjuntos activos en x=16.75
membresias_16_75 = membresias_activos(16.75, universo)
union_16_75 = union_max(membresias_16_75)

# b) Unión de conjuntos activos en x=37.29
membresias_37_29 = membresias_activos(37.29, universo)
union_37_29 = union_max(membresias_37_29)

# c) Intersección de conjuntos activos en x=20
membresias_20 = membresias_activos(20, universo)
interseccion_20 = interseccion_min(membresias_20)

# d) Intersección de conjuntos activos en x=40
membresias_40 = membresias_activos(40, universo)
interseccion_40 = interseccion_min(membresias_40)
```

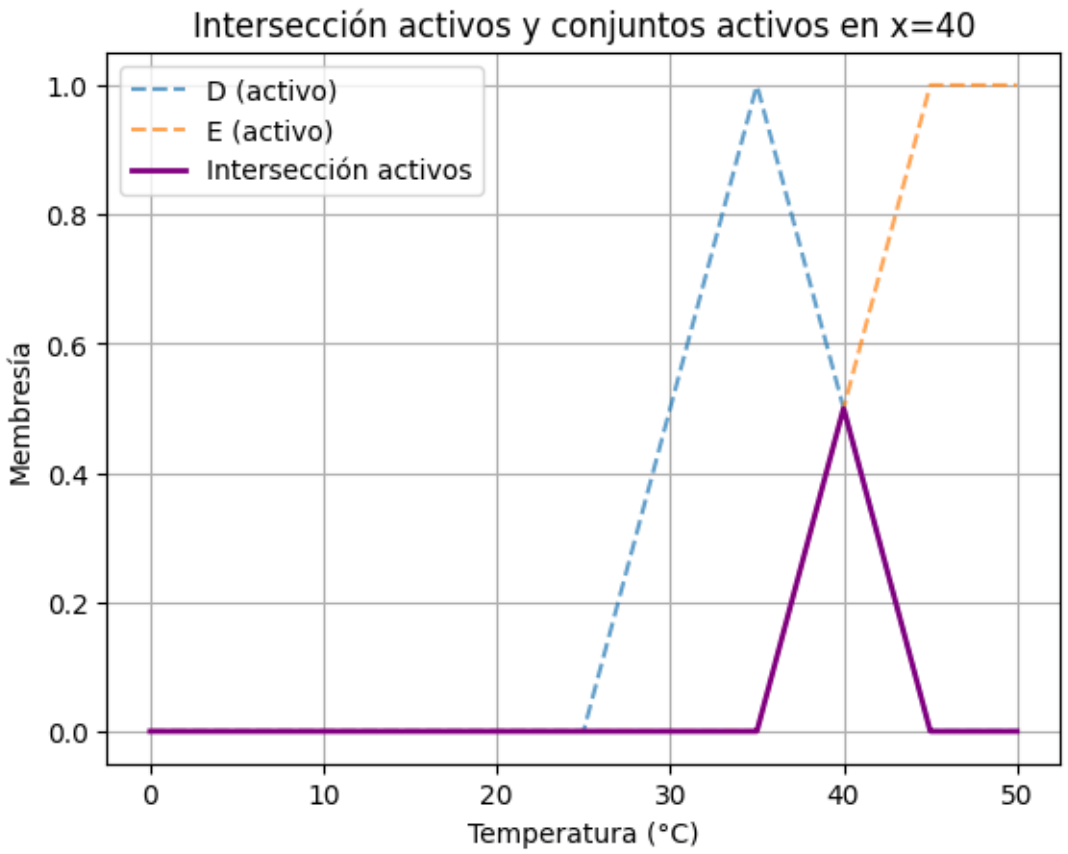
[74]:
```
# Gráficos separados mostrando solo los conjuntos fuzzy activos en cada caso
def plot_activos(x, universo, resultado, color, titulo):
    activos = conjuntos_activos(x)
    nombres = [nombre for nombre, _ in activos]
    for nombre in nombres:
        y = membresia_difusa(universo, conjuntos[nombre]['tipo'],␣
 ↪conjuntos[nombre]['parametros'])
        plt.plot(universo, y, '--', alpha=0.7, label=f'{nombre} (activo)')
    plt.plot(universo, resultado, color=color, linewidth=2, label=titulo)
    plt.title(f'{titulo} y conjuntos activos en x={x}')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
```

