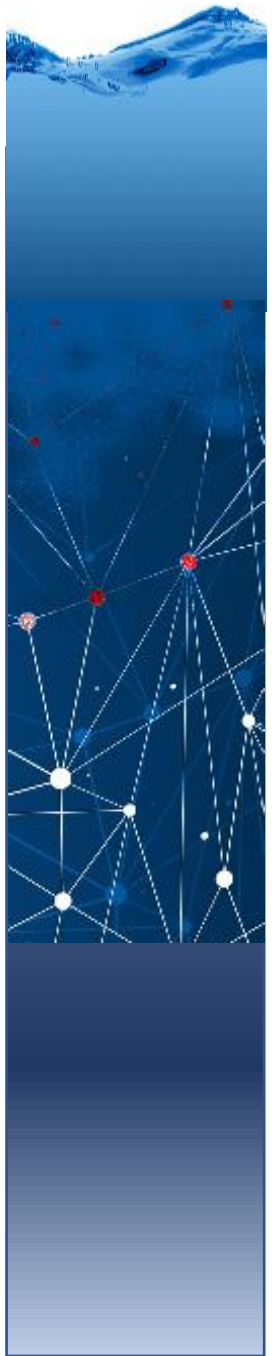
PPGERHA



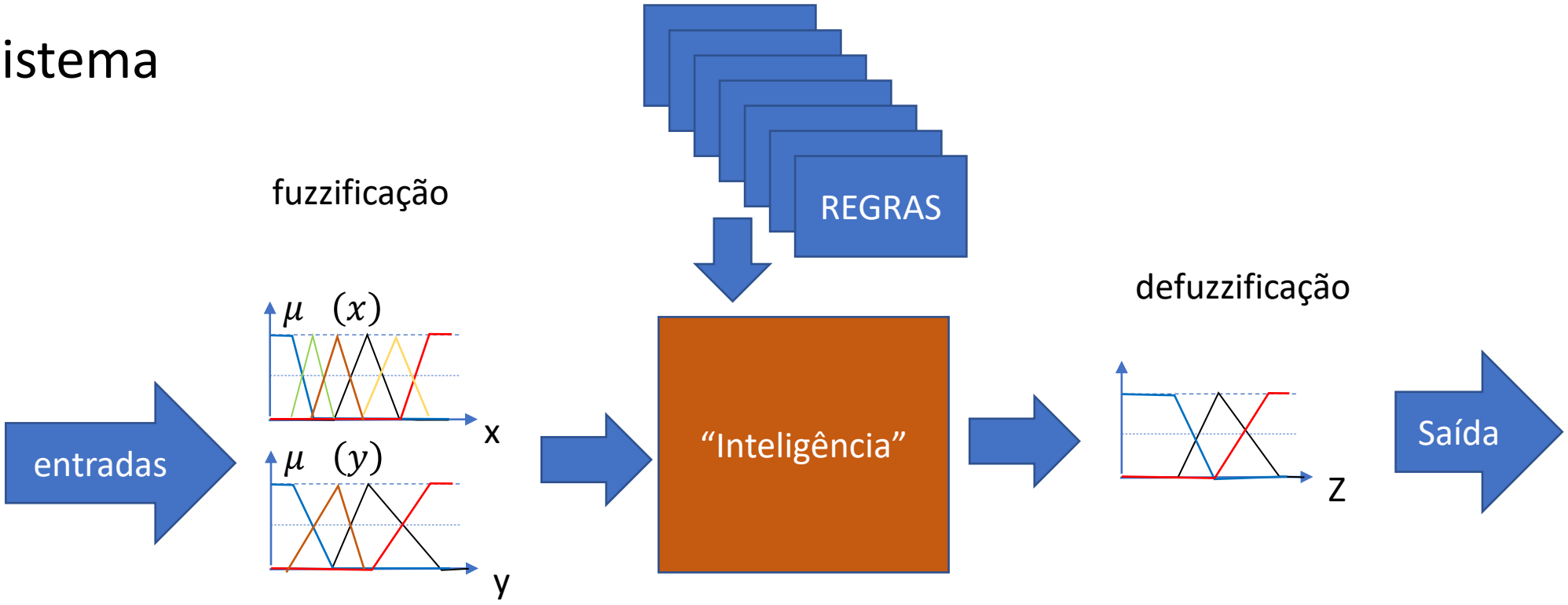
Lógica fuzzy

programa simples

Jorge Centeno

Sistema

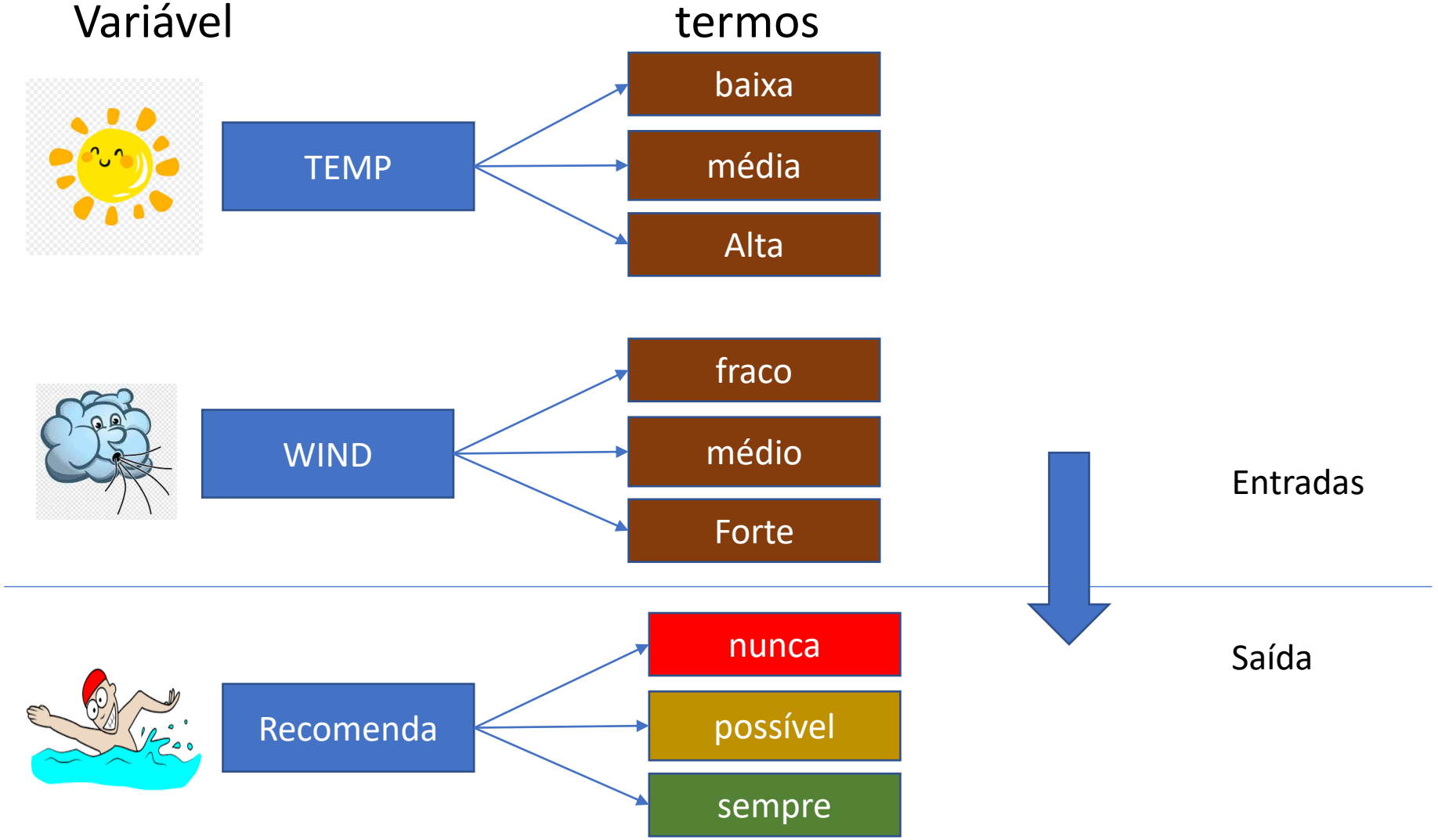
sistema



Problema:

