Relatório sobre Conjuntos, Funções e Operadores Fuzzy

Doutorado CEFET

May 7, 2025

1 Introdução

Este relatório apresenta a implementação e análise de funções de pertinência, fuzzificação, operações fuzzy e relações fuzzy. As funções de pertinência são amplamente utilizadas em lógica fuzzy para representar graus de pertencimento de elementos a conjuntos fuzzy.

2 Funções de Pertinência

2.1 Implementação de Funcões de Pertinência

As funções de pertinência implementadas incluem:

- Triangular
- Trapezoidal
- Gaussiana
- Sigmoidal
- Sinoidal (Bell)
- Função S
- Função Z
- Cauchy
- Gaussiana Dupla

- Logarítmica
- Retangular

Para a construção dos gráficos utilizando python utilizamos a função $plot_results$

```
def plot_results(x, y, params, type):
    # Plotando a função
    plt.figure(figsize=(8, 5))
    plt.plot(x, y, label=f"{type.capitalize()} {params}")
    plt.title(f"Função {type.capitalize()}")
    plt.xlabel("x")
    plt.ylabel("Grau de Pertinência")
    plt.legend()
    plt.grid()
    plt.savefig(f'{output_dir}/{type}.png')
    plt.show()
```

2.1.1 Função Triangular

A função triangular é definida por três parâmetros (a, b, c), onde:

- a é o ponto inicial onde a pertinência começa a aumentar;
- b é o ponto onde a pertinência atinge o valor máximo (1);
- c é o ponto final onde a pertinência retorna a 0.

A fórmula é dada por:

$$\mu(x) = \begin{cases} \frac{x-a}{b-a}, & \text{se } a \leq x < b, \\ \frac{c-x}{c-b}, & \text{se } b \leq x < c, \\ 0, & \text{caso contrário.} \end{cases}$$

O código Python correspondente é:

```
def triangular(x, a, b, c):
    if a <= x < b:
        return (x - a) / (b - a)
    elif b <= x < c:
        return (c - x) / (c - b)
    else:
```

```
        return 0

# Exemplo de plotagem para a função triangular
a, b, c = 5, 10, 15  # Parâmetros da função triangular
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]

# Calcula os graus de pertinência
y = [triangular(val, a, b, c) for val in x]

# Plotando a função triangular
plot_results(x, y, [a, b, c], "triangular")
```

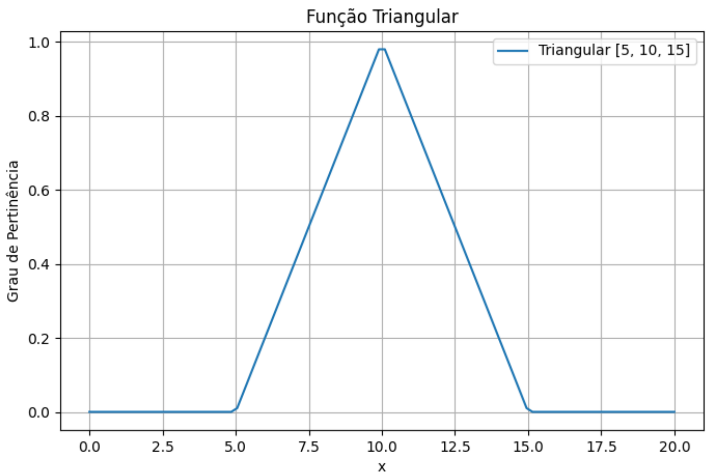

Figure 1: Exemplo de função triangular com $a = 5$, $b = 10$, $c = 15$.

2.1.2 Função Trapezoidal

A função trapezoidal é definida por quatro parâmetros (a, b, c, d), onde:

- a e d são os pontos onde a pertinência é 0;
- b e c definem a região onde a pertinência é 1.

A fórmula é:

$$\mu(x) = \begin{cases} \frac{x-a}{b-a}, & \text{se } a \leq x < b, \\ 1, & \text{se } b \leq x \leq c, \\ \frac{d-x}{d-c}, & \text{se } c < x \leq d, \\ 0, & \text{caso contrário.} \end{cases}$$

O código Python correspondente é:

```
def trapezoidal(x, a, b, c, d):
    if a <= x < b:
        return (x - a) / (b - a)
    elif b <= x <= c:
        return 1
    elif c < x <= d:
        return (d - x) / (d - c)
    else:
        return 0

# Exemplo de plotagem para a função trapezoidal
a, b, c, d = 5, 10, 15, 20  # Parâmetros da função trapezoidal
x = np.linspace(0, 25, 100)  # Valores de x no intervalo [0, 25]

# Calcula os graus de pertinência
y = [trapezoidal(val, a, b, c, d) for val in x]

# Plotando a função trapezoidal
plot_results(x, y, [a, b, c, d], "trapezoidal")
```

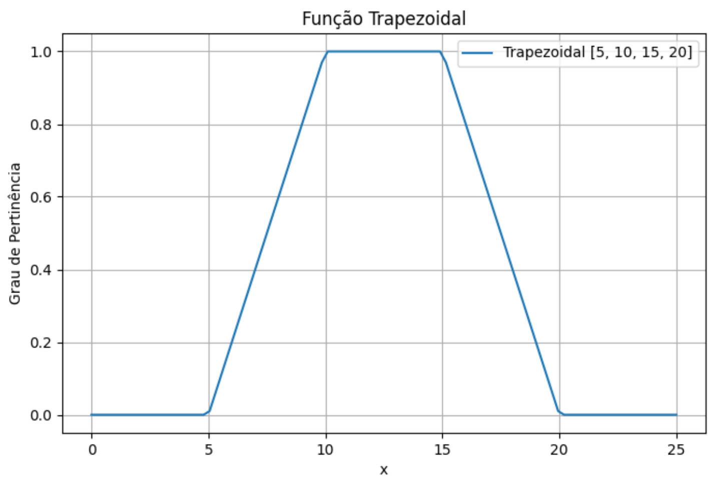

Figure 2: Exemplo de função trapezoidal com $a = 5$, $b = 10$, $c = 15$, $d = 20$.

2.1.3 Função Gaussiana

A função gaussiana é definida por dois parâmetros (c, σ), onde:

- c é o centro da curva, onde a pertinência é máxima (1);
- σ controla a largura da curva.