```
plot_activos(16.75, universo, union_16_75, 'blue', 'Unión activos')
plot_activos(37.29, universo, union_37_29, 'green', 'Unión activos')
plot_activos(20, universo, interseccion_20, 'red', 'Intersección activos')
plot_activos(40, universo, interseccion_40, 'purple', 'Intersección activos')
```

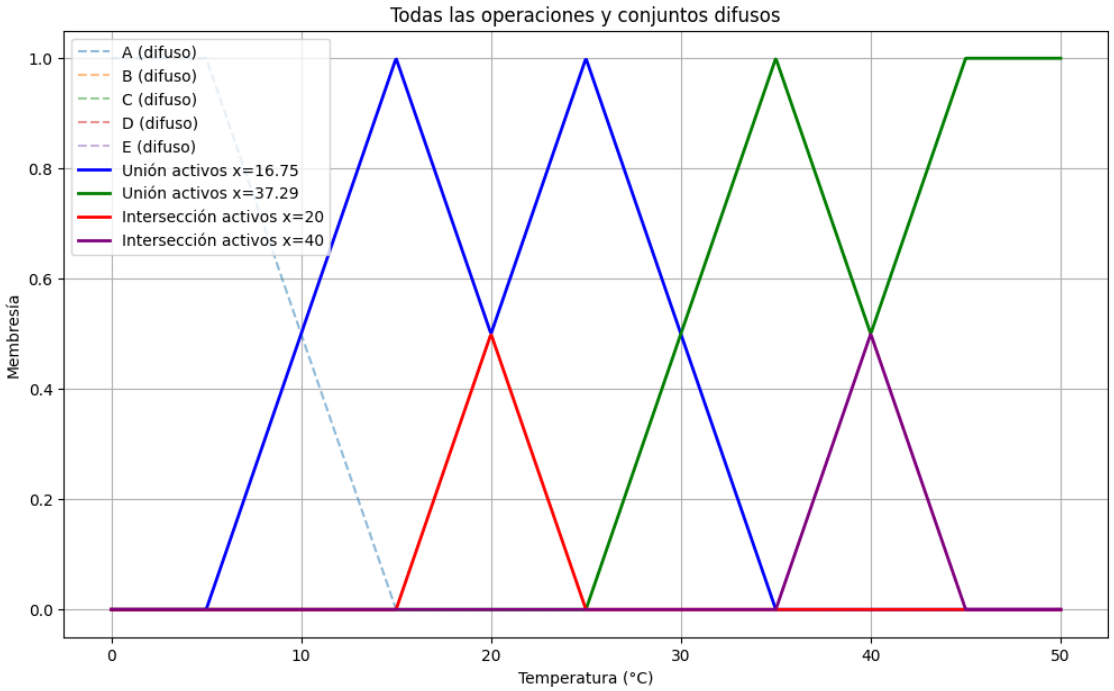



Unión activos y conjuntos activos en x=37.29
D (activo)
E (activo)
Unión activos
Membresía
Temperatura (°C)
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50


Intersección activos y conjuntos activos en x=20
B (activo)
C (activo)
Intersección activos
Membresía
Temperatura (°C)
0.0
0.2
0.4
0.6
0.8
1.0
0
10
20
30
40
50



[75]:
```
_, membresias_todos = mapear_conjuntos(1000)
plt.figure(figsize=(12,7))
for nombre, y in membresias_todos.items():
    plt.plot(universo, y, '--', alpha=0.5, label=f'{nombre} (difuso)')
plt.plot(universo, union_16_75, color='blue', linewidth=2, label='Unión activos␣
 ↪x=16.75')
plt.plot(universo, union_37_29, color='green', linewidth=2, label='Unión␣
 ↪activos x=37.29')
plt.plot(universo, interseccion_20, color='red', linewidth=2,␣
 ↪label='Intersección activos x=20')
plt.plot(universo, interseccion_40, color='purple', linewidth=2,␣
 ↪label='Intersección activos x=40')
plt.title('Todas las operaciones y conjuntos difusos')
plt.xlabel('Temperatura (°C)')
plt.ylabel('Membresía')
plt.legend()
plt.grid(True)
plt.show()
```



En este ejercicio se analizan las operaciones de unión e intersección entre los conjuntos difusos activos para valores específicos de temperatura, utilizando los operadores máximo (unión) y mínimo (intersección) sobre los conjuntos identificados como activos.

Unión de conjuntos activos: - Para cada temperatura dada (x=16.75 y x=37.29), se identifican los conjuntos difusos cuya función de membresía es mayor que cero (activos). - La unión se calcula tomando el máximo de las funciones de membresía de los conjuntos activos en cada punto del universo de temperatura. - El resultado representa el grado de pertenencia más alto que puede alcanzar cualquier conjunto activo para cada temperatura.

Intersección de conjuntos activos: - Para x=20 y x=40, se identifican los conjuntos activos y se calcula la intersección usando el operador mínimo. - La intersección muestra los rangos donde los conjuntos activos se solapan y el grado de pertenencia más bajo entre ellos. - Es útil para identificar zonas de ambigüedad o transición entre etiquetas difusas.

Interpretación de la gráfica: - Cada curva representa la operación (unión o intersección) entre los conjuntos activos para una temperatura específica. - La unión suele abarcar un rango más amplio y mostrar los máximos grados de pertenencia, mientras que la intersección es más restrictiva y resalta las zonas de solapamiento. - Visualizar todas las curvas en la misma figura permite comparar cómo varían las operaciones según la temperatura elegida y entender la dinámica de los conjuntos difusos en el sistema.

Estas operaciones son fundamentales en lógica difusa para la toma de decisiones, ya que permiten combinar información de diferentes etiquetas lingüísticas y modelar la incertidumbre inherente a variables físicas como la temperatura.

4 Operaciones con Suma y Producto Algebraico

Realice nuevamente el ejercicio anterior utilizando los operadores Suma Algebraica (Unión) y producto algebraico (Intersección).

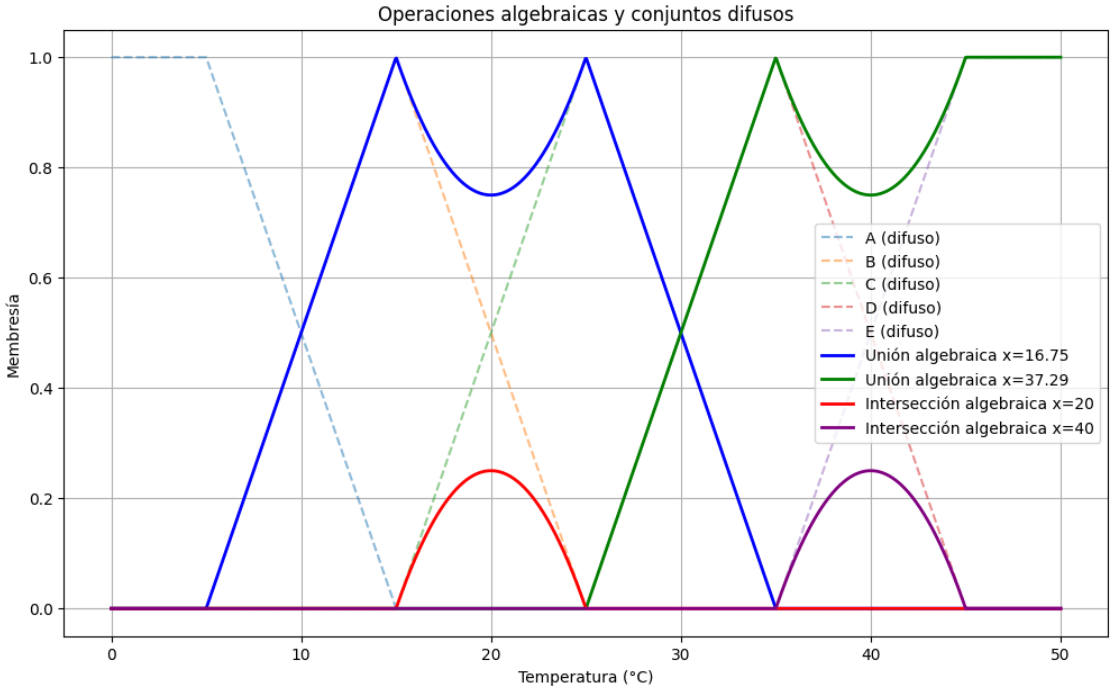
Imprima todos los gráficos de este ejercicio en la misma hoja.

[76]:
```
# --- Operadores algebraicos para conjuntos difusos ---
def union_suma_algebraica(membresias):
    vals = list(membresias.values())
    resultado = vals[0].copy()
    for arr in vals[1:]:
        resultado = resultado + arr - resultado * arr
    return np.clip(resultado, 0, 1)

def interseccion_producto_algebraico(membresias):
    vals = list(membresias.values())
    resultado = vals[0].copy()
    for arr in vals[1:]:
        resultado = resultado * arr
    return np.clip(resultado, 0, 1)
```

[77]:
```
# --- Operaciones fuzzy con suma y producto algebraico ---
universo = np.linspace(0, 50, 1000)

def membresias_activos(x, universo):
    activos = conjuntos_activos(x)
    return {nombre: membresia_difusa(universo, conjuntos[nombre]['tipo'],␣
 ↪conjuntos[nombre]['parametros']) for nombre, _ in activos}

