Exercício 09

Escreva um programa em MatLab para testar as seguintes funções de pertinência:

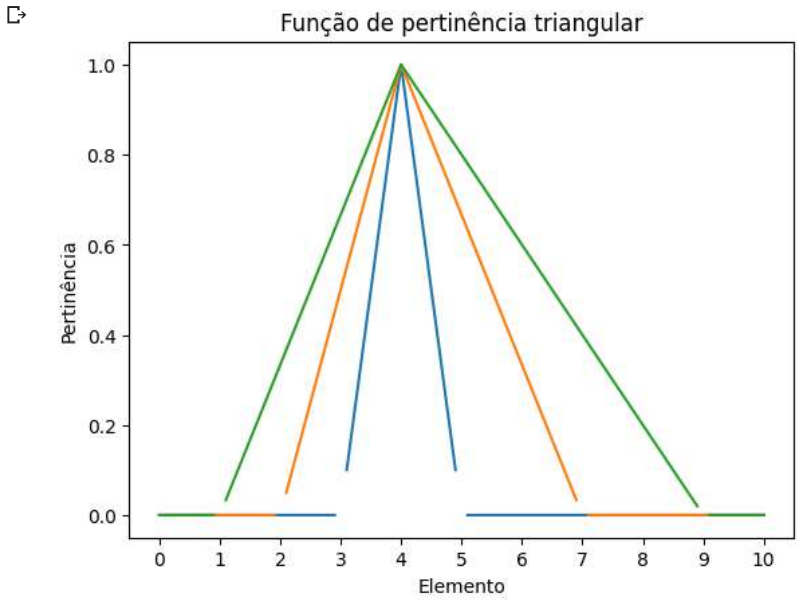
```
import math
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0.0, 10.1, 0.1).tolist()
```

- função de pertinência triangular: trimf(x, a, b, c), onde x é um vetor representando a variável de interesse; a, b e c representam os vértices (parâmetros) da função.

```
def trimf(x, a, b, c):
    def get_value(x, a, b, c):
        if x < a or c < x: return 0
        elif a < x <= b: return (x-a)/(b-a)
        elif b < x < c: return (c-x)/(c-b)
    if hasattr(x, '__iter__'):
        return [get_value(x_value, a, b, c) for x_value in x]
    else:
        return get_value(x, a, b, c)

y1 = trimf(x, 3, 4, 5)
y2 = trimf(x, 2, 4, 7)
y3 = trimf(x, 1, 4, 9)
plt.plot(x, y1, x, y2, x, y3)
plt.title('Função de pertinência triangular')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()
```