A fórmula é:

$$\mu(x) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}.$$

O código Python correspondente é:

```
def gaussian(x, c, sigma):
    return np.exp(-0.5 * ((x - c) / sigma) ** 2)


# Exemplo de uso
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]
c = 10  # Centro da curva (onde a pertinência é máxima)
sigma = 3  # Largura da curva

# Calcula os graus de pertinência
y = [gaussian(val, c, sigma) for val in x]

# Plotando a função gaussiana
plot_results(x, y, [c, sigma], "gaussian")
```

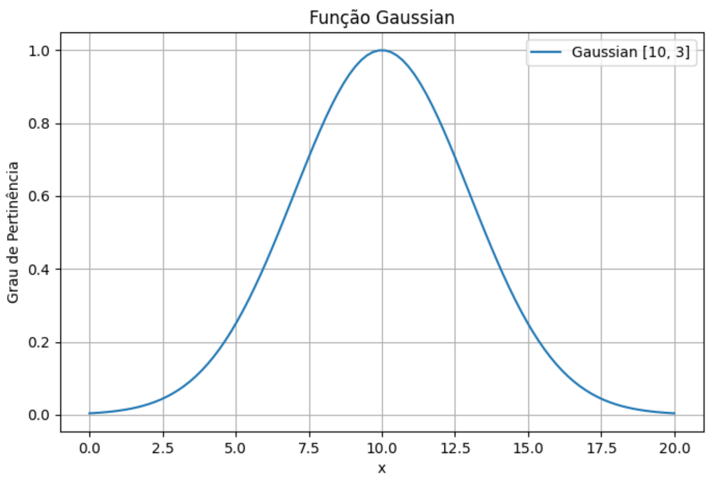

Figure 3: Exemplo de função gaussiana com $c = 10$, $\sigma = 3$.

2.1.4 Função Sigmoidal

A função sigmoidal é definida por dois parâmetros (a, c), onde:

- a controla a inclinação da curva;
- c é o ponto central onde a pertinência é 0.5.

A fórmula é:

$$\mu(x) = \frac{1}{1 + e^{-a(x-c)}}.$$

O código Python correspondente é:

```
def sigmoidal(x, a, c):
    return 1 / (1 + np.exp(-a * (x - c)))

# Exemplo de plotagem para a função sigmoidal
a, c = 1, 10  # Parâmetros da função sigmoidal
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]

# Calcula os graus de pertinência
y = [sigmoidal(val, a, c) for val in x]

# Plotando a função sigmoidal
plot_results(x, y, [a, c], "sigmoidal")
```

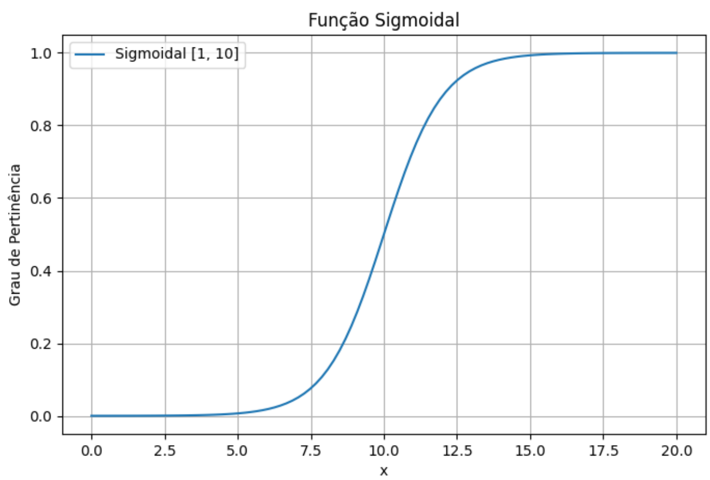

Figure 4: Exemplo de função sigmoidal com $a = 1$, $c = 10$.

2.1.5 Função Sinoidal (Bell)

A função Bell é definida por três parâmetros (a, b, c), onde:

- a controla a largura da curva;
- b controla a inclinação;
- c é o centro da curva.

A fórmula é:

$$\mu(x) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}.$$

O código Python correspondente é:

```
def bell_function(x, a, b, c):
    return 1 / (1 + abs((x - c) / a) ** (2 * b))
# Exemplo de plotagem para a função Bell
a, b, c = 2, 4, 10  # Parâmetros da função Bell
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]

# Calcula os graus de pertinência
y = [bell_function(val, a, b, c) for val in x]

# Plotando a função Bell
plot_results(x, y, [a, b, c], "bell")
```

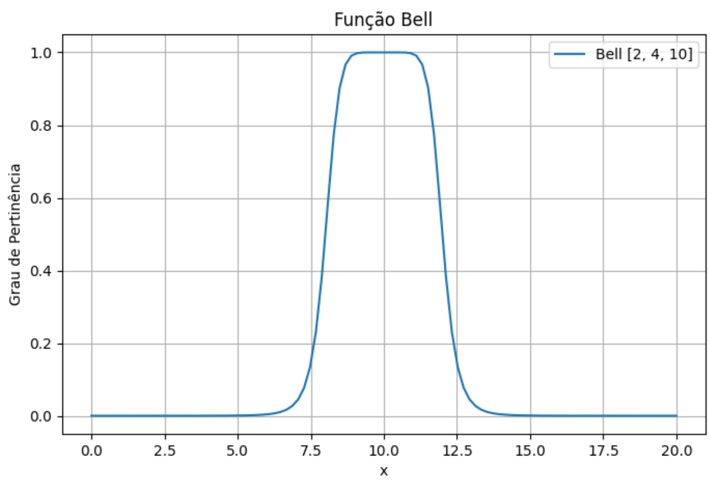

Figure 5: Exemplo de função Bell com $a = 2$, $b = 4$, $c = 10$.

2.1.6 Função S

A função S é definida por dois parâmetros (a, b), onde:

- a é o ponto onde a pertinência começa a aumentar;
- b é o ponto onde a pertinência atinge 1.

A fórmula é:

$$\mu(x) = \begin{cases} 0, & \text{se } x \leq a, \\ 2\left(\frac{x-a}{b-a}\right)^2, & \text{se } a < x < b, \\ 1, & \text{se } x \geq b. \end{cases}$$

O código Python correspondente é:

```
def s_function(x, a, b):
    if x <= a:
        return 0
    elif a < x < b:
        return 2 * ((x - a) / (b - a)) ** 2
    elif x >= b:
        return 1

# Exemplo de plotagem para a função S
a, b = 5, 15  # Parâmetros da função S
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]

# Calcula os graus de pertinência
y = [s_function(val, a, b) for val in x]

# Plotando a função S
plot_results(x, y, [a, b], "s")
```

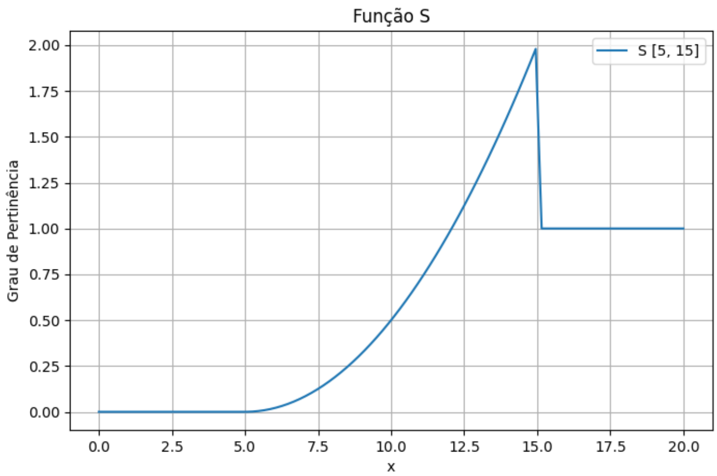

Figure 6: Exemplo de função S com $a = 5$, $b = 15$.