# a) Unión algebraica de conjuntos activos en x=16.75
membresias_16_75_alg = membresias_activos(16.75, universo)
union_16_75_alg = union_suma_algebraica(membresias_16_75_alg)

# b) Unión algebraica de conjuntos activos en x=37.29
membresias_37_29_alg = membresias_activos(37.29, universo)
union_37_29_alg = union_suma_algebraica(membresias_37_29_alg)

# c) Intersección algebraica de conjuntos activos en x=20
membresias_20_alg = membresias_activos(20, universo)
interseccion_20_alg = interseccion_producto_algebraico(membresias_20_alg)

# d) Intersección algebraica de conjuntos activos en x=40
membresias_40_alg = membresias_activos(40, universo)
interseccion_40_alg = interseccion_producto_algebraico(membresias_40_alg)
```

[78]:
```
# Gráfico conjunto: suma y producto algebraico de todas las operaciones y␣
 ↪conjuntos difusos
```

```
_, membresias_todos = mapear_conjuntos(1000)
plt.figure(figsize=(12,7))
for nombre, y in membresias_todos.items():
    plt.plot(universo, y, '--', alpha=0.5, label=f'{nombre} (difuso)')
plt.plot(universo, union_16_75_alg, color='blue', linewidth=2, label='Unión␣
 ↪algebraica x=16.75')
plt.plot(universo, union_37_29_alg, color='green', linewidth=2, label='Unión␣
 ↪algebraica x=37.29')
plt.plot(universo, interseccion_20_alg, color='red', linewidth=2,␣
 ↪label='Intersección algebraica x=20')
plt.plot(universo, interseccion_40_alg, color='purple', linewidth=2,␣
 ↪label='Intersección algebraica x=40')
plt.title('Operaciones algebraicas y conjuntos difusos')
plt.xlabel('Temperatura (°C)')
plt.ylabel('Membresía')
plt.legend()
plt.grid(True)
plt.show()
```