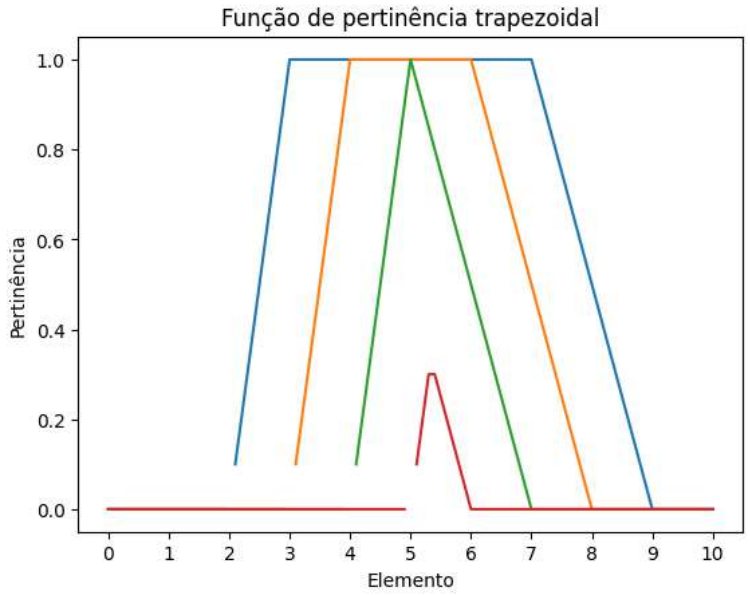


- função de pertinência trapezoidal: trapmf (x, a , b , c, d), onde x é um vetor representando a variável de interesse; a, b, c e d representam os vértices (parâmetros) da função.

```
def trapmf(x, a, b, c, d):
    def get_value(x, a, b, c, d):
        if x < a or d <= x: return 0
        if b > c:
            cross_x = (b*d - a*c)/(d - c + b - a)
            if a < x <= cross_x: return (x-a)/(b-a)
            elif cross_x < x < d: return (d-x)/(d-c)
        else:
            if a < x <= b: return (x-a)/(b-a)
            elif b < x <= c: return 1
            elif c < x < d: return (d-x)/(d-c)
```

```
    if hasattr(x, '__iter__'):
        return [get_value(x_value, a, b, c, d) for x_value in x]
    else:
        return get_value(x, a, b, c, d)

y1 = trapmf(x, 2, 3, 7, 9)
y2 = trapmf(x, 3, 4, 6, 8)
y3 = trapmf(x, 4, 5, 5, 7)
y4 = trapmf(x, 5, 6, 4, 6)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência trapezoidal')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()
```

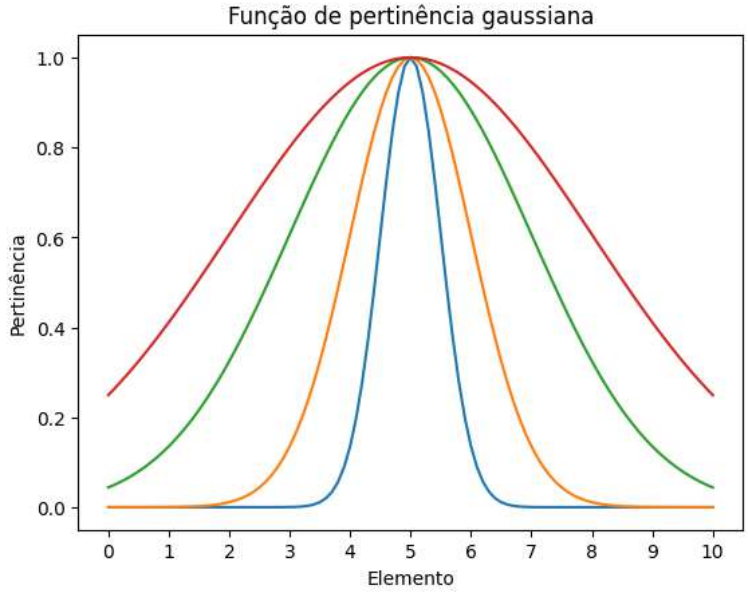


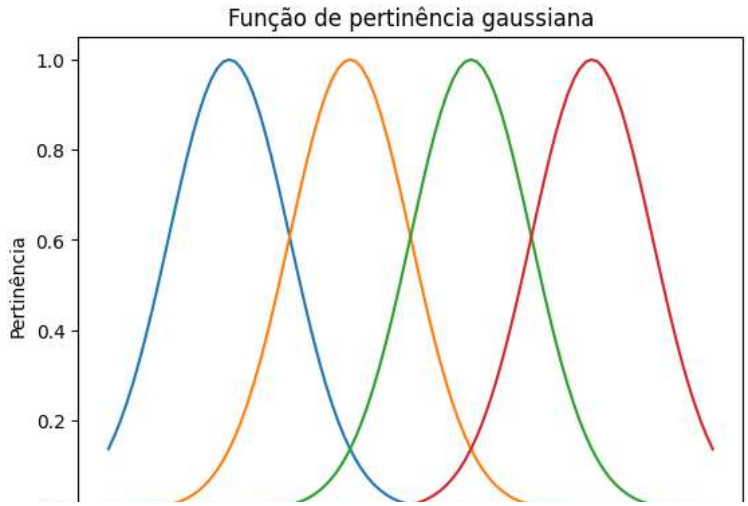
- função de pertinência gaussiana: gaussmf (x, c, sigma), onde x é um vetor representando a variável de interesse; c e sigma representam o centro e a largura (parâmetros) da função.

```
def gaussmf(x, c, sigma):
    def get_value(x, c, sigma):
        return math.exp(-0.5*((x-c)/sigma)**2)
    if hasattr(x, '__iter__'):
        return [get_value(x_value, c, sigma) for x_value in x]
    else:
        return get_value(x, c, sigma)

y1 = gaussmf(x, 5, 0.5)
y2 = gaussmf(x, 5, 1)
y3 = gaussmf(x, 5, 2)
y4 = gaussmf(x, 5, 3)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência gaussiana')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()

y1 = gaussmf(x, 2, 1)
y2 = gaussmf(x, 4, 1)
y3 = gaussmf(x, 6, 1)
y4 = gaussmf(x, 8, 1)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência gaussiana')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()
```