2.1.7 Função Z

A função Z é definida por dois parâmetros (a, b), onde:

- a é o ponto onde a pertinência começa a diminuir;
- b é o ponto onde a pertinência atinge 0.

A fórmula é:

$$\mu(x) = \begin{cases} 1, & \text{se } x \leq a, \\ 1 - 2\left(\frac{x-a}{b-a}\right)^2, & \text{se } a < x < b, \\ 0, & \text{se } x \geq b. \end{cases}$$

O código Python correspondente é:

```
def z_function(x, a, b):
    if x <= a:
        return 1
    elif a < x < b:
        return 1 - 2 * ((x - a) / (b - a)) ** 2
    else:
        return 0

# Exemplo de plotagem para a função Z
a, b = 5, 15  # Parâmetros da função Z
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]
```

```
# Calcula os graus de pertinência
y = [z_function(val, a, b) for val in x]

# Plotando a função Z
plot_results(x, y, [a, b], "z")
```

2.1.8 Função Cauchy

A função Cauchy é definida por dois parâmetros (c, γ), onde:

- c é o centro da curva;
- γ controla a largura da curva.

A fórmula é:

$$\mu(x) = \frac{1}{1 + \left(\frac{x-c}{\gamma}\right)^2}.$$

O código Python correspondente é:

```
def cauchy_function(x, c, gamma):
    return 1 / (1 + ((x - c) / gamma) ** 2)

# Exemplo de plotagem para a função Cauchy
c, gamma = 10, 3  # Parâmetros da função Cauchy
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]

# Calcula os graus de pertinência
y = [cauchy_function(val, c, gamma) for val in x]

# Plotando a função Cauchy
plot_results(x, y, [c, gamma], "cauchy")
```

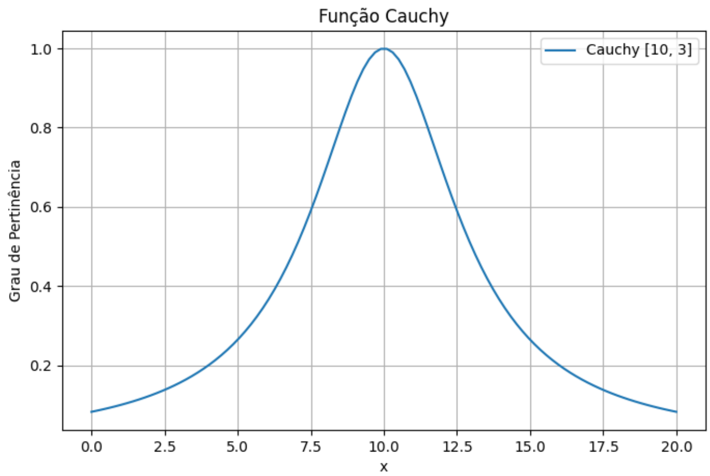

Figure 7: Exemplo de função Cauchy com $c = 10$, $\gamma = 3$.

2.1.9 Função Gaussiana Dupla

A função Gaussiana Dupla é definida por três parâmetros (c, σ_1, σ_2), onde:

- c é o centro da curva;
- σ_1 controla a largura da curva para $x \leq c$;
- σ_2 controla a largura da curva para $x > c$.

A fórmula é:

$$\mu(x) = \begin{cases} e^{-\frac{1}{2}\left(\frac{x-c}{\sigma_1}\right)^2}, & \text{se } x \leq c, \\ e^{-\frac{1}{2}\left(\frac{x-c}{\sigma_2}\right)^2}, & \text{se } x > c. \end{cases}$$

O código Python correspondente é:

```
def double_gaussian(x, c, sigma1, sigma2):
    if x <= c:
        return np.exp(-0.5 * ((x - c) / sigma1) ** 2)
    else:
        return np.exp(-0.5 * ((x - c) / sigma2) ** 2)

# Exemplo de plotagem para a função Gaussiana Dupla
c1, sigma1 = 8, 2  # Parâmetros da primeira gaussiana
c2, sigma2 = 14, 3  # Parâmetros da segunda gaussiana
x = np.linspace(0, 20, 100)  # Valores de x no intervalo [0, 20]
```

```
# Calcula os graus de pertinência
y = [double_gaussian(val, c1, sigma1, c2, sigma2) for val in x]

# Plotando a função Gaussiana Dupla
plot_results(x, y, [c1, sigma1, c2, sigma2], "double_gaussian")
```

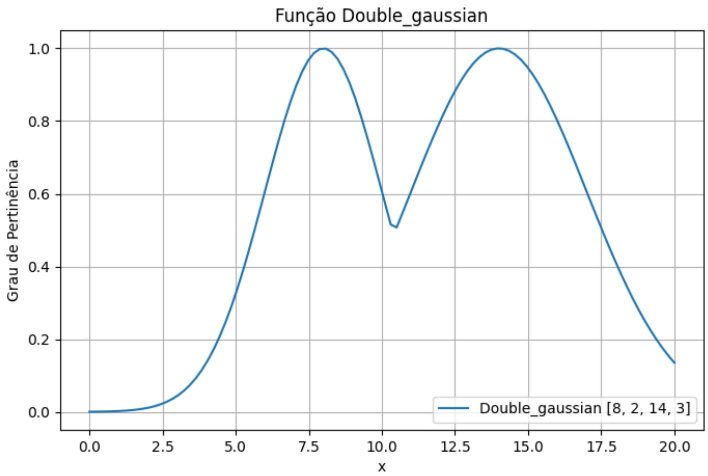

Figure 8: Exemplo de função Gaussiana Dupla com $c = 10$, $\sigma_1 = 3$, $\sigma_2 = 5$.

2.1.10 Função Retangular

A função Retangular é definida por dois parâmetros (a, b), onde:

- a é o início do intervalo onde a pertinência é 1;
- b é o final do intervalo onde a pertinência é 1.

A fórmula é:

$$\mu(x) = \begin{cases} 1, & \text{se } a \leq x \leq b, \\ 0, & \text{caso contrário.} \end{cases}$$

O código Python correspondente é:

```
def rectangular(x, a, b):
    if a <= x <= b:
        return 1
    else:
```

```
        return 0

# Exemplo de plotagem para a função Retangular
a, b = 3, 7  # Parâmetros da função Retangular
x = np.linspace(0, 10, 100)  # Valores de x no intervalo [0, 10]

# Calcula os graus de pertinência
y = [rectangular_function(val, a, b) for val in x]

# Plotando a função Retangular
plot_results(x, y, [a, b], "rectangular")
```

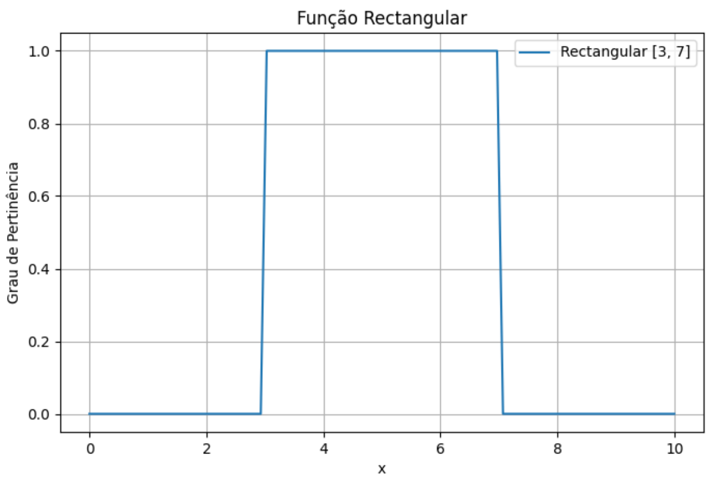

Figure 9: Exemplo de função Retangular com $a = 10$, $b = 20$.