Los operadores de suma y producto algebraico generan curvas más suaves y conservadoras que los operadores máximo y mínimo. La unión algebraica tiende a valores altos cuando varios conjuntos tienen membresía significativa, pero nunca supera el valor máximo. La intersección algebraica es más restrictiva, mostrando valores bajos salvo cuando todos los conjuntos activos tienen membresía alta. Esto permite modelar la combinación de información difusa de manera menos extrema, útil en sistemas donde se busca una agregación gradual y realista.

5 Operaciones compuestas entre conjuntos difusos

A partir de los procedimientos computacionales anteriores, imprima los gráficos resultantes de las siguientes operaciones:

5.1 $A \cup B \cup C$

5.2 $B \cap (C \cup D)$

5.3 $(A \cap B) \cup (B \cap C)$

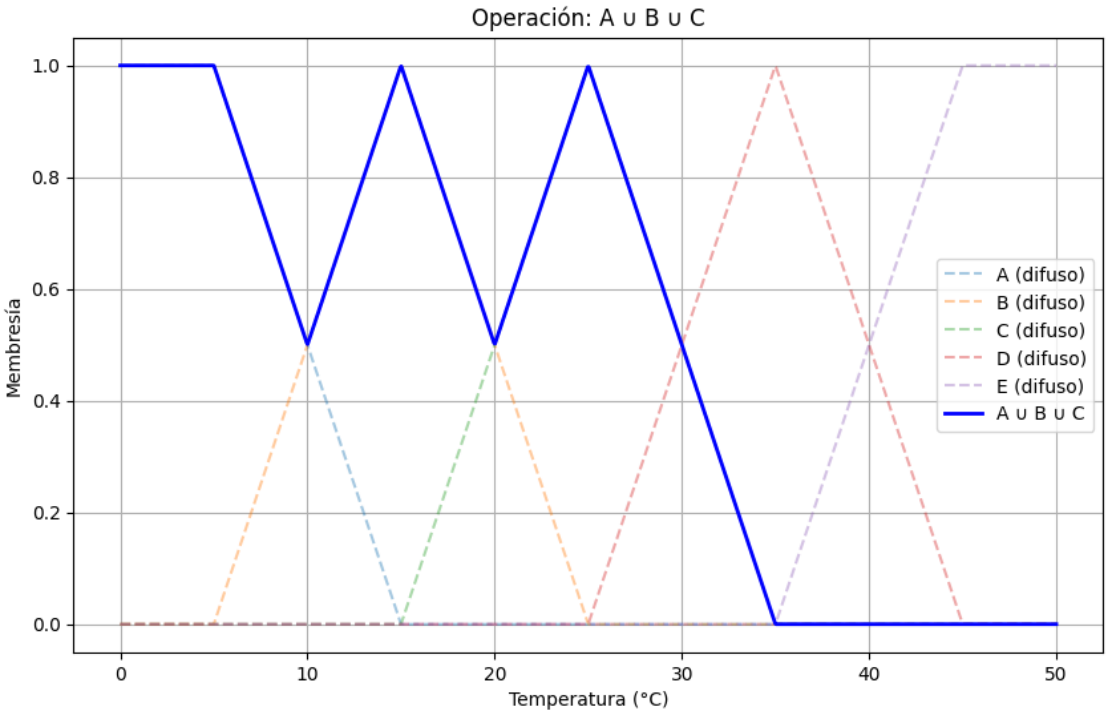
5.4 $\overline{A} \cup (B \cap C) \cup \overline{D}$

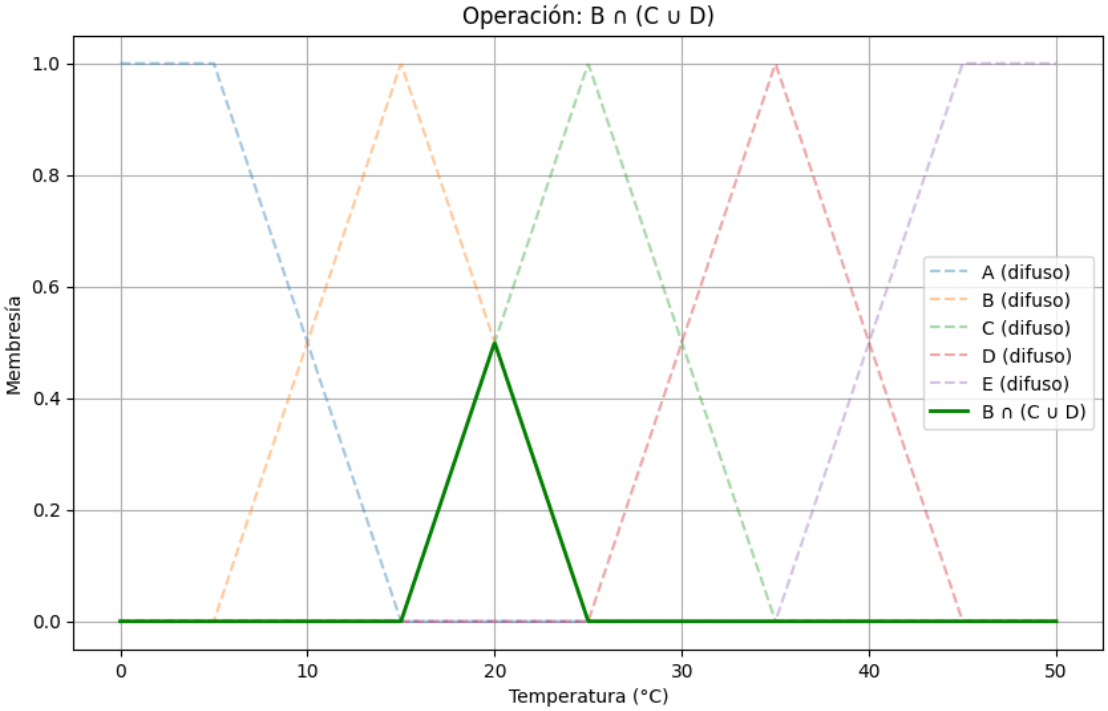
Imprima todos los gráficos de este ejercicio en la misma hoja, utilizando los operadores Máximo (Unión), Mínimo (Intersección) con 1000 puntos de discretización.

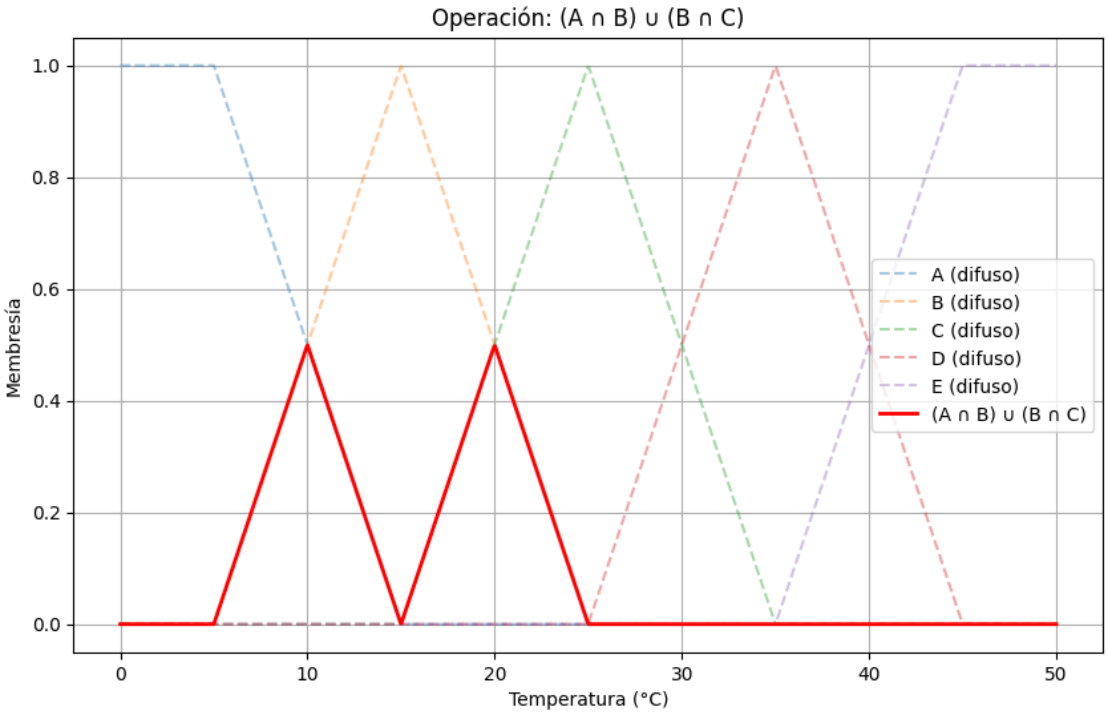
[79]:
```
# --- Punto 5: Operaciones compuestas entre conjuntos difusos ---
universo, membresias = mapear_conjuntos(1000)

# a) A   B   C
op_a = union_max({k: membresias[k] for k in ['A', 'B', 'C']})

# b) B   (C   D)
union_c_d = union_max({k: membresias[k] for k in ['C', 'D']})
op_b = interseccion_min({'B': membresias['B'], 'C D': union_c_d})

# c) (A   B)   (B   C)
inter_a_b = interseccion_min({'A': membresias['A'], 'B': membresias['B']})
inter_b_c = interseccion_min({'B': membresias['B'], 'C': membresias['C']})
op_c = union_max({'A B': inter_a_b, 'B C': inter_b_c})