dada uma temperatura e a velocidade do vento, recomendaria ir dar um “pulinho” na piscina?

Problema:

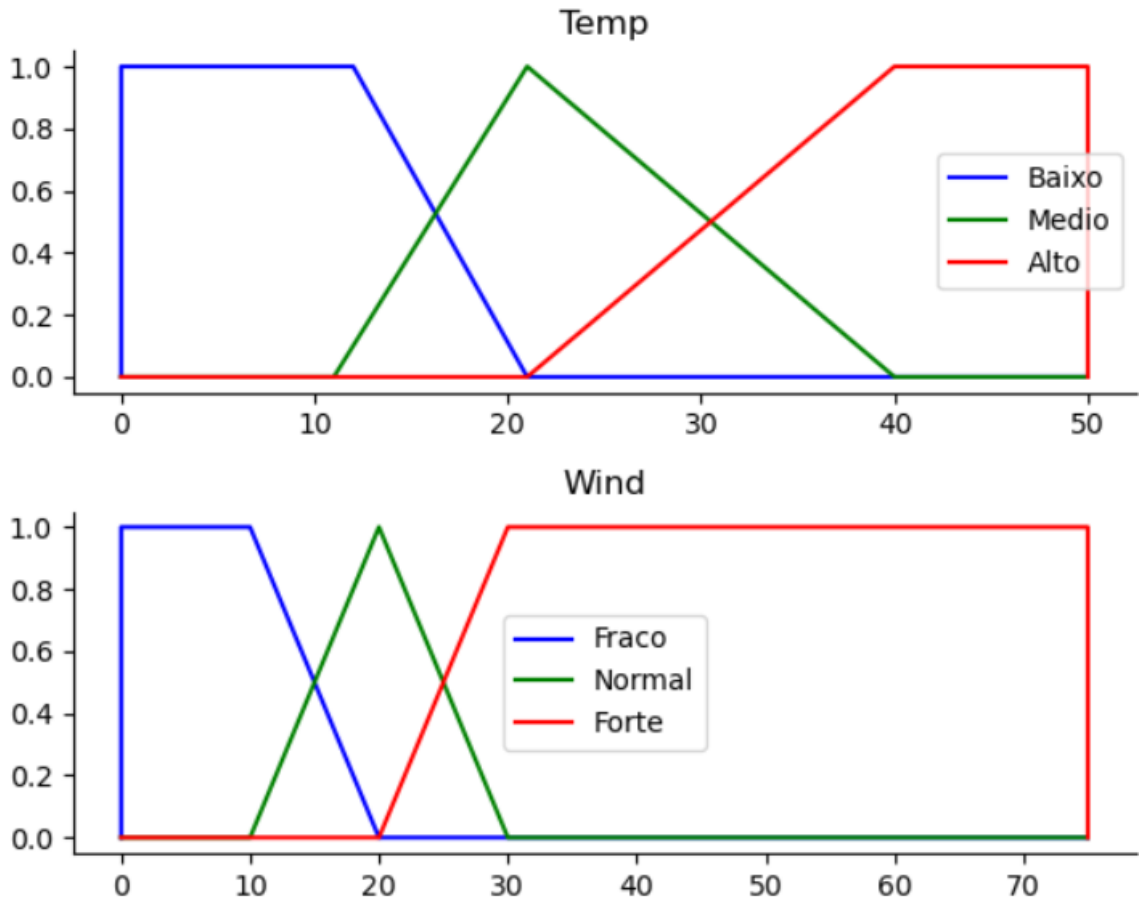

Variável
termos
TEMP
baixa
média
Alta
WIND
fraco
médio
Forte
Entradas
Saída
Recomenda
nunca
possível
sempre

Regras:

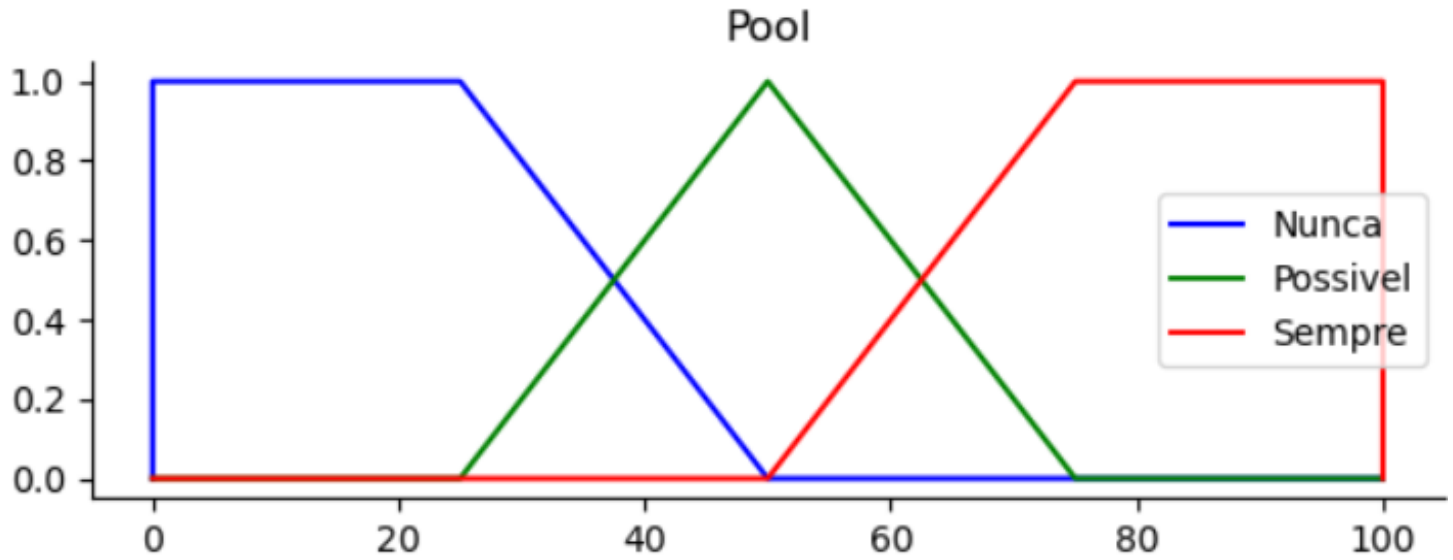
	baixa	média	alta
Fraco	possível	possível	sempre
médio	nunca	possível	possível
forte	nunca	nunca	nunca

Se Temp=baixa E VENTO=fraco ENTÃO PISCINA=possível
Se Temp=baixa E VENTO=médio ENTÃO PISCINA=nunca
Se Temp=baixa E VENTO=forte ENTÃO PISCINA=nunca
Se Temp=média E VENTO=fraco ENTÃO PISCINA=possível
Se Temp=média E VENTO=médio ENTÃO PISCINA=possível
Se Temp=média E VENTO=forte ENTÃO PISCINA=nunca
Se Temp=alta E VENTO=fraco ENTÃO PISCINA=sempre
Se Temp=alta E VENTO=médio ENTÃO PISCINA= possível
Se Temp=alta E VENTO=forte ENTÃO PISCINA=nunca

Funções propostas pelos “experts”