2.1.11 Função Logarítmica

A função Logarítmica é definida por dois parâmetros (a, b), onde:

- a controla o deslocamento da curva;
- b controla a inclinação da curva.

A fórmula é:

$$\mu(x) = \begin{cases} 0, & \text{se } x \leq a, \\ \log_b(x - a + 1), & \text{se } x > a. \end{cases}$$

O código Python correspondente é:

```
import math

def logarithmic_function(x, a, b):
    if x <= a:
        return 0
    else:
        return math.log(x - a + 1, b)

# Exemplo de plotagem para a função Logarítmica
a, b = 2, 1  # Parâmetros da função Logarítmica
x = np.linspace(0.1, 10, 100)  # Valores de x no intervalo [0.1, 10] (evitando z

# Calcula os graus de pertinência
y = [logarithmic_function(val, a, b) for val in x]

# Plotando a função Logarítmica
plot_results(x, y, [a, b], "logarithmic")
```

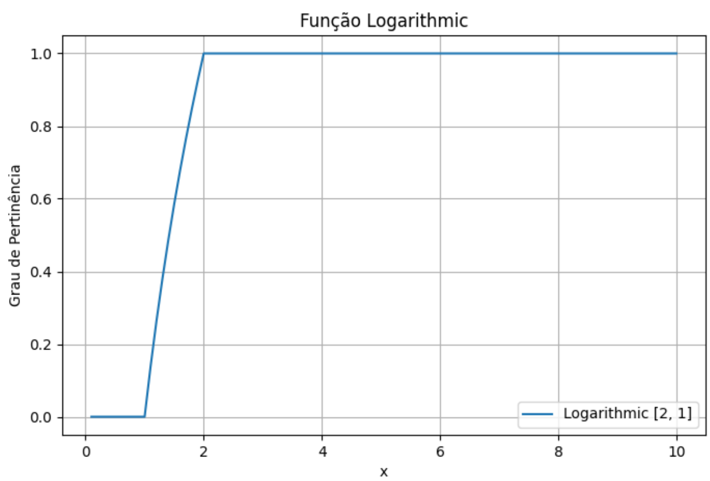

Figure 10: Exemplo de função Logarítmica com $a = 5$, $b = 2$.

2.2 Fuzzificação e Análise Comparativa

Para a fuzzificação, escolhemos uma variável de entrada com universo de discurso definido e particionamos o domínio em funções de pertinência uniformemente espaçadas. A seguir, apresentamos os resultados para duas amostras distintas.

Para as variáveis de entrada definir um universo de estudo como segue, particionado esse domínio em quatro funções de pertinência uniformemente espaçadas.

- **Frio**: Representa temperaturas baixas, variando de 0 a aproximadamente 25 graus.
- **Morno**: Temperatura intermediária, abrangendo valores entre 20 e 50 graus.
- **Quente**: Engloba valores de temperatura entre 45 e 75 graus.
- **Muito Quente**: Abrange temperaturas elevadas, variando de 70 a 100 graus.

Para realizar o espaçamento fizemos o algoritmo que se segue onde retorna os parametros:

```
def generate_params(X, types, n):
    centers = np.linspace(X[0], X[1], n)  # Centros uniformemente distribuídos
    step = (X[1] - X[0]) / (n - 1) if n > 1 else (X[1] - X[0])  # Espaçamento en
    params = []

    for i, t in enumerate(types):
        if t == 'gaussian':
            sigma = step / 2  # Sigma proporcional ao espaçamento
            params.append([centers[i], sigma])
        elif t == 'triangular':
            a = max(X[0], centers[i] - step)  # Início da base
            b = centers[i]                    # Pico
            c = min(X[1], centers[i] + step)  # Fim da base
            params.append([a, b, c])
        elif t == 'trapezoidal':
            a = max(X[0], centers[i] - step)  # Início da base
            b = max(X[0], centers[i] - step / 2)  # Início do topo
            c = min(X[1], centers[i] + step / 2)  # Fim do topo
            d = min(X[1], centers[i] + step)  # Fim da base
            params.append([a, b, c, d])
        elif t == 'sigmoidal':
            a = 1  # Inclinação padrão
            c = centers[i]  # Centro
            params.append([a, c])
```

```
        elif t == 'bell':
            a = step / 2  # Largura do sino
            b = 2  # Inclinação padrão
            c = centers[i]  # Centro
            params.append([a, b, c])
        elif t == 'z':
            a = max(X[0], centers[i] - step)  # Início do decaimento
            b = centers[i]  # Fim do decaimento
            params.append([a, b])
        elif t == 's':
            a = centers[i]  # Início do crescimento
            b = min(X[1], centers[i] + step)  # Fim do crescimento
            params.append([a, b])

        elif t == 'cauchy':
            c = centers[i]  # Centro
            gamma = step / 2  # Largura
            params.append([c, gamma])
        elif t == 'double_gaussian':
            c1 = max(X[0], centers[i] - step / 2)  # Centro da primeira gaussian
            sigma1 = step / 4  # Largura da primeira gaussiana
            c2 = min(X[1], centers[i] + step / 2)  # Centro da segunda gaussiana
            sigma2 = step / 4  # Largura da segunda gaussiana
            params.append([c1, sigma1, c2, sigma2])
        elif t == 'retangular':
            a = max(X[0], centers[i] - step / 2)  # Início do intervalo
            b = min(X[1], centers[i] + step / 2)  # Fim do intervalo
            params.append([a, b])
        elif t == 'linear':  # Adicionando a função linear
            a = max(X[0], centers[i] - step)  # Início da base
            b = min(X[1], centers[i] + step)  # Fim da base
            params.append([a, b])
        else:
            raise ValueError(f"Tipo de função de pertinência'{t}' não suportado!

    return params
```