# d) ¬A   (B   C)   ¬D
comp_a = complemento_conjunto(membresias, 'A')
inter_b_c = interseccion_min({'B': membresias['B'], 'C': membresias['C']})
comp_d = complemento_conjunto(membresias, 'D')
op_d = union_max({'¬A': comp_a, 'B C': inter_b_c, '¬D': comp_d})
```

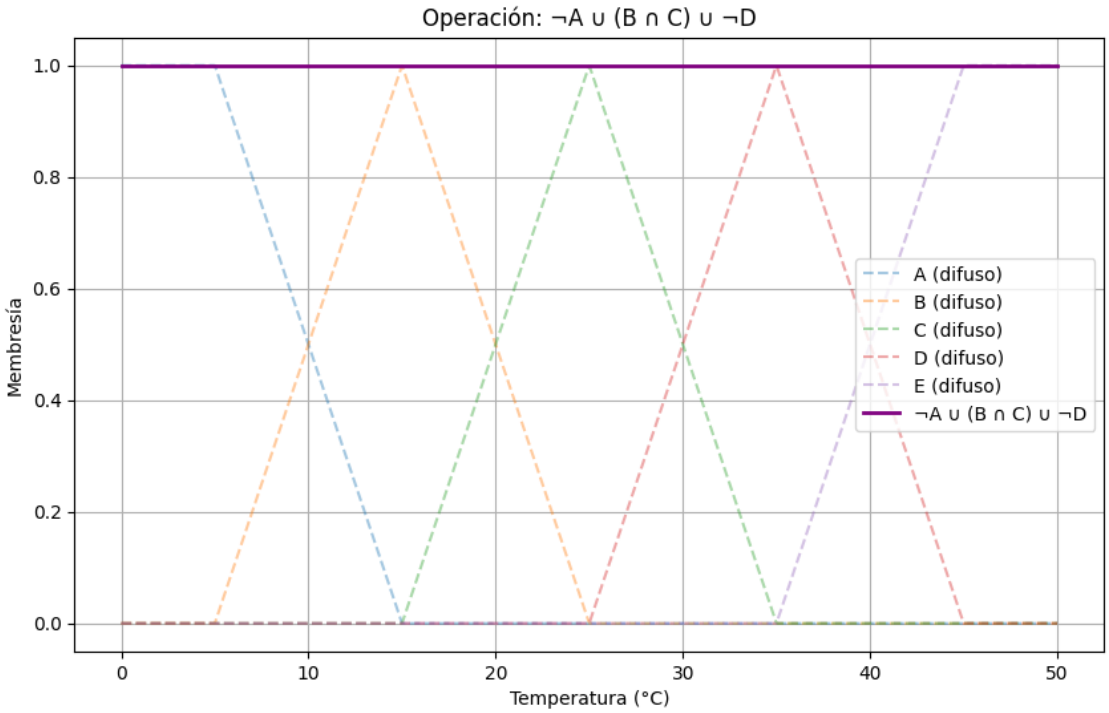
[80]:
```
# --- Gráficos separados para cada operación compuesta del punto 5 ---
operaciones = [
    (op_a, 'A   B   C', 'blue'),
    (op_b, 'B   (C   D)', 'green'),
    (op_c, '(A   B)   (B   C)', 'red'),
    (op_d, '¬A   (B   C)   ¬D', 'purple')
    ]

for resultado, titulo, color in operaciones:
    plt.figure(figsize=(10,6))
    for nombre, y in membresias.items():
        plt.plot(universo, y, '--', alpha=0.4, label=f'{nombre} (difuso)')
```

```
    plt.plot(universo, resultado, color=color, linewidth=2, label=titulo)
    plt.title(f'Operación: {titulo}')
    plt.xlabel('Temperatura (°C)')
    plt.ylabel('Membresía')
    plt.legend()
    plt.grid(True)
    plt.show()
```

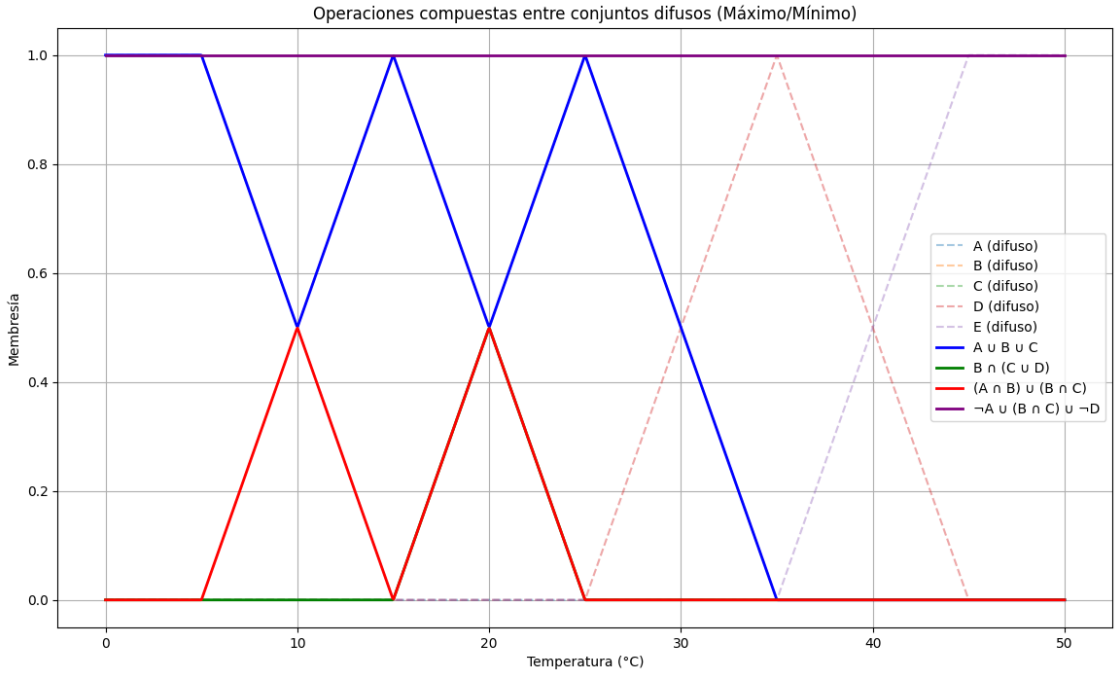



Operación: B ∩ (C ∪ D)
Membresía
Temperatura (°C)
A (difuso)
B (difuso)
C (difuso)
D (difuso)
E (difuso)
B ∩ (C ∪ D)


Operación: (A ∩ B) ∪ (B ∩ C)
Membresía
Temperatura (°C)
A (difuso)
B (difuso)
C (difuso)
D (difuso)
E (difuso)
(A ∩ B) ∪ (B ∩ C)



[81]:
```
# --- Gráfico conjunto ---
plt.figure(figsize=(14,8))
for nombre, y in membresias.items():
    plt.plot(universo, y, '--', alpha=0.4, label=f'{nombre} (difuso)')
plt.plot(universo, op_a, color='blue', linewidth=2, label='A  B  C')
plt.plot(universo, op_b, color='green', linewidth=2, label='B  (C  D)')
plt.plot(universo, op_c, color='red', linewidth=2, label='(A  B)  (B  C)')
plt.plot(universo, op_d, color='purple', linewidth=2, label='¬A  (B  C)  ¬D')
plt.title('Operaciones compuestas entre conjuntos difusos (Máximo/Mínimo)')
plt.xlabel('Temperatura (°C)')
plt.ylabel('Membresía')
plt.legend()
plt.grid(True)
plt.show()
```



6 Composición de relaciones fuzzy discretas

Considere tres relaciones fuzzy discretas que describen el nivel de relación entre las variables de un proceso, definidas por $R(x, y)$ y $S(y, z)$, donde $x \in X$, $y \in Y$ y $z \in Z$, con los respectivos universos de discurso $X = \{x_1, x_2, x_3, ..., x_{10}\}$; $Y = \{y_1, y_2, y_3, ..., y_5\}$ y $Z = \{z_1, z_2, z_3, ..., z_{15}\}$. A partir de la generación de valores aleatorios uniformemente distribuidos entre 0 y 1 para $R(x, y)$ y $S(y, z)$, realice lo siguiente:

6.0.1 Calcule e imprima la composición resultante de $R(x, y) \circ S(y, z)$.

6.0.2 Calcule e imprima la composición resultante de $R(x, y) \cdot S(y, z)$.

6.0.3 Calcule e imprima la composición resultante de $R(x, y) \oplus S(y, z)$.

[82]:
```python
# --- Punto 6: Composición de relaciones fuzzy discretas ---
import numpy as np
np.random.seed(42)  # Para reproducibilidad