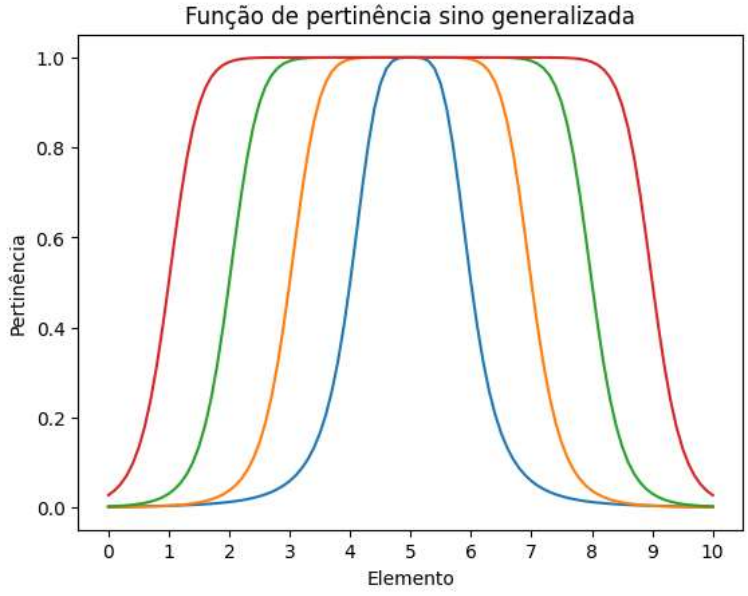


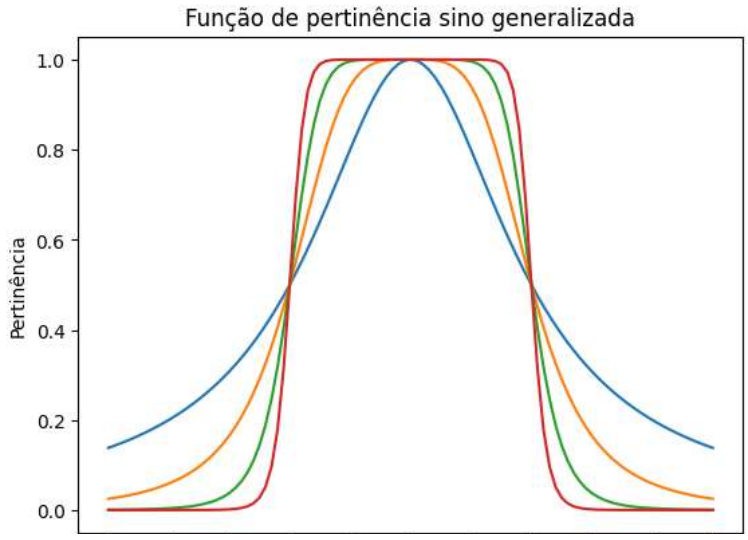
- função de pertinência Sino Generalizada: gbellmf (x, c, sigma, inc), onde x é um vetor representando a variável de interesse; c e sigma representam o centro e a largura (parâmetros) da função; inc é o parâmetro de inclinação nos crossover points.

```
def gbellmf(x, c, sigma, inc):
    def get_value(x, c, sigma, inc):
        return 1/(1 + abs((x-c)/sigma)**(2*inc))
    if hasattr(x, '__iter__'):
        return [get_value(x_value, c, sigma, inc) for x_value in x]
    else:
        return get_value(x, c, sigma, inc)

y1 = gbellmf(x, 5, 1, 2)
y2 = gbellmf(x, 5, 2, 4)
y3 = gbellmf(x, 5, 3, 6)
y4 = gbellmf(x, 5, 4, 8)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência sino generalizada')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()

y1 = gbellmf(x, 5, 2, 1)
y2 = gbellmf(x, 5, 2, 2)
y3 = gbellmf(x, 5, 2, 4)
y4 = gbellmf(x, 5, 2, 8)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência sino generalizada')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()
```