Outra função e para calcular o graus de pertinência para cada atributo, nesse algortmo esolhemos as funções de pertinência relacionadas a seção 2.

```
### Função Membership
```

```
def calculate_membership(dominio, types, params):
    x = np.linspace(dominio[0], dominio[1], 100)  # Gera os valores de x no domí
    results = []
    for func_type, func_params in zip(types, params):

        # Calcula os graus de pertinência com base no tipo de função
        if func_type == 'linear':
            results.append([linear_function(val, *func_params) for val in x])
        elif func_type == 'triangular':
            results.append([triangular(val, *func_params) for val in x])
        elif func_type == 'trapezoidal':
            results.append([trapezoidal(val, *func_params) for val in x])
        elif func_type == 'gaussian':
            results.append([gaussian(val, *func_params) for val in x])
        elif func_type == 'sigmoidal':
            results.append([sigmoidal(val, *func_params) for val in x])
        elif func_type == 'z':
            results.append([z_function(val, *func_params) for val in x])
        elif func_type == 's':
            results.append([s_function(val, *func_params) for val in x])
        elif func_type == 'pi':
            results.append([pi_function(val, *func_params) for val in x])
        elif func_type == 'bell':
            results.append([bell_function(val, *func_params) for val in x])
        elif func_type == 'singleton':
            results.append([singleton_function(val, *func_params) for val in x])
        elif func_type == 'cauchy':
            results.append([cauchy_function(val, *func_params) for val in x])
        elif func_type == 'double_gaussian':
            results.append([double_gaussian(val, *func_params) for val in x])
        elif func_type == 'retangular':
            results.append([rectangular_function(val, *func_params) for val in x
        elif func_type == 'logaritmica':
            results.append([logarithmic_function(val, *func_params) for val in x
        else:
            raise ValueError(f"Tipo de função desconhecido: {func_type}")

    return x, results
```

Para a lotagem dos graficos usamos a função

```
def plot_membership_with_samples(x, results, labels, samples, title, activations
```

```
plt.figure(figsize=(10, 6))
for j, sample in enumerate(samples):
        # Adiciona uma linha vertical para cada amostra
        random_color = np.random.rand(3,)
        plt.axvline(sample, color=random_color, linestyle='--', alpha=0.7, l
for i, result in enumerate(results):
    # Plota a função de pertinência com o rótulo correspondente
    plt.plot(x, result, label=f"{labels[i]}")
    for j, sample in enumerate(samples):

        # Pega o grau de ativação correspondente
        activation = activations[i][j]
        # Adiciona um ponto no gráfico para destacar a ativação
        random_color = np.random.rand(3,)
        plt.scatter(sample, activation, color=random_color,
                    label=f"Sample {sample}: {activation:.2f}")

# Configurações do gráfico
plt.title(title)
plt.xlabel("x")
plt.ylabel("Grau de Pertinência")
plt.legend(loc="upper right", bbox_to_anchor=(1.3, 1), title="Funções Fuzzy"
plt.grid()
plt.tight_layout()
plt.savefig(f'{output_dir}/{title.replace("- ", "").lower().replace(" ", "_"
plt.show()
```

Por fim construir a função fuzificação

```
def fuzzificacao(n, type, dominio, samples, labels, liguistica):
# Número de funções de pertinência para cada atributo
types = [type] * n

# Geração dos parâmetros para cada tipo de função
params = generate_params(dominio, types, n)

for i in range(n):
    v = dominio[1] - dominio[0]  # Tamanho do intervalo
    p = v / n  # Espaçamento correto entre os pontos
    intervalo = [round(dominio[0] + p * i, 2), round(dominio[0] + p * (i + 1
    parametros_formatados = [float(round(val, 2)) for val in params[i]]
```

```
    print(f'{labels[i]}: {liguistica} - {intervalo} - Parâmetros: {parametro
# Cálculo dos graus de pertinência para cada atributo
x, results = calculate_membership(dominio, types, params)
# Cálculo do grau de ativação para cada amostra
activations = []
for i, result in enumerate(results):
    sample_activations = []
    for sample in samples:
        # Encontra o índice mais próximo do valor da amostra no domínio
        idx = np.abs(x - sample).argmin()
        activation = result[idx]
        sample_activations.append(activation)
    activations.append(sample_activations)

# Imprime as ativações calculadas com melhor formatação
print(f"\nAtivações para o tipo {type}:")
print(f"Samples: {samples}")
for i, sample_activations in enumerate(activations):
     # Arredonda os valores para duas casas decimais
    rounded_activations = np.round(sample_activations, 2)
    print(f"{labels[i]}: {rounded_activations.tolist()}")

# Plotagem das funções de pertinência para o atributo com as amostras
plot_membership_with_samples(
    x, results, labels, samples, f"Funções de Pertinência - {type}", activat
)
```

com todas as função feitas definir o exemplo:

```
# Definição do universo de discurso
dominio = (0, 100)  # Intervalo do universo de discurso
samples = [25, 75]  # Amostras para fuzzificação
n = 5
linguistica = 'temperatura'
funcao_pertinecia = [
    'triangular', 'trapezoidal', 'gaussian', 'sigmoidal', 'bell', 's', 'z',
]

labels = ['Muito Firo', 'Frio', 'Morno', 'Quente', 'Muito quente']
```

```
for type in funcao_pertinecia:

    fuzzificacao(n, type  , dominio, samples, [labels[i] for i in range(n)],
```

dominio = (0, 100) samples = [25, 75]

2.2.1 Função Triangular

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 0.0, 25.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [0.0, 25.0, 50.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [25.0, 50.0, 75.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [50.0, 75.0, 100.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [75.0, 100.0, 100.0]

Ativações para o tipo triangular: Samples: [25, 75] Muito Firo: [0, 0] Frio: [0.99, 0.0] Morno: [0.01, 0.01] Quente: [0.0, 0.99] Muito quente: [0, 0]

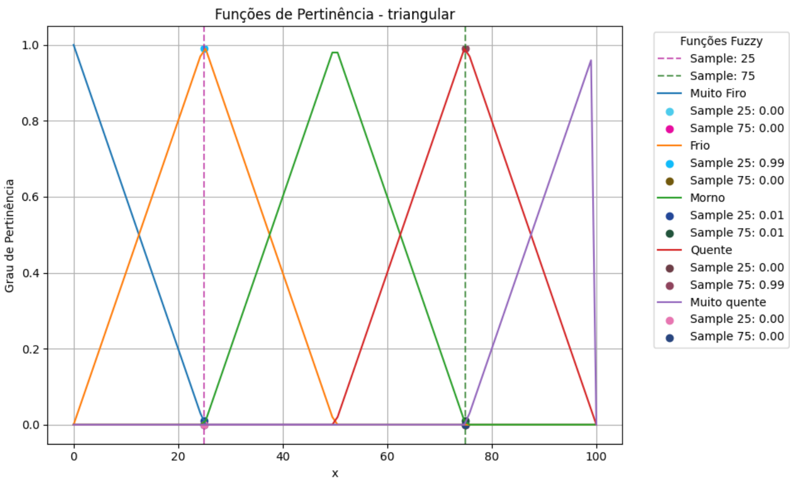

Figure 11: Funções de pertinência triangulares e graus de ativação.