# Universos de discurso
X = [f'x{i+1}' for i in range(10)]
Y = [f'y{j+1}' for j in range(5)]
Z = [f'z{k+1}' for k in range(15)]

# Relaciones fuzzy aleatorias
R = np.random.uniform(0, 1, (len(X), len(Y)))  # R(x, y)
```

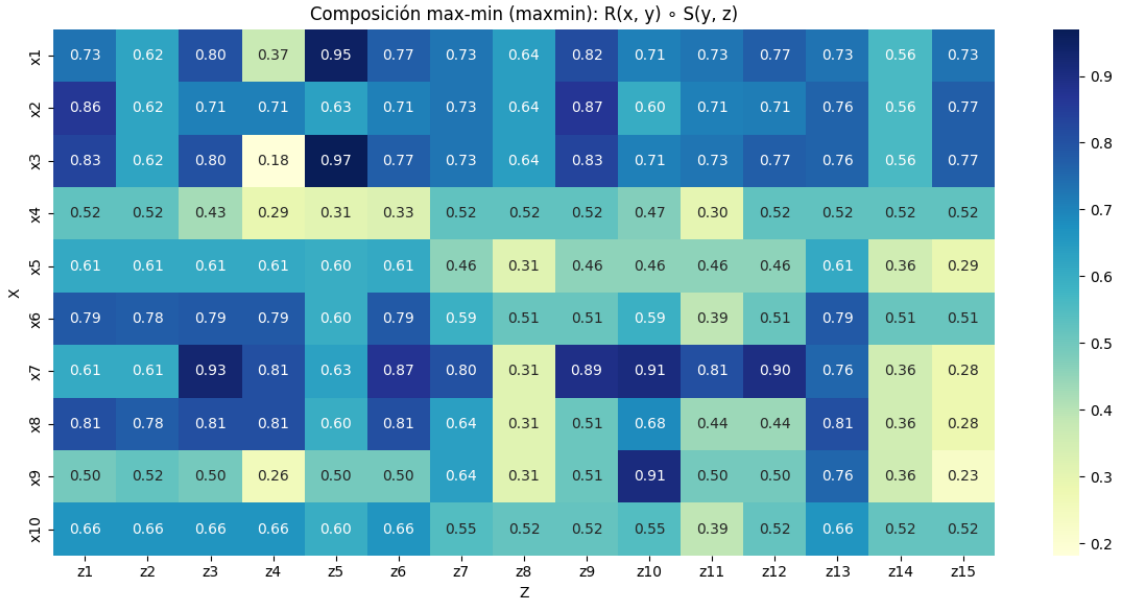
```
S = np.random.uniform(0, 1, (len(Y), len(Z)))  # S(y, z)

def composicion_maxmin(R, S):
    T = np.zeros((R.shape[0], S.shape[1]))
    for i in range(R.shape[0]):
        for k in range(S.shape[1]):
            T[i, k] = np.max([np.minimum(R[i, j], S[j, k]) for j in range(R.
 ↪shape[1])])
    return T

def composicion_maxprod(R, S):
    T = np.zeros((R.shape[0], S.shape[1]))
    for i in range(R.shape[0]):
        for k in range(S.shape[1]):
            T[i, k] = np.max([R[i, j] * S[j, k] for j in range(R.shape[1])])
    return T

def composicion_maxmedia(R, S):
    T = np.zeros((R.shape[0], S.shape[1]))
    for i in range(R.shape[0]):
        for k in range(S.shape[1]):
            T[i, k] = np.max([0.5 * (R[i, j] + S[j, k]) for j in range(R.
 ↪shape[1])])
    return T

# Calcular y mostrar resultados
T_maxmin = composicion_maxmin(R, S)
T_maxprod = composicion_maxprod(R, S)
T_maxmedia = composicion_maxmedia(R, S)

print('a) Composición (composicion_maxmin):')
print(np.round(T_maxmin, 3))

print('\nb) (composicion_maxprod):')
print(np.round(T_maxprod, 3))