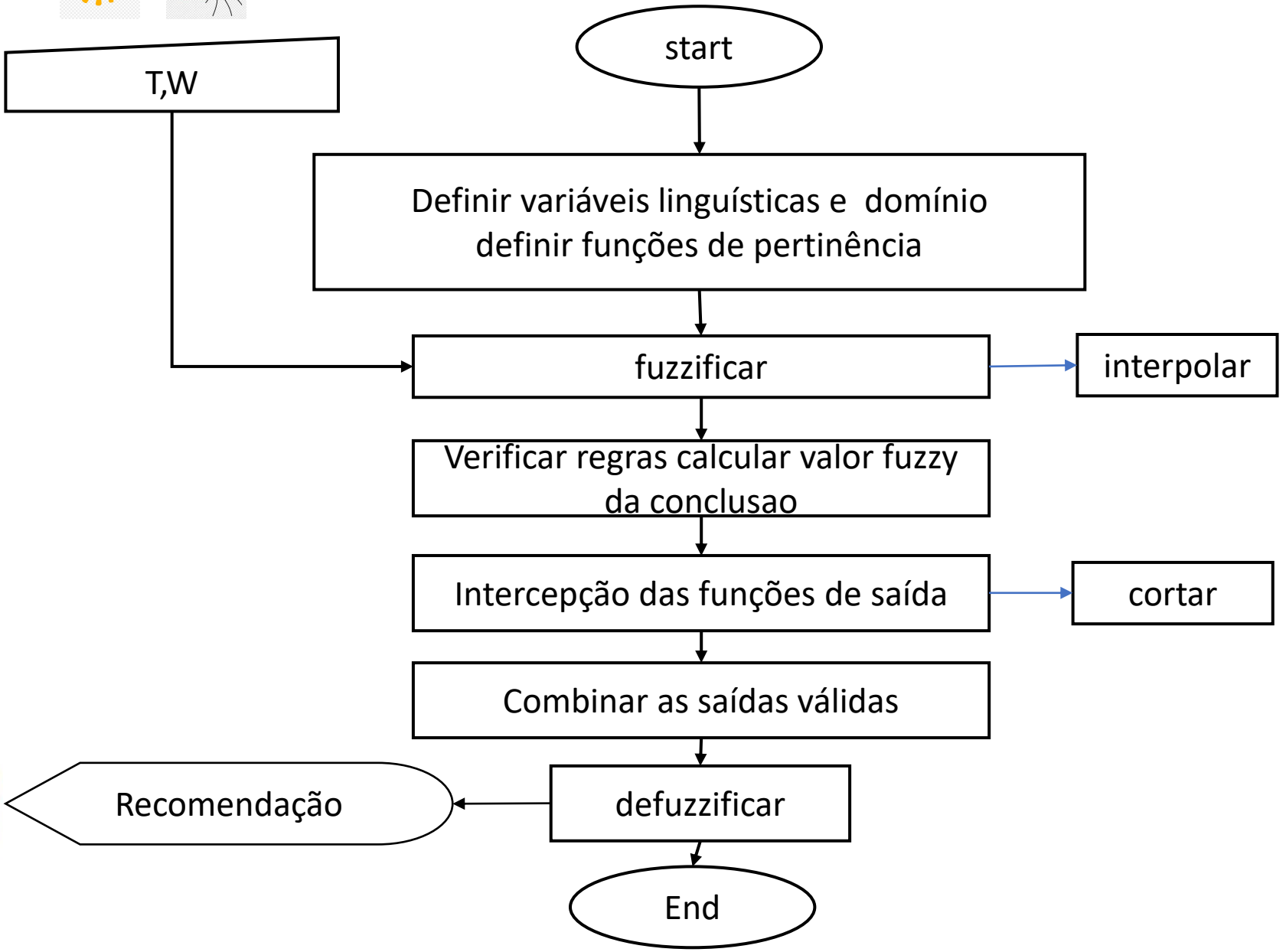


Função de saída, variando de 0-100%

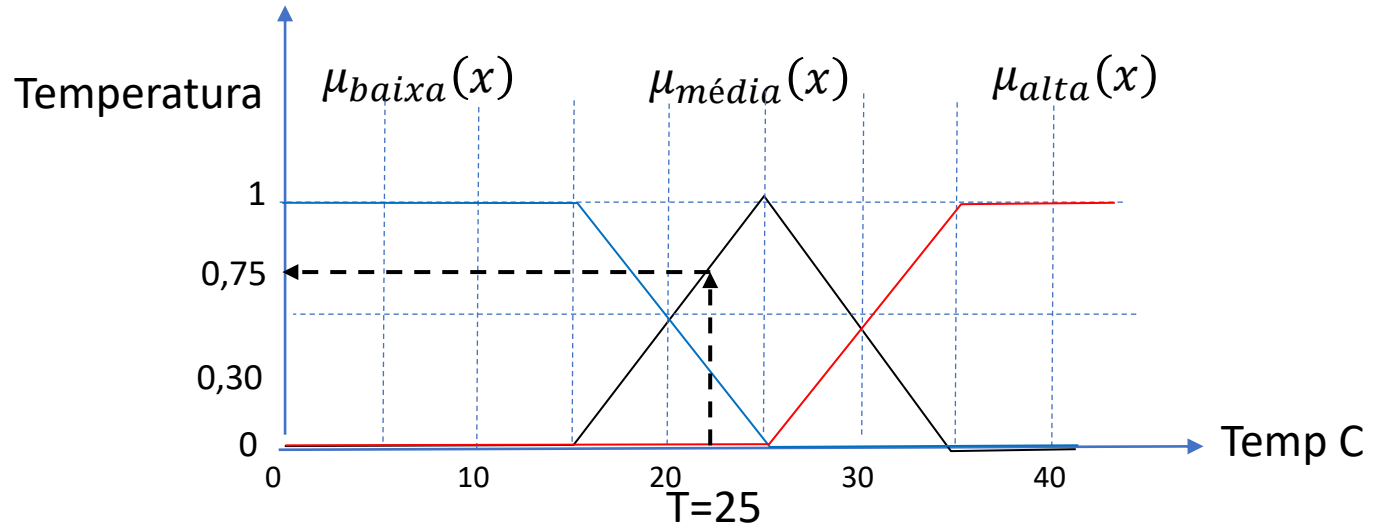


Passos

fluxograma

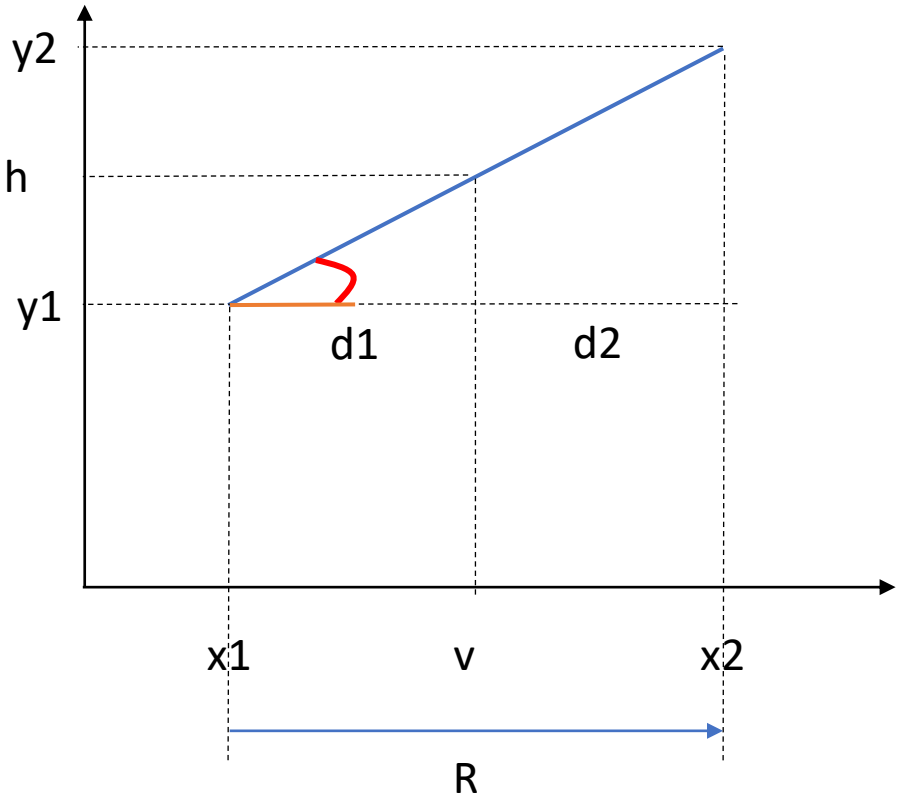


Interpolação para Fuzzificar



Dado um valor, dentro da faixa permitida da variável, é necessário calcular a coordenada “y” (função de pertinência) para este valor.

Interpolação, dados dois pares (XY)



h= = y2 (d1/R) + y1 (d2/R)

tan(a)=(y2-y1)/R

tan(a)=(h-y1)/d1

Igualando:

(y2-y1)/R = (h-y1)/d1

d1 (y2-y1)/R = (h-y1)

h=d1 (y2-y1)/R +y1

= d1*y2/R – d1*y1/R + y1*R/R

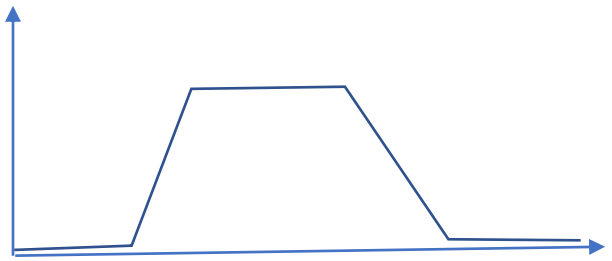
= y2(d1/R) + (y1 R – y1 d1)/R

= y2(d1/R) + y1(R – d1)/R

= y2(d1/R) + y1(d2/R)