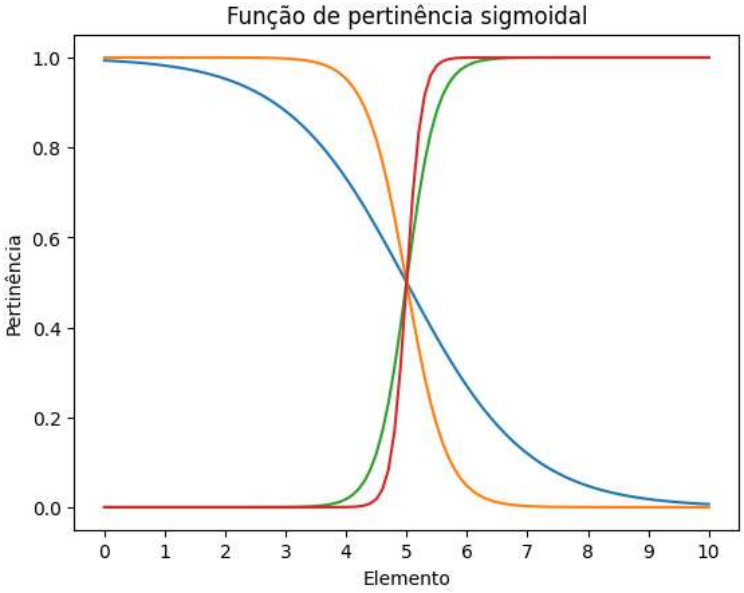


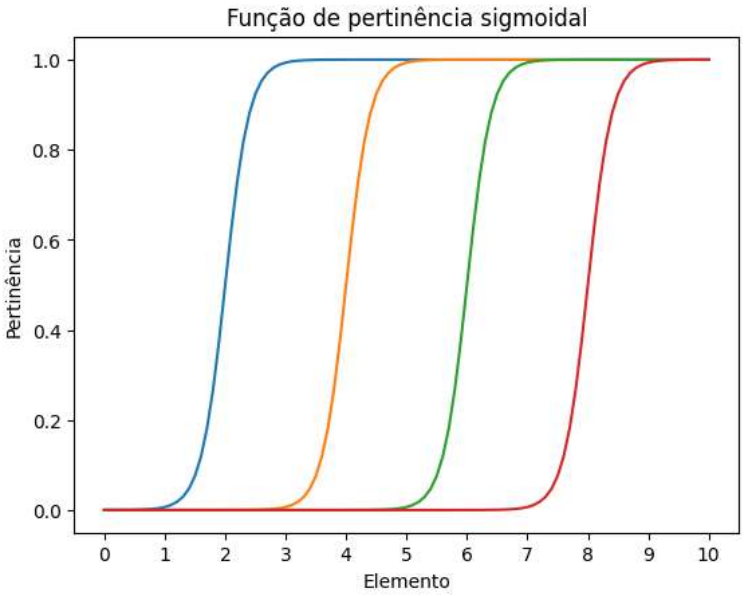
- função de pertinência sigmoidal: sigmf (x, a, c), onde x é um vetor representando a variável de interesse; a eontrola a inclinação da função no crossover point x = c.

```
def sigmf(x, a, c):
    def get_value(x, a, c):
        return 1/(1 + math.exp(-a*(x-c)))
    if hasattr(x, '__iter__'):
        return [get_value(x_value, a, c) for x_value in x]
    else:
        return get_value(x, a, c)

y1 = sigmf(x, -1, 5)
y2 = sigmf(x, -3, 5)
y3 = sigmf(x, 4, 5)
y4 = sigmf(x, 8, 5)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência sigmoidal')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()

y1 = sigmf(x, 5, 2)
y2 = sigmf(x, 5, 4)
y3 = sigmf(x, 5, 6)
y4 = sigmf(x, 5, 8)
plt.plot(x, y1, x, y2, x, y3, x, y4)
plt.title('Função de pertinência sigmoidal')
plt.xlabel('Elemento')
plt.ylabel('Pertinência')
plt.xticks(np.arange(0, 11))
plt.show()
```





✓ 0s conclusão: 19:46