2.2.2 Função Trapezoidal

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 0.0, 12.5, 25.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [0.0, 12.5, 37.5, 50.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [25.0, 37.5, 62.5, 75.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [50.0, 62.5, 87.5, 100.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [75.0, 87.5, 100.0, 100.0]

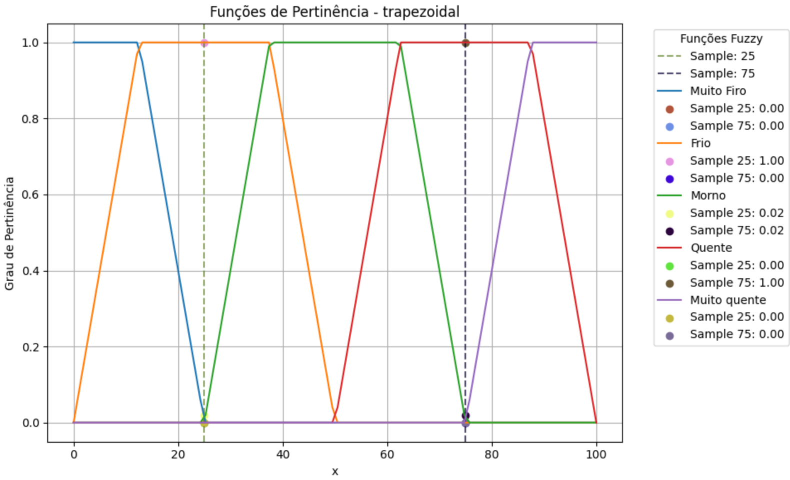

Figure 12: Funções de pertinência trapezoidais e graus de ativação.

Ativações para o tipo trapezoidal: Samples: [25, 75] Muito Firo: [0, 0] Frio: [1, 0] Morno: [0.02, 0.02] Quente: [0, 1] Muito quente: [0, 0]

2.2.3 Função Gaussiana

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 12.5] Frio: temperatura - [20.0, 40.0] - Parâmetros: [25.0, 12.5] Morno: temperatura - [40.0, 60.0] - Parâmetros: [50.0, 12.5] Quente: temperatura - [60.0, 80.0] - Parâmetros: [75.0, 12.5] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [100.0, 12.5]

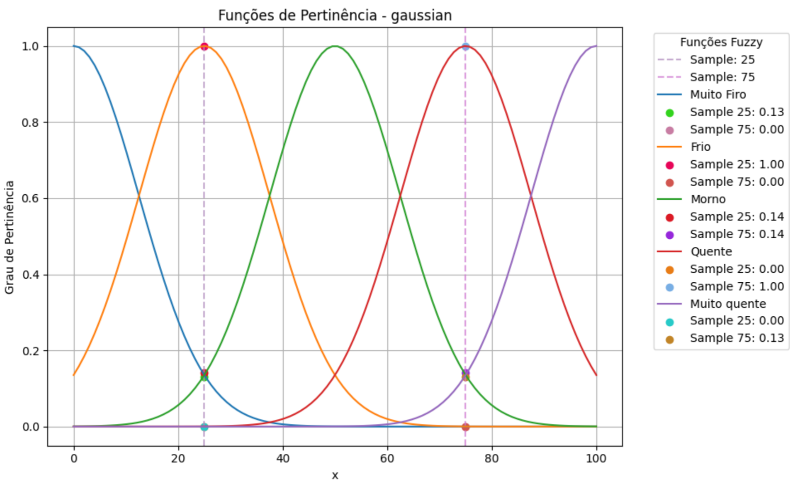

Figure 13: Funções de pertinência gaussianas e graus de ativação.

Samples: [25, 75] Muito Firo: [0.13, 0.0] Frio: [1.0, 0.0] Morno: [0.14, 0.14] Quente: [0.0, 1.0] Muito quente: [0.0, 0.13]

2.2.4 Função Sigmoidal

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [1.0, 0.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [1.0, 25.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [1.0, 50.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [1.0, 75.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [1.0, 100.0]

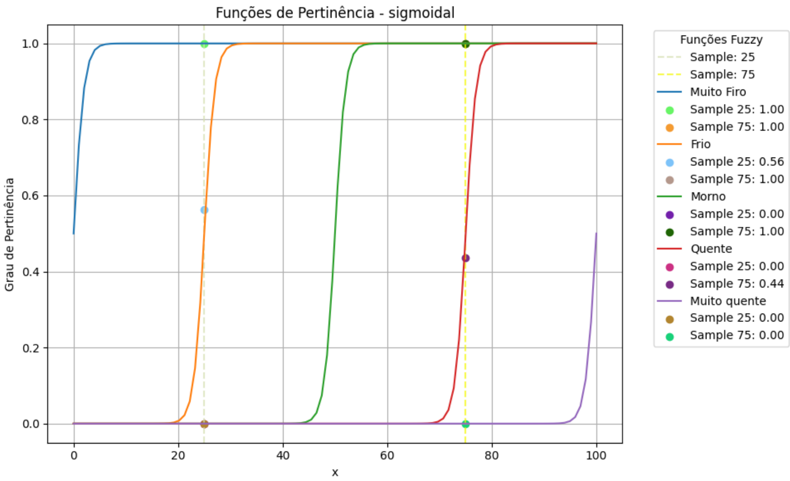

Figure 14: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo sigmoidal: Samples: [25, 75] Muito Firo: [1.0, 1.0] Frio: [0.56, 1.0] Morno: [0.0, 1.0] Quente: [0.0, 0.44] Muito quente: [0.0, 0.0]

2.2.5 Função Bell

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [12.5, 2.0, 0.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [12.5, 2.0, 25.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [12.5, 2.0, 50.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [12.5, 2.0, 75.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [12.5, 2.0, 100.0]

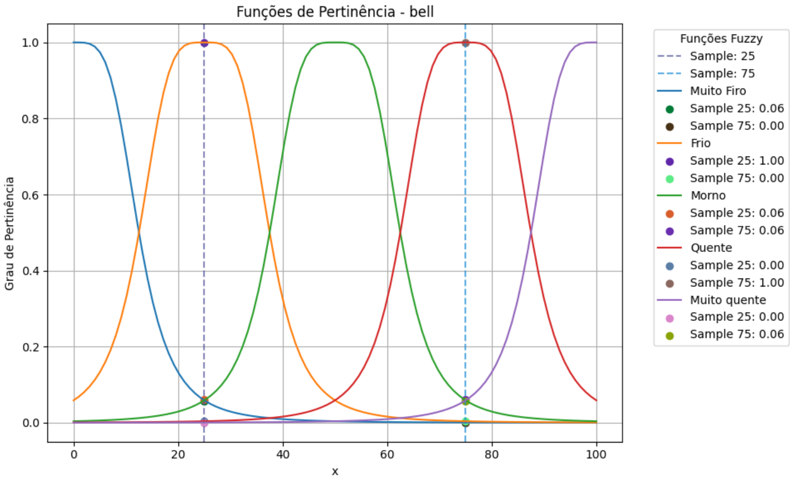

Figure 15: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo bell: Samples: [25, 75] Muito Firo: [0.06, 0.0] Frio: [1.0, 0.0] Morno: [0.06, 0.06] Quente: [0.0, 1.0] Muito quente: [0.0, 0.06]

2.2.6 Função S

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 25.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [25.0, 50.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [50.0, 75.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [75.0, 100.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [100.0, 100.0]

Quente: [0, 0] Muito quente: [0, 0]