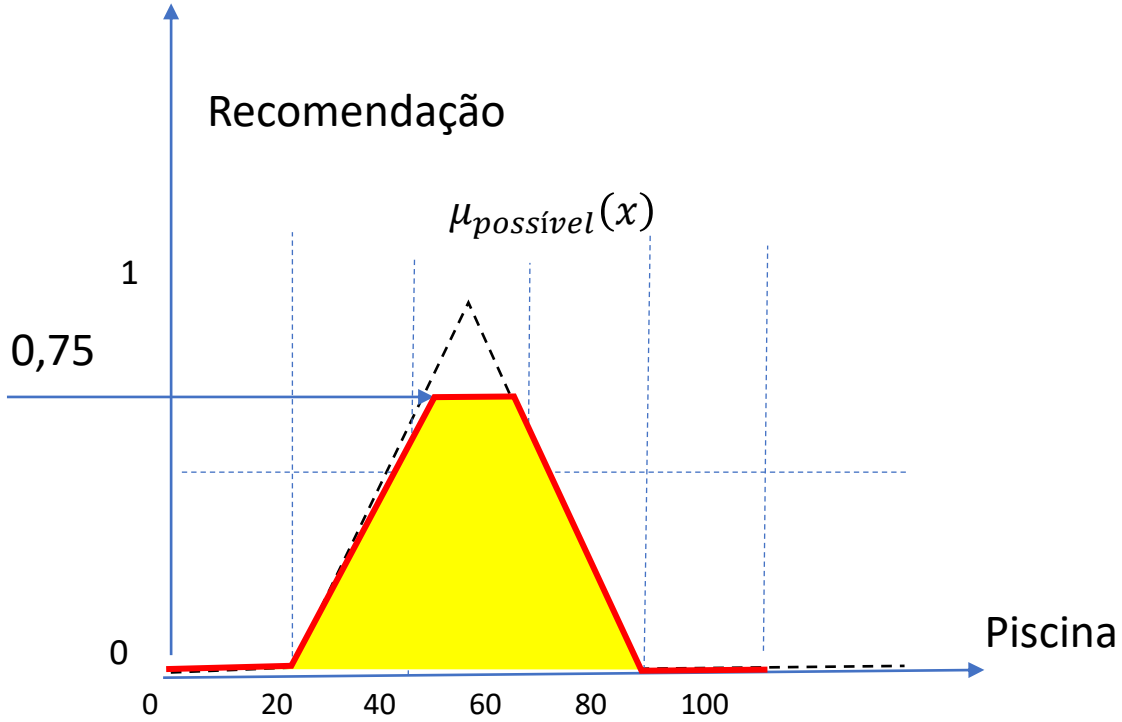
Sub-função de interpolação

```
def interpolar(funcao,Y, valor):
    for i in range(5): #existem 5 segmentos em nossa funcao de 6 ptos
        faixa=funcao[i+1]-funcao[i] # a faixa de um segmento
        if faixa>0: # se x forem iguais, nao fazer nada
        # se sao diferentes, analisa,
        # verificar se o valor se encontra nesta faixa
                if valor>=funcao[i] and valor<=funcao[i+1]:
                        # se está na fiaxa, interpolar
                        d1=valor-funcao[i]
                        d2=faixa-d1
                        y= (d1* Y[i+1] + d2*Y[i])/(faixa)
                        print('Interpolado(',valor,')', y)
return y
```



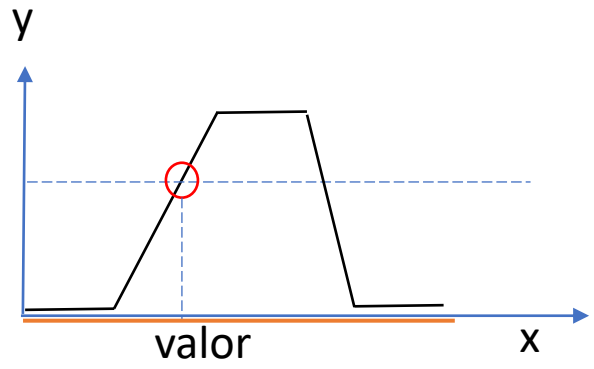
Cortar saída

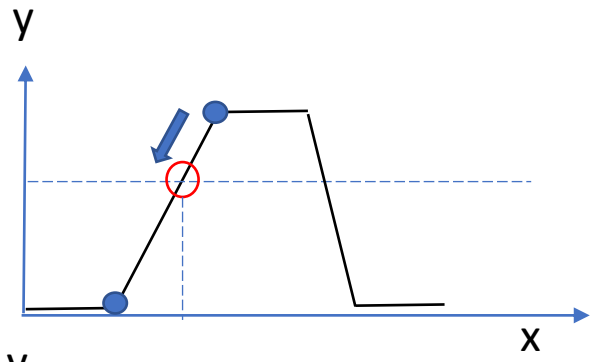
Após avaliar as regras, Transferimos esse valor à conclusão correspondente (possível) e cortamos a função neste valor:

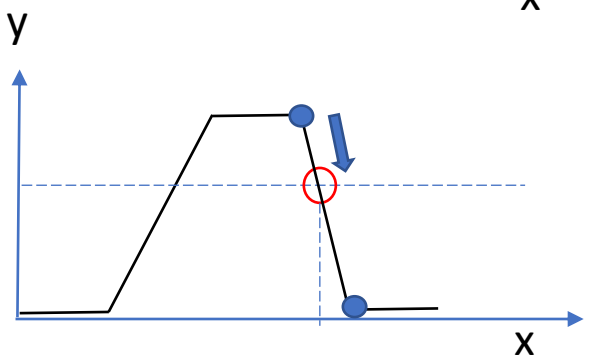


Função: cortar a saída

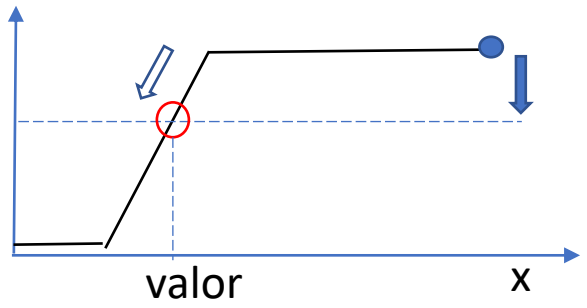
```
def cortar(X,Y, valor):
    x=[-1,-1, -1,-1, -1,-1]  # serie de X e outra de Y,
    y=[0,0,0,0,0,0]
    for i in range(5):  # varia as faixas
        dif1=Y[i]-valor    # distancia ao menor valor
        dif2=Y[i+1]-valor  # distancia ao segundo valor
        if dif1*dif2<0:  # passou de positivo para negativo
            d1=abs(dif1)   # distancias aos extremos
            d2=abs(dif2)
            v=X[i]*d2 + X[i+1]*d1  # valor interpolado
         # O extremo acima do valor deve ser
            # rebaixado, e sua coordenada "x" modificada
            if Y[i]>Y[i+1]:
                x[i]=v       y[i]=valor
            if Y[i]<Y[i+1]:
                x[i+1]=v     y[i+1]=valor
```







```
 # em uma segunda etapa, os valores acima do VALOR que não foram processados são diminuídos
    for i in range(6):
        if Y[i]>valor and x[i]==-1:
            x[i]=v
            y[i]=valor
        if Y[i]<valor:
            x[i]=X[i]
return x,y
```



Nsso programa FUZZY

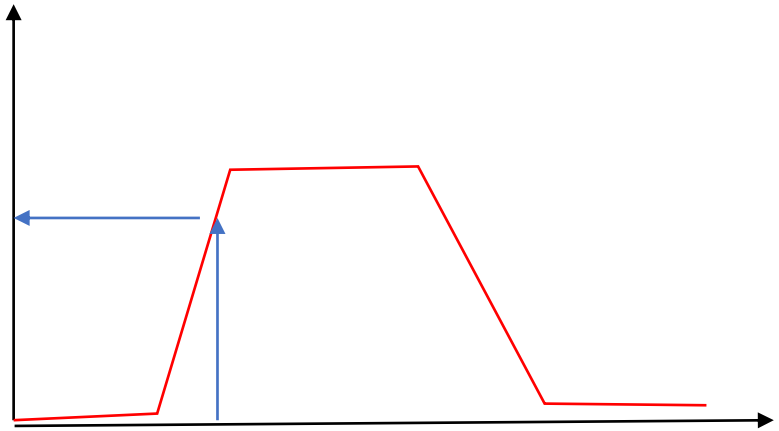
```
#PARTE 1
# Definir variáveis linguísticas e seu domínio, Neste exemplo: Temperatura e Vento
#  saída: recomendação 0-100%
x_Temo = np.arange(0, 50, 1)  # o terceiro numero é o passo. varia de 0-50, com passo 1
x_Wind = np.arange(0, 75, 1)
x_Pool = np.arange(0, 26, 1)
Y=[0,0,1,1,0,0]   # valores de Y da função de pertinenciade ( trapézio)
```

Nsso programa FUZZY

```
#PARTE 1 ...
# agora declaramos as funcoes de pertinência:
Temp_b=[0, 0, 0, 12, 21, 50]
Temp_m=[0 , 11 , 21 , 21 , 40 , 50]
Temp_a=[0, 21, 40, 50, 50, 50]
Wind_b=[ 0, 0, 0, 10, 20, 75]
Wind_m=[0, 10 ,20, 20, 30, 75]
Wind_a=[0, 20, 30, 75, 75, 75]
#saida
Pool_n=[0, 0 ,0  , 25, 50, 100]
Pool_p=[0, 25, 50, 50, 75, 100]
Pool_s=[0, 50, 75, 100, 100, 100]
# para facilitar, definimos sa saida como uma matriz, um arranjo de 3
vetores
Pool= [Pool_n, Pool_p, Pool_s]
```

Fuzzificar e verificar regras

```
#parte 2: Interpolar valores fuzzy
T=28
W=18
T_b=interpolar(Temp_b, Y,T)
T_m=interpolar(Temp_m, Y, T)
T_a=interpolar(Temp_a, Y, T)
W_b=interpolar(Wind_b, Y,W)
W_m=interpolar(Wind_m, Y, W)
W_a=interpolar(Wind_a, Y, W)
```



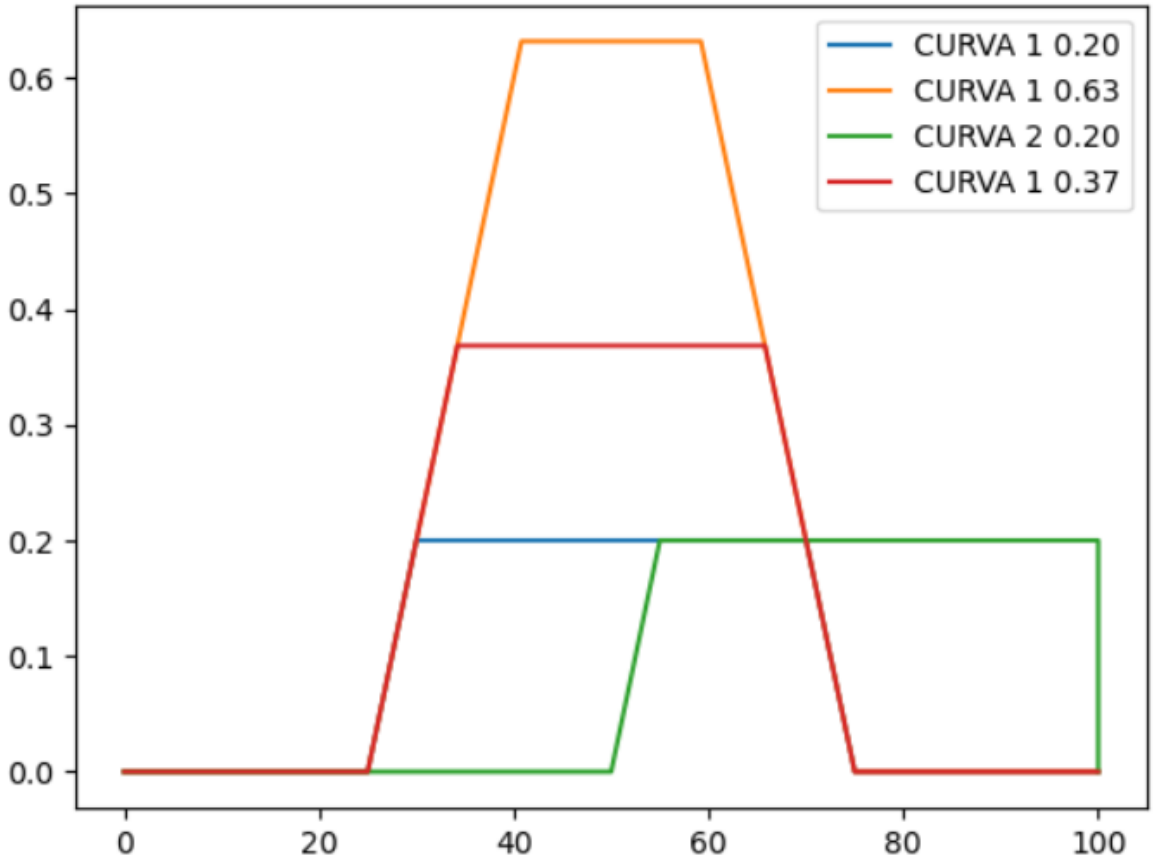
PARTE 3: definimos regras

```
R= [0, 0, 0, 0, 0, 0, 0, 0, 0]   # vetor de regras ativas, a principio todas inativas
CONCLU=[1,0,0,1,1,0,2,1,0]       # Para facilitar, as conclusões (3) são representadas
como números
#       0=nunca,         1=possível,     2=sempre
# e verificamos REGRAS Usando operador MIN
R[0]=min(T_b, W_b)
R[1]=min(T_b, W_m)
R[2]=min(T_b, W_a)
R[3]=min(T_m, W_b)
R[4]=min(T_m, W_m)
R[5]=min(T_m, W_a)
R[6]=min(T_a, W_b)
R[7]=min(T_a, W_m)
R[8]=min(T_a, W_a)
```