print('\nc) (composicion_maxmedia):')
print(np.round(T_maxmedia, 3))
```

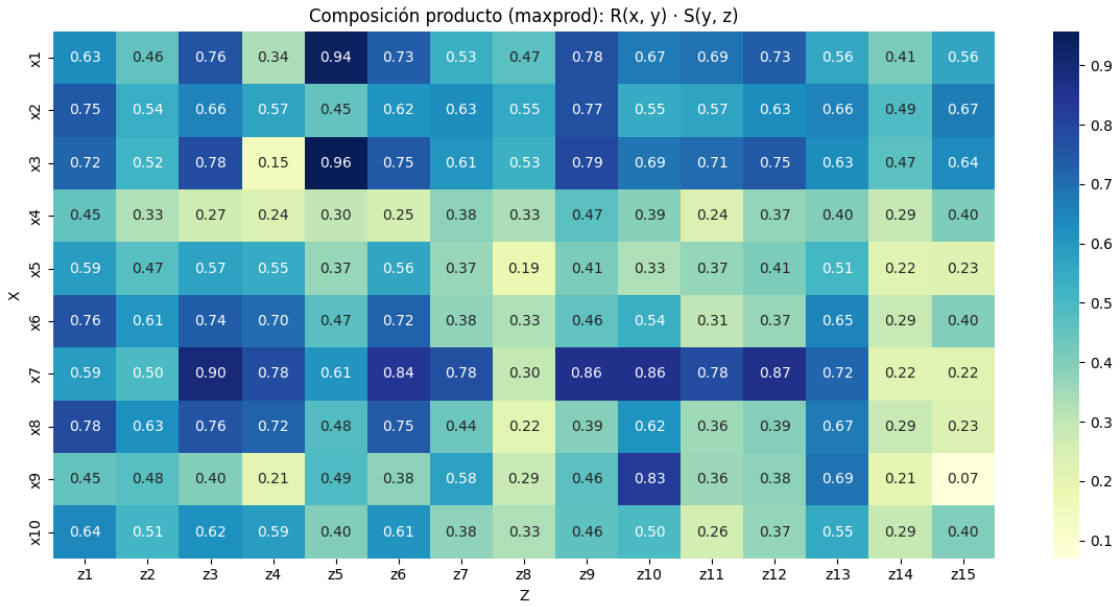
```
a) Composición (composicion_maxmin):
[[0.732 0.623 0.802 0.375 0.951 0.772 0.73  0.638 0.815 0.707 0.729 0.771
  0.732 0.561 0.732]
 [0.863 0.623 0.708 0.708 0.633 0.708 0.73  0.638 0.866 0.601 0.708 0.713
  0.761 0.561 0.771]
 [0.832 0.623 0.802 0.182 0.97  0.772 0.73  0.638 0.832 0.707 0.729 0.771
  0.761 0.561 0.771]
 [0.525 0.525 0.428 0.291 0.311 0.325 0.525 0.525 0.525 0.472 0.304 0.525
```

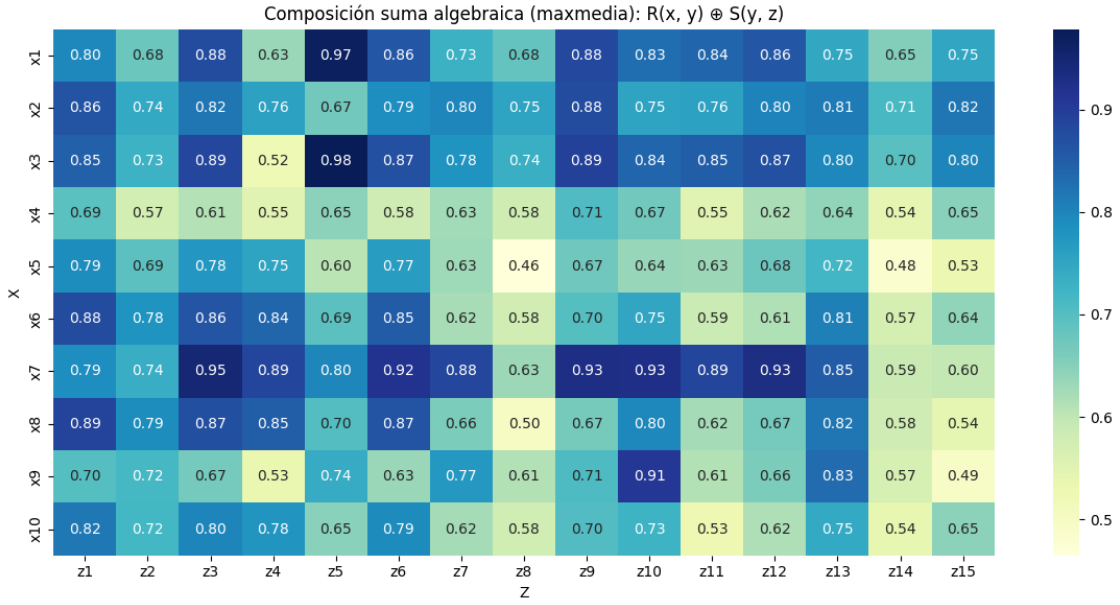
```
  0.525 0.525 0.525]
 [0.612 0.612 0.612 0.612 0.598 0.612 0.456 0.314 0.456 0.456 0.456 0.456
  0.612 0.357 0.292]
 [0.785 0.775 0.785 0.785 0.598 0.785 0.592 0.514 0.514 0.592 0.389 0.514
  0.785 0.514 0.514]
 [0.608 0.608 0.93  0.808 0.633 0.871 0.804 0.314 0.893 0.908 0.807 0.896
  0.756 0.357 0.281]
 [0.808 0.775 0.808 0.808 0.598 0.808 0.636 0.314 0.509 0.684 0.44  0.44
  0.808 0.357 0.281]
 [0.495 0.523 0.495 0.259 0.495 0.495 0.636 0.314 0.509 0.908 0.495 0.495
  0.756 0.358 0.228]
 [0.663 0.663 0.663 0.663 0.598 0.663 0.547 0.52  0.52  0.547 0.389 0.52
  0.663 0.52  0.52 ]]

b) (composicion_maxprod):
[[0.632 0.456 0.763 0.335 0.938 0.734 0.534 0.467 0.775 0.672 0.693 0.733
  0.557 0.411 0.564]
 [0.748 0.54  0.658 0.572 0.448 0.617 0.632 0.552 0.768 0.546 0.572 0.634
  0.659 0.486 0.668]
 [0.718 0.519 0.778 0.147 0.957 0.749 0.607 0.531 0.791 0.686 0.707 0.748
  0.633 0.467 0.642]
 [0.453 0.327 0.271 0.235 0.3   0.254 0.383 0.335 0.466 0.392 0.235 0.374
  0.399 0.295 0.405]
 [0.593 0.474 0.575 0.548 0.366 0.564 0.367 0.186 0.407 0.332 0.368 0.409
  0.507 0.218 0.225]
 [0.761 0.609 0.738 0.703 0.469 0.724 0.377 0.328 0.456 0.538 0.305 0.367
  0.651 0.289 0.396]
 [0.589 0.496 0.898 0.78  0.612 0.842 0.776 0.298 0.862 0.861 0.78  0.865
  0.717 0.217 0.22 ]
 [0.784 0.627 0.759 0.723 0.483 0.745 0.435 0.215 0.393 0.621 0.355 0.394
  0.67  0.288 0.227]
 [0.449 0.475 0.397 0.209 0.489 0.382 0.579 0.286 0.462 0.825 0.361 0.382
  0.687 0.208 0.07 ]
 [0.642 0.514 0.622 0.593 0.396 0.611 0.379 0.332 0.461 0.496 0.258 0.371
  0.549 0.292 0.401]]