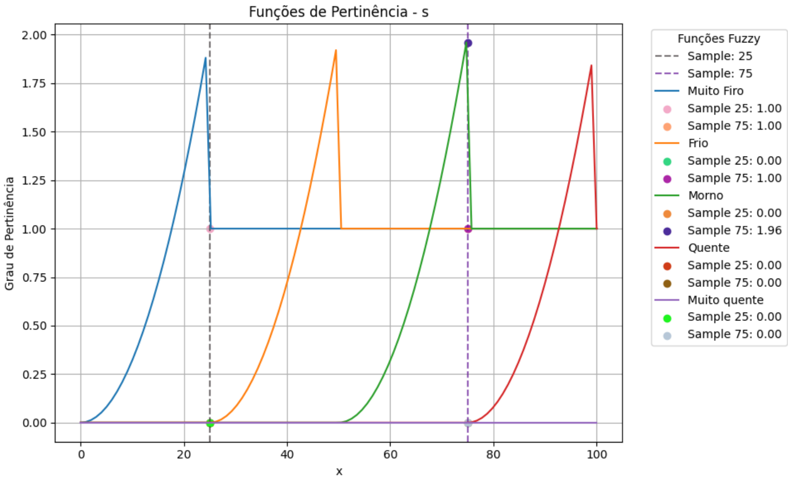

Figure 16: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo s: Samples: [25, 75] Muito Firo: [1, 1] Frio: [0.0, 1.0] Morno: [0.0, 1.96]

2.2.7 Função S

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 25.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [25.0, 50.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [50.0, 75.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [75.0, 100.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [100.0, 100.0]

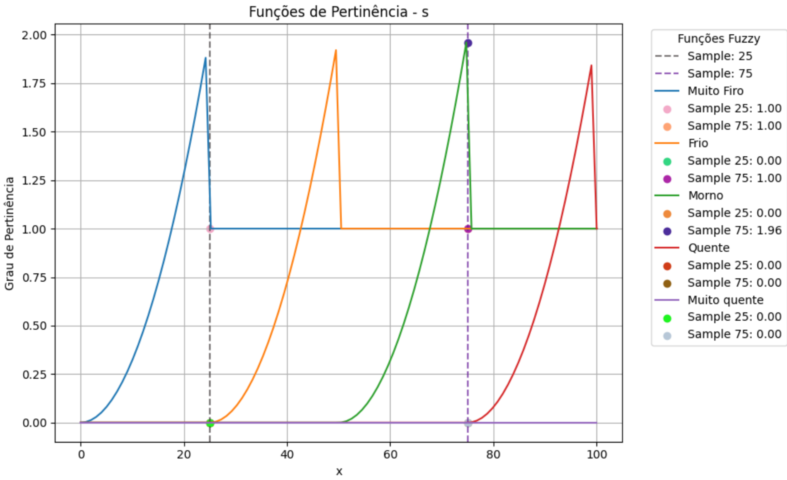

Figure 17: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo s: Samples: [25, 75] Muito Firo: [1, 1] Frio: [0.0, 1.0] Morno: [0.0, 1.96] Quente: [0, 0] Muito quente: [0, 0]

2.2.8 Função Z

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 0.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [0.0, 25.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [25.0, 50.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [50.0, 75.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [75.0, 100.0]

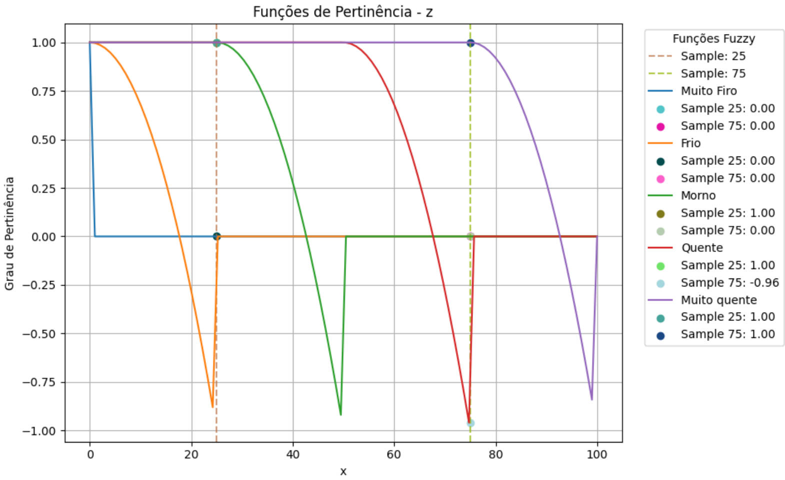

Figure 18: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo z: Samples: [25, 75] Muito Firo: [0, 0] Frio: [0, 0] Morno: [1.0, 0.0] Quente: [1.0, -0.96] Muito quente: [1, 1]

2.2.9 Função Cauchy

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 12.5] Frio: temperatura - [20.0, 40.0] - Parâmetros: [25.0, 12.5] Morno: temperatura - [40.0, 60.0] - Parâmetros: [50.0, 12.5] Quente: temperatura - [60.0, 80.0] - Parâmetros: [75.0, 12.5] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [100.0, 12.5]

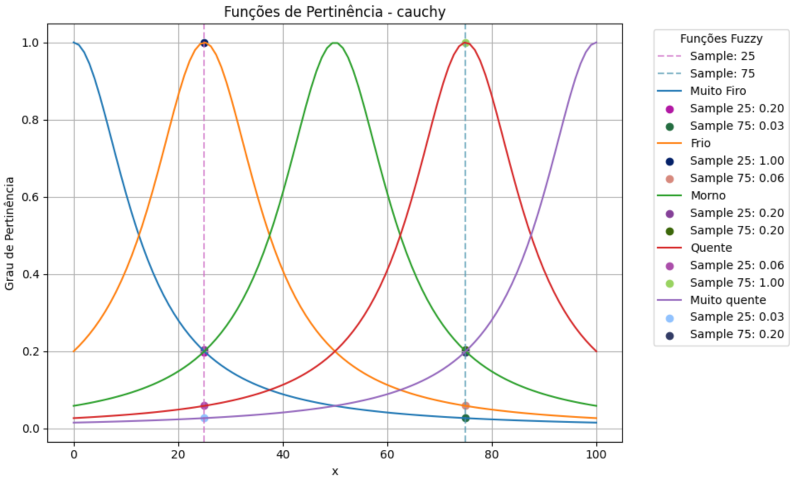

Figure 19: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo cauchy: Samples: [25, 75] Muito Firo: [0.2, 0.03] Frio: [1.0, 0.06] Morno: [0.2, 0.2] Quente: [0.06, 1.0] Muito quente: [0.03, 0.2]

2.2.10 Função Gaussiana Dupla

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 6.25, 12.5, 6.25] Frio: temperatura - [20.0, 40.0] - Parâmetros: [12.5, 6.25, 37.5, 6.25] Morno: temperatura - [40.0, 60.0] - Parâmetros: [37.5, 6.25, 62.5, 6.25] Quente: temperatura - [60.0, 80.0] - Parâmetros: [62.5, 6.25, 87.5, 6.25] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [87.5, 6.25, 100.0, 6.25]