Projetar valor nas conclusões

```
conta=0           # contador de regras com conclusão válida
for i in range(NREGRAS):  # varremos todas as regras (sao 9)
    if R[i]>0:  # se a regra for ativa...
        c=CONCLU[i]     # a conclusao pode ser 0,1 ou 2
        conta=conta+1   # contamos q achamos uma conclusao
        print('Regra(', i,') ativa', R[i], 'conclu=',c)
        V=Pool[c]       # separamos a funcao de pertinencia (0,1 ou 2
        vectorx, vectory=cortar(V,Y, R[i])  # cortar a funcao com a altura de R
        if conta==1:
            COX=vectorx # na primeira vez, copiamos os vetores
            COY=vectory
        else:
            COX=COX+vectorx  # depois, concatenamos vetores
            COY=COY+vectory
        print(vectorx, vectory)
```

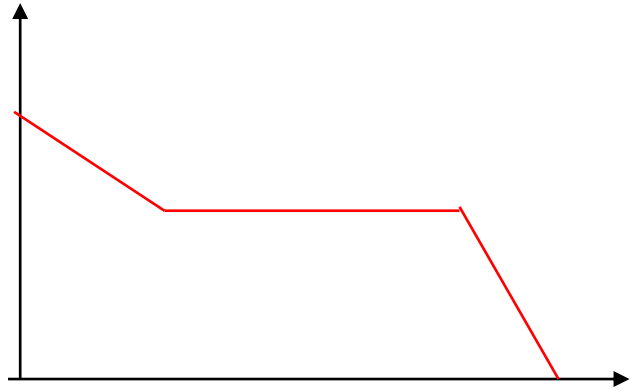
Saídas cortadas



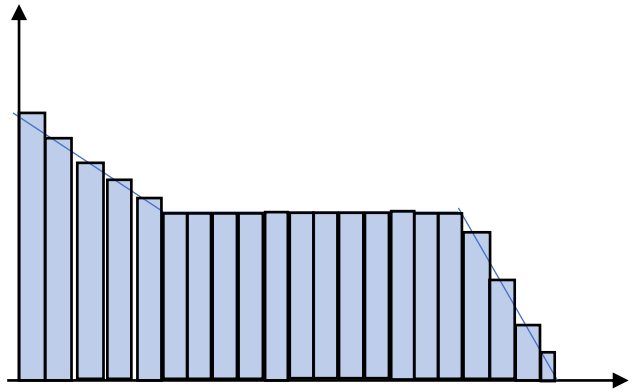
PARTE 5, calcula o centroide das saída, discretizando

Dividimos a função em intervalões regulares (discretização) e calculamos o centroide

Como: X=SOMA(A(i)*X(i)) Area



```
FAIXA=100
passo=1
n=int( np.round( FAIXA/passo) +1 )  # nro de ticks
F= np.zeros((n),dtype = float)   # criação de funcao
vazia para o resultado, discreta
```

