Author: E. Arima (University of Texas at Austin) Department of Geography and the Environment

Date: 09/14/2020

Modifications:

Usage: Stand alone script

Title: Full fuzzy analysis in GIS example for a single cell, including defuzzification phase.

Intro: Most fuzzy interence in GIS out there that I have seen are a stripped down, simplified version (or "vanilla" version) of fuzzy analysis. I have never used fuzzy inference in my own research but wanted to show to my students what a full fuzzy inference analysis would look like. I present here a simple example by illustrating the steps of fuzzy analysis, including the often skipped "defuzzification" part. I presume this part is skipped because it is not available as a function in ArcPro (as of version 3.x). This could be easily introduced in future versions because it is already part of the scikit package.

Overall Approach: The overall approach to cartographic modeling usually follow the steps below.

1. Define objective
2. Define criteria
    - through linguistic values
    - Measure criteria
3. Convert measured criteria into membership (Fuzzification phase)
    - Choose membership functions, parameters
    - Apply functions
4. Apply criteria as rules
    - Apply fuzzy logic, fuzzy algebra
5. Output variable (not usually done in GIS) (Defuzzification phase)
    - Convert linguistic outcome into membership
    - Convert membership back to crisp value

Simple example:

Objective: what is the appropriateness of a particular area (one single cell) for development? Appropriateness is measured on a scale of 0-100.

Criteria:

- Slope is steep OR area far from UT then area not good
- If slope is average, then okay

- Slope is gentle OR close to UT, then very good

First step is to convert a particular observed value of slope and distance to UT to each one of the classes.

Suppose our cell slope value is 6^o and is 11 km from UT.

Import modules of interest. We need numpy, skfuzzy, and matplotlib to plot graphs

In [1]:
```
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
```

```
C:\Users\ea9267\.conda\envs\geo\lib\site-packages\scipy\__init__.py:146: UserWarning:
A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected
version 1.23.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Create arrays with the potential range of values for the two variables (this is only to plot figures). We know that slope in degrees is between 0-89.9 and let's say that the longest distance in our study area is 30 km.

In [2]:
```
x_slope = np.arange(0, 90,1)
x_dist = np.arange(0,31,1)
```

In [3]:
```
x_dist
```

Out[3]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])
```