c) (composicion_maxmedia):
[[0.798 0.678 0.876 0.635 0.969 0.861 0.731 0.685 0.883 0.829 0.84  0.861
  0.746 0.655 0.751]
 [0.865 0.745 0.819 0.758 0.671 0.79  0.798 0.752 0.877 0.754 0.758 0.802
  0.813 0.714 0.819]
 [0.848 0.728 0.886 0.522 0.978 0.871 0.781 0.735 0.893 0.838 0.849 0.871
  0.797 0.697 0.802]
 [0.694 0.574 0.61  0.55  0.646 0.581 0.627 0.581 0.706 0.67  0.549 0.619
  0.643 0.543 0.648]
 [0.791 0.693 0.776 0.753 0.605 0.767 0.63  0.465 0.674 0.637 0.632 0.676
  0.72  0.484 0.532]
 [0.877 0.78  0.862 0.84  0.692 0.854 0.622 0.576 0.701 0.75  0.587 0.614
```

```
 0.807 0.571 0.643]
[0.789 0.736 0.948 0.887 0.8   0.919 0.885 0.632 0.929 0.928 0.887 0.931
 0.852 0.589 0.597]
[0.889 0.792 0.874 0.852 0.703 0.865 0.66  0.502 0.666 0.796 0.624 0.668
 0.819 0.583 0.545]
[0.702 0.716 0.668 0.533 0.741 0.634 0.773 0.612 0.709 0.908 0.612 0.66
 0.832 0.569 0.493]
[0.816 0.719 0.801 0.779 0.649 0.792 0.625 0.579 0.704 0.727 0.526 0.617
 0.746 0.541 0.646]]
```

[83]:
```python
# Visualización de las composiciones fuzzy (maxmin, maxprod, maxmedia) como mapas de calor

def plot_heatmap(matrix, x_labels, y_labels, title):
    plt.figure(figsize=(12, 6))
    sns.heatmap(matrix, annot=True, fmt='.2f', cmap='YlGnBu', xticklabels=y_labels, yticklabels=x_labels)
    plt.title(title)
    plt.xlabel('Z')
    plt.ylabel('X')
    plt.tight_layout()
    plt.show()

plot_heatmap(T_maxmin, X, Z, 'Composición max-min (maxmin): R(x, y)  S(y, z)')
plot_heatmap(T_maxprod, X, Z, 'Composición producto (maxprod): R(x, y) · S(y, z)')
plot_heatmap(T_maxmedia, X, Z, 'Composición suma algebraica (maxmedia): R(x, y)  S(y, z)')
```




Composición producto (maxprod): R(x, y) · S(y, z)
x1 0.63 0.46 0.76 0.34 0.94 0.73 0.53 0.47 0.78 0.67 0.69 0.73 0.56 0.41 0.56
x2 0.75 0.54 0.66 0.57 0.45 0.62 0.63 0.55 0.77 0.55 0.57 0.63 0.66 0.49 0.67
x3 0.72 0.52 0.78 0.15 0.96 0.75 0.61 0.53 0.79 0.69 0.71 0.75 0.63 0.47 0.64
x4 0.45 0.33 0.27 0.24 0.30 0.25 0.38 0.33 0.47 0.39 0.24 0.37 0.40 0.29 0.40
x5 0.59 0.47 0.57 0.55 0.37 0.56 0.37 0.19 0.41 0.33 0.37 0.41 0.51 0.22 0.23
x6 0.76 0.61 0.74 0.70 0.47 0.72 0.38 0.33 0.46 0.54 0.31 0.37 0.65 0.29 0.40
x7 0.59 0.50 0.90 0.78 0.61 0.84 0.78 0.30 0.86 0.86 0.78 0.87 0.72 0.22 0.22
x8 0.78 0.63 0.76 0.72 0.48 0.75 0.44 0.22 0.39 0.62 0.36 0.39 0.67 0.29 0.23
x9 0.45 0.48 0.40 0.21 0.49 0.38 0.58 0.29 0.46 0.83 0.36 0.38 0.69 0.21 0.07
x10 0.64 0.51 0.62 0.59 0.40 0.61 0.38 0.33 0.46 0.50 0.26 0.37 0.55 0.29 0.40
X
z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 z12 z13 z14 z15
Z
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1


Composición suma algebraica (maxmedia): R(x, y) ⊕ S(y, z)
x1 0.80 0.68 0.88 0.63 0.97 0.86 0.73 0.68 0.88 0.83 0.84 0.86 0.75 0.65 0.75
x2 0.86 0.74 0.82 0.76 0.67 0.79 0.80 0.75 0.88 0.75 0.76 0.80 0.81 0.71 0.82
x3 0.85 0.73 0.89 0.52 0.98 0.87 0.78 0.74 0.89 0.84 0.85 0.87 0.80 0.70 0.80
x4 0.69 0.57 0.61 0.55 0.65 0.58 0.63 0.58 0.71 0.67 0.55 0.62 0.64 0.54 0.65
x5 0.79 0.69 0.78 0.75 0.60 0.77 0.63 0.46 0.67 0.64 0.63 0.68 0.72 0.48 0.53
x6 0.88 0.78 0.86 0.84 0.69 0.85 0.62 0.58 0.70 0.75 0.59 0.61 0.81 0.57 0.64
x7 0.79 0.74 0.95 0.89 0.80 0.92 0.88 0.63 0.93 0.93 0.89 0.93 0.85 0.59 0.60
x8 0.89 0.79 0.87 0.85 0.70 0.87 0.66 0.50 0.67 0.80 0.62 0.67 0.82 0.58 0.54
x9 0.70 0.72 0.67 0.53 0.74 0.63 0.77 0.61 0.71 0.91 0.61 0.66 0.83 0.57 0.49
x10 0.82 0.72 0.80 0.78 0.65 0.79 0.62 0.58 0.70 0.73 0.53 0.62 0.75 0.54 0.65
X
z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 z12 z13 z14 z15
Z
0.9 0.8 0.7 0.6 0.5