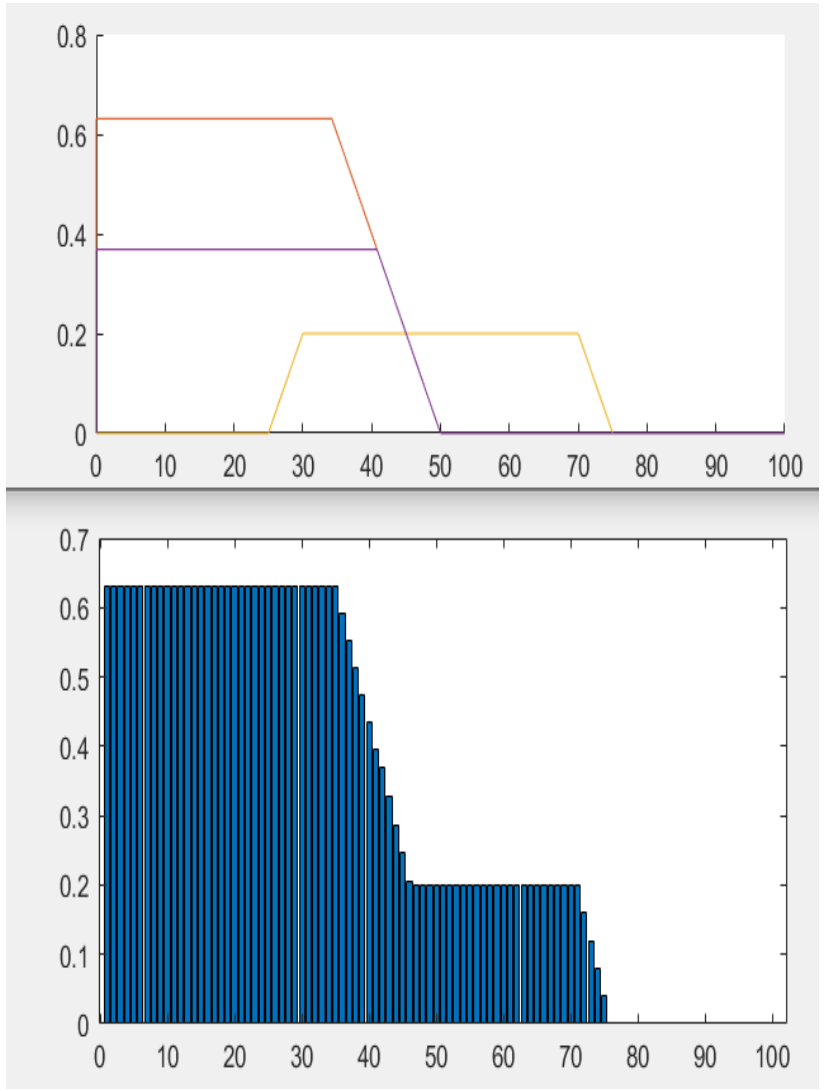


PARTE 5, calcula o centroide das saída, discretizando

```
for i in range (conta):  # varrendo toda  as solucoes validas
    inicio=6*i     #(0, 6, 12, etc...)
    fim=6*(i+1)     # (5, 11, 17,...)
    A=COX[inicio:fim] # copia as coordenadas X e Y da solucao em A e B
    B=COY[inicio:fim]
    for j in range (5): # varrendo todos os segmentos da solucao
        x1=int( np.round( A[j] ) )  # coordenada x do ponto inicial
        x2=int( np.round( A[j+1] ) )  # coordenada x do ponto final
        y1=B[j]     # idem para Y
        y2= B[j+1]
        for k in range(x1,x2): # varrendo apenas este intervalo
            xrange=x2-x1       # Interpolar uma cota para cada "x"
            d1=(k-x1)/ xrange
            d2=1-d1
            y=y1*d2 + y2*d1
            x=np.round(k)+1
            if F[x]<y :   # Armazenar o maior valor!
                F[x]=y
```

PARTE 5, calcula o centroide das saída, discretizando

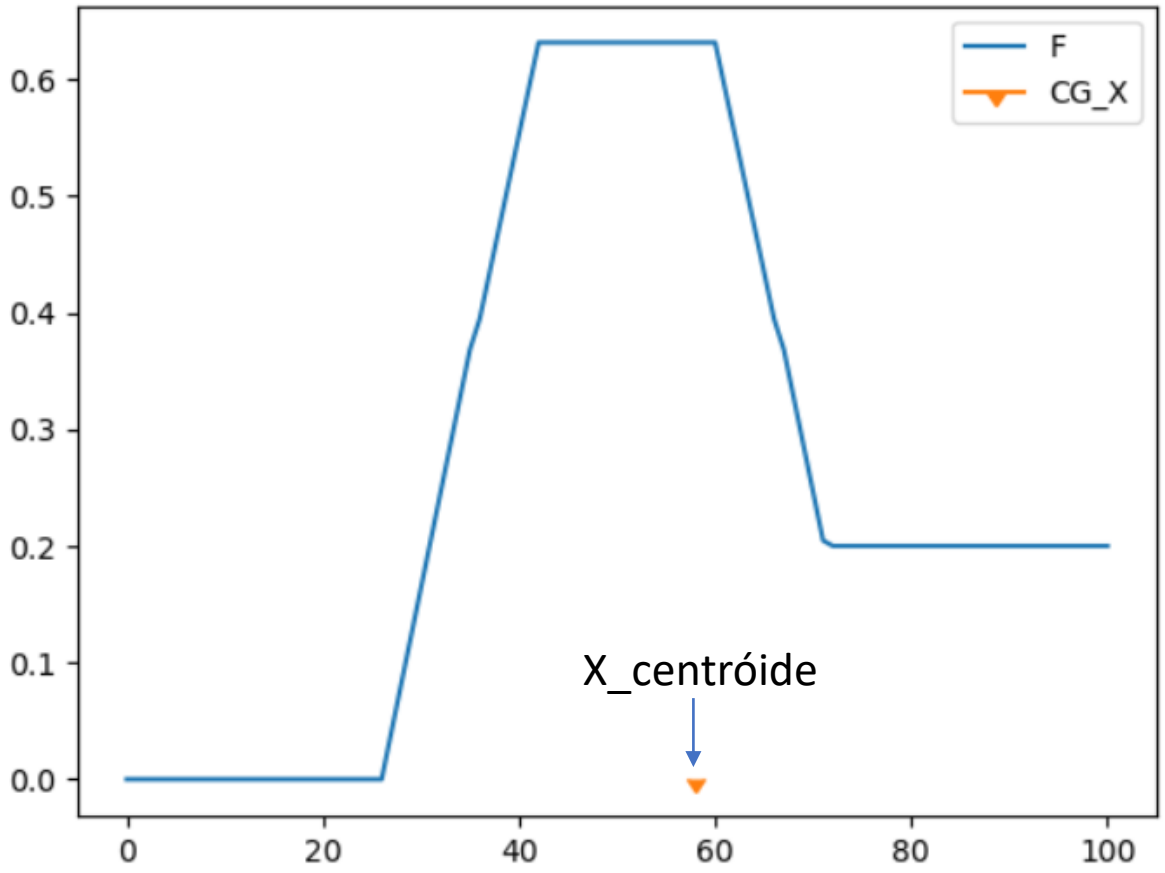
F[x]=discreta



agora calcular centroide deste gráfico

```
area=0  # area total abaixo da curva
xcg=0   # coordenada X do centroide
for i in range(FAIXA):  # varremos toda a faixa de variação (0-100)
    area=area+F[i]   # acumulamos area
    xcg=xcg+i*F[i]   # acumulamos A(i)*x(i) , para calcular centroide
xcg=xcg/area         # calculamos centroide
print('area=%3.2f  xcg=%3.2f\n' % (area, xcg ) );

# mostrar saida
plt.plot(F,label='F')
plt.plot(xcg, 0, marker=11, label='CG_X')
plt.legend()
plt.show()
```


F
CG_X
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
X_centróide


OK, pode me mandar os dados de Temp e Wind!

Já terminou?

Combinar as conclusões

1. Para caombinar os conjuntos fuzzy das conclusões, usa-se o operador “OU”. Ou uma ou a outra regra... logo

`Resumo da Conclusão: max{` $\mu_{P\_nunca}(p), \mu_{P\_possivel}(p)\ \}$