In [4]:
```
x_slope
```

Out[4]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
       51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,
       68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,
       85, 86, 87, 88, 89])
```

Now, let's choose the membership function that will convert the slope and distance to membership class values. There are plenty of options here. Let's illustrate some in graphs.

Triangular membership functions

In [5]:
```
slp_gent = fuzz.trimf(x_slope, [0,0, 5])
slp_med = fuzz.trimf(x_slope, [0,10, 20])
slp_stp = fuzz.trimf(x_slope, [15, 99, 99])
```

Sigmoid membership functions sigmf(x,b,c) b center value of sigmoid where membership == 0.5 c width of sigmoid, how fast approaches 0 or 1. If negative, sigmoid will be decreasing, if positive, will be increasing.

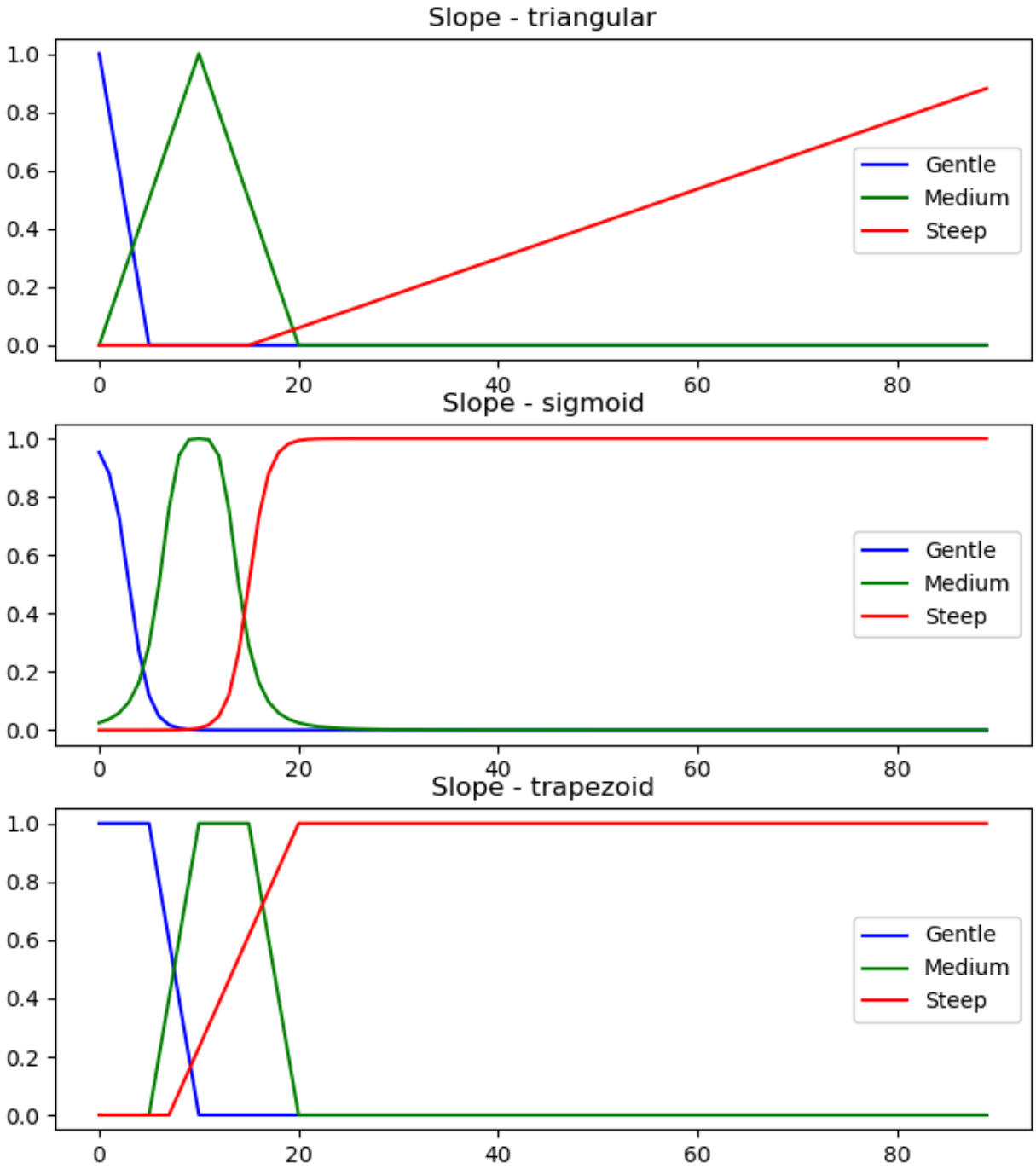
In [6]:
```
slp_gent1 = fuzz.sigmf(x_slope, 3, -1)
slp_med1 = fuzz.gbellmf(x_slope, 4, 2, 10)
slp_stp1 = fuzz.sigmf(x_slope, 15, 1)
```

Trapezoidal membership functions

In [7]:
```
slp_gent2 = fuzz.trapmf(x_slope, [0,0,5,10])
slp_med2 = fuzz.trapmf(x_slope, [5, 10, 15, 20])
slp_stp2 = fuzz.trapmf(x_slope, [7, 20, 99,99])
```

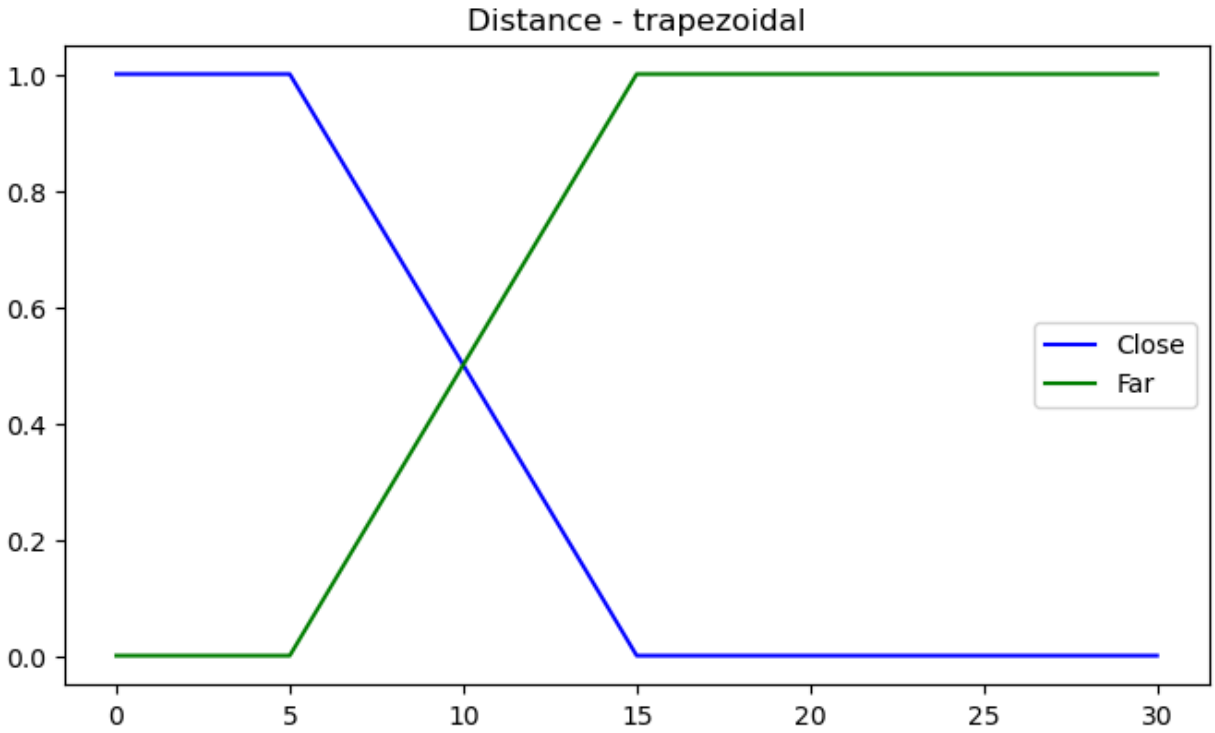
Plot those graphs to see the difference

In [8]:
```
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))
ax0.plot(x_slope, slp_gent, 'b', linewidth=1.5, label='Gentle')
ax0.plot(x_slope, slp_med, 'g', linewidth=1.5, label='Medium')
ax0.plot(x_slope, slp_stp, 'r', linewidth=1.5, label='Steep')
ax0.set_title('Slope - triangular')
ax0.legend()