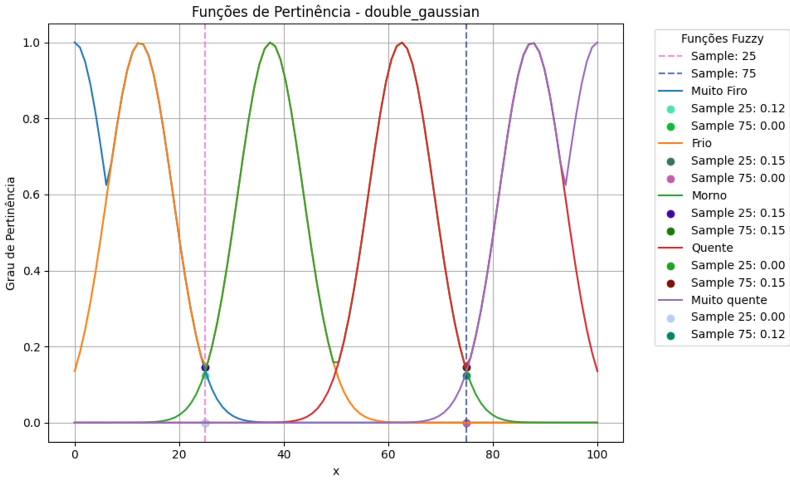

Figure 20: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo double gaussian: Samples: [25, 75] Muito Firo: [0.12, 0.0] Frio: [0.15, 0.0] Morno: [0.15, 0.15] Quente: [0.0, 0.15] Muito quente: [0.0, 0.12]

2.2.11 Função Retangular

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 12.5] Frio: temperatura - [20.0, 40.0] - Parâmetros: [12.5, 37.5] Morno: temperatura - [40.0, 60.0] - Parâmetros: [37.5, 62.5] Quente: temperatura - [60.0, 80.0] - Parâmetros: [62.5, 87.5] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [87.5, 100.0]

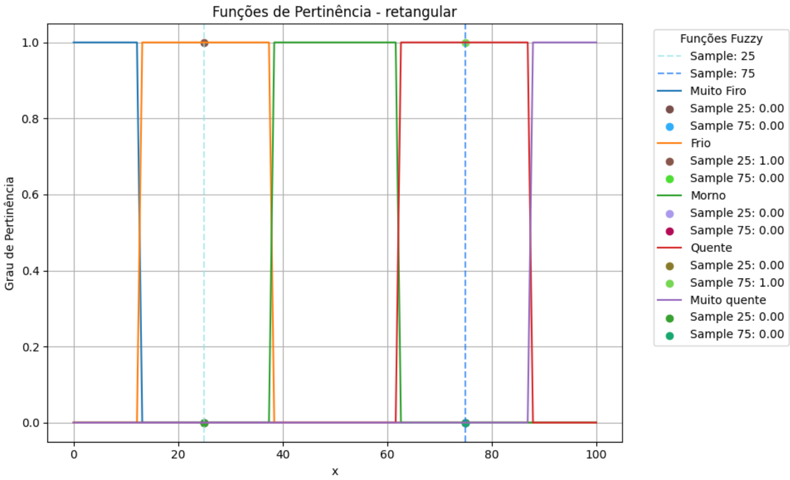

Figure 21: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo retangular: Samples: [25, 75] Muito Firo: [0, 0] Frio: [1, 0] Morno: [0, 0] Quente: [0, 1] Muito quente: [0, 0]

2.2.12 Função Linear

Muito Firo: temperatura - [0.0, 20.0] - Parâmetros: [0.0, 25.0] Frio: temperatura - [20.0, 40.0] - Parâmetros: [0.0, 50.0] Morno: temperatura - [40.0, 60.0] - Parâmetros: [25.0, 75.0] Quente: temperatura - [60.0, 80.0] - Parâmetros: [50.0, 100.0] Muito quente: temperatura - [80.0, 100.0] - Parâmetros: [75.0, 100.0]

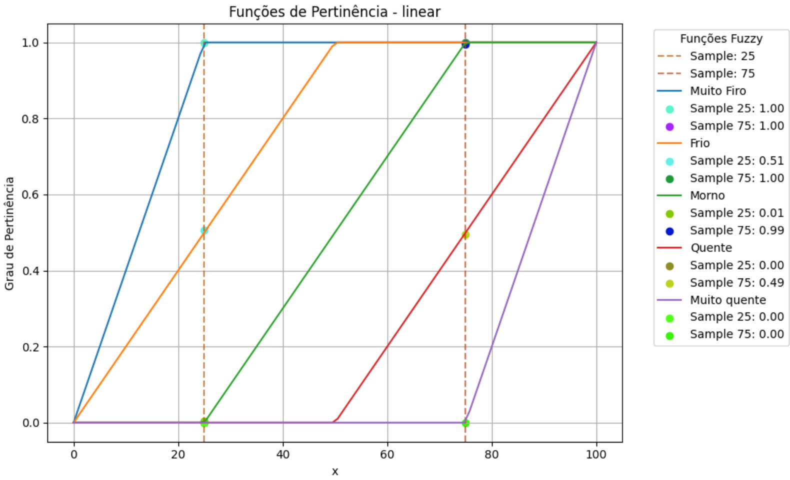

Figure 22: Funções de pertinência sigmoidais e graus de ativação.

Ativações para o tipo gaussian:

Ativações para o tipo linear: Samples: [25, 75] Muito Firo: [1, 1] Frio: [0.51, 1.0] Morno: [0.01, 0.99] Quente: [0.0, 0.49] Muito quente: [0, 0]

3 Operações Fuzzy

3.1 Complemento, União e Interseção

3.1.1 Complemento:

Zadeh

```
def complemento_zadeh(u):
return 1 - np.array(u)
```

Sugeno:

```
def complemento_sugeno(u, lamb=0.5):
return (1 - u) / (1 + lamb * u)
```

Yager:

```
def complemento_yager(u, w=2):
return (1 - u**w)**(1/w)
```

3.1.2 União (t-conormas):

Máximo

```
def complemento_yager(u, w=2):
return (1 - u**w)**(1/w)
```

Soma Probabilística

```
def uniao_soma_probabilistica(u1, u2):
    return u1 + u2 - u1 * u2
```

Soma Limitada

```
def uniao_soma_limitada(u1, u2):
    return np.minimum(1, u1 + u2)
```

Soma Drástica

```
def uniao_soma_drastica(u1, u2):
    return np.where((u1 == 0) & (u2 == 0), 0, np.maximum(u1, u2))
```

3.1.3 Interseção (t-conormas)

Mínimo

```
def intersecao_minimo(u1, u2):
    return np.minimum(u1, u2)
```

Produto

```
def intersecao_produto(u1, u2):
    return u1 * u2
```

Produto Limitado

```
def intersecao_produto_limitado(u1, u2):
    return np.maximum(0, u1 + u2 - 1)
```

Produto Drástico

```
def intersecao_produto_drastico(u1, u2):
    return np.where((u1 == 1) & (u2 == 1), np.minimum(u1, u2), 0)
```

3.2 Relaçõess Fuzzy

Este relatório apresentou a definição, particionamento e análise de funções de pertinência para a variável temperatura ambiente. A análise gráfica e textual destacou as diferenças entre os tipos de funções, evidenciando suas características e aplicações.