ax1.plot(x_slope, slp_gent1, 'b', linewidth=1.5, label='Gentle')
ax1.plot(x_slope, slp_med1, 'g', linewidth=1.5, label='Medium')
ax1.plot(x_slope, slp_stp1, 'r', linewidth=1.5, label='Steep')
ax1.set_title('Slope - sigmoid')
ax1.legend()

ax2.plot(x_slope, slp_gent2, 'b', linewidth=1.5, label='Gentle')
ax2.plot(x_slope, slp_med2, 'g', linewidth=1.5, label='Medium')
ax2.plot(x_slope, slp_stp2, 'r', linewidth=1.5, label='Steep')
ax2.set_title('Slope - trapezoid')
ax2.legend()
outFig = 'fuzzification_fcts.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



For distance to UT, let's see how a trapezoidal function would look like

In [9]:
```
dclose = fuzz.trapmf(x_dist, [0,0,5,15])
dfar = fuzz.trapmf(x_dist, [5,15, 30,30])
fig, (ax0) = plt.subplots(nrows=1, figsize=(8, 4.5))
ax0.plot(x_dist, dclose, 'b', linewidth=1.5, label='Close')
ax0.plot(x_dist, dfar, 'g', linewidth=1.5, label='Far')
ax0.set_title('Distance - trapezoidal')
ax0.legend()
outFig = 'trapezoidal_distance.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



Now, let's define the values for our cell. If this were a raster, we would input our raster array. Slope is 6, and distance is 11.

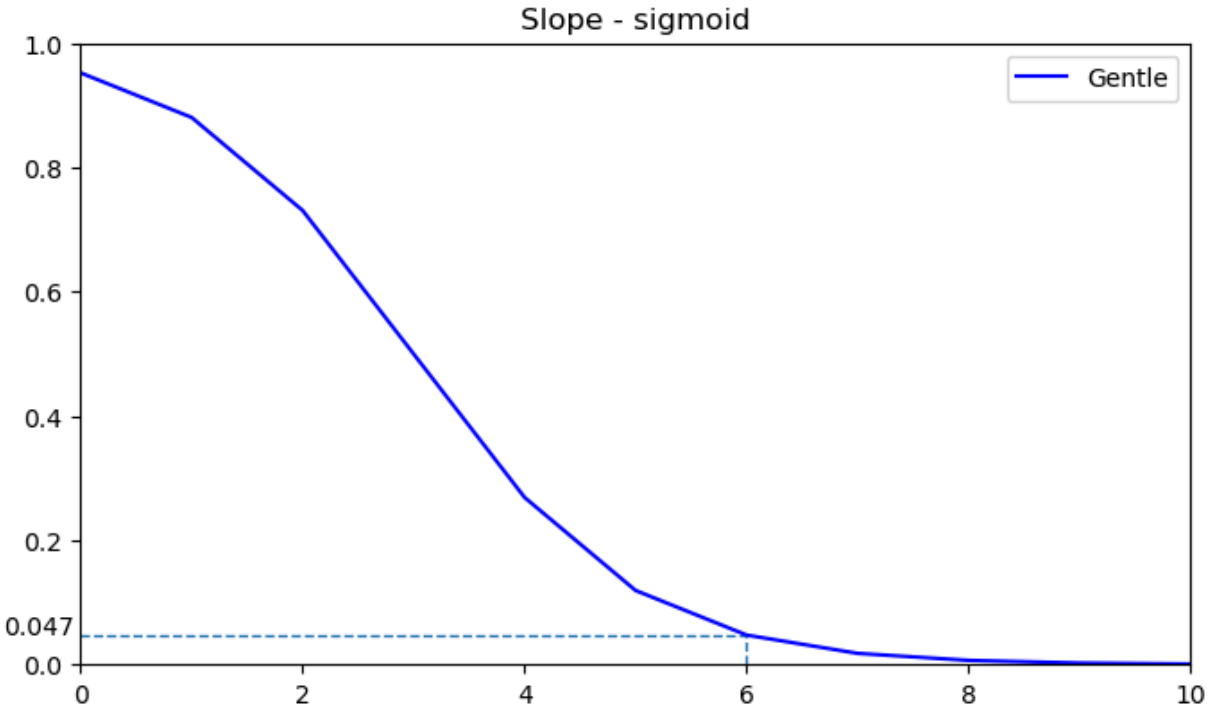
In [10]:
```
#Slope value of the cell in question
slp  = np.array(6.0)
```

In [11]:
```
#Distance to UT
d2ut = np.array(11.0)
```

Here is the part that we convert those values to membership to the classes we define. Let's use sigmoid functions for slope and trapezoidal for distance.

In [12]:
```
slp_f = fuzz.interp_membership(x_slope, slp_gent1, slp)
slp_m = fuzz.interp_membership(x_slope, slp_med1, slp)
slp_s = fuzz.interp_membership(x_slope, slp_stp1, slp)
```

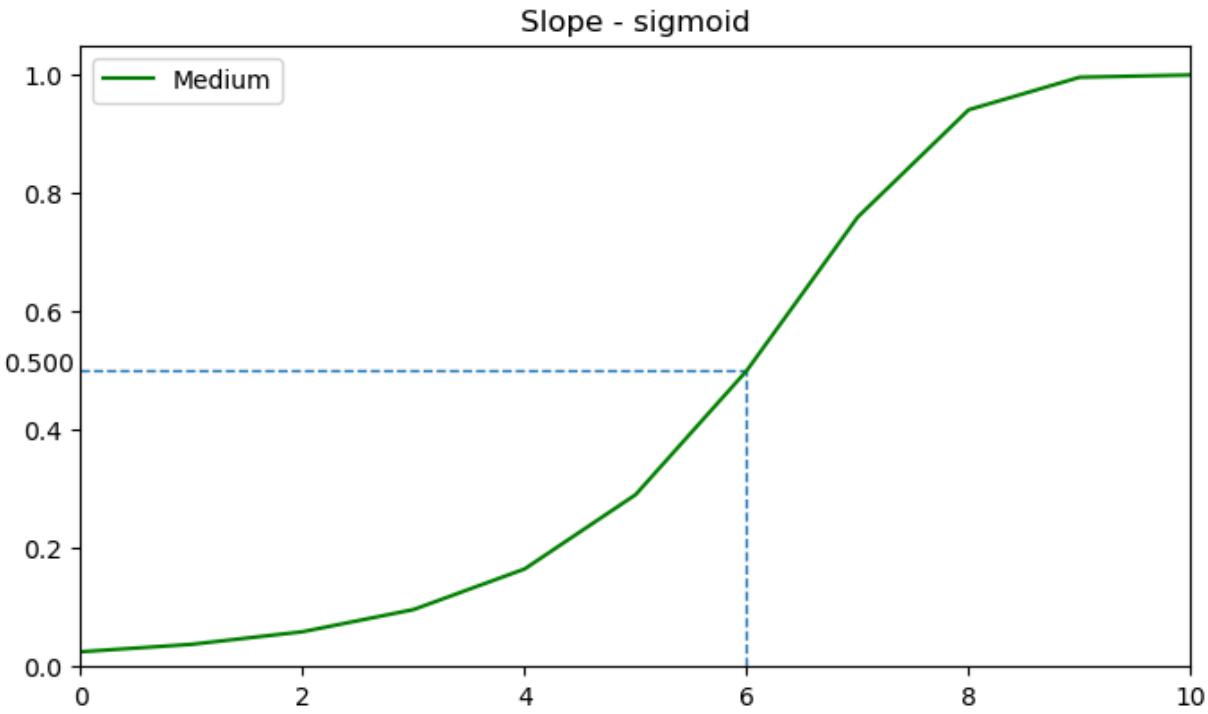
In [13]:
```
fig,ax1 = plt.subplots(nrows=1, figsize=(8, 4.5))
ax1.plot(x_slope, slp_gent1, 'b', linewidth=1.5, label='Gentle')
ax1.set_title('Slope - sigmoid')
ax1.set_xlim([0, 10])
ax1.set_ylim([0,1])
ax1.legend()
lab1 = f'{slp_f:.3f}'
ax1.annotate(lab1, xy=(slp, slp_f), xytext=(-0.7, slp_f))
plt.vlines(x = slp, ymin = 0, ymax = slp_f, linewidth = 1.0, linestyles = "dashed")
plt.hlines(y = slp_f, xmin = 0, xmax = slp, linewidth = 1.0, linestyles = "dashed")
plt.savefig("slope_gentle.png", dpi = 300)
plt.show()
```



In [14]:
```
f'Membership to gentle slope is: {slp_f:.3f}'
```

Out[14]:
```
'Membership to gentle slope is: 0.047'
```

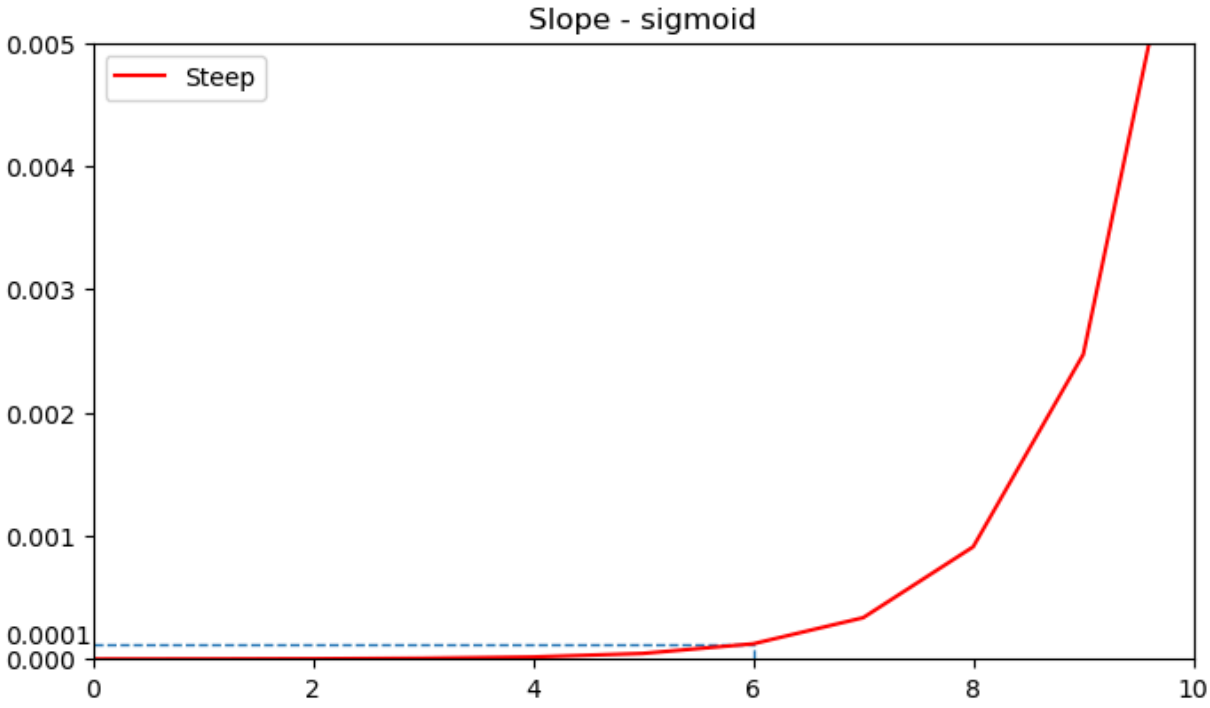
In [15]:
```
fig,ax1 = plt.subplots(nrows=1, figsize=(8, 4.5))
ax1.plot(x_slope, slp_med1, 'g', linewidth=1.5, label='Medium')
ax1.set_title('Slope - sigmoid')
ax1.set_xlim([0, 10])
ax1.set_ylim([0,1.05])
ax1.legend()
lab1 = f'{slp_m:.3f}'
ax1.annotate(lab1, xy=(slp, slp_m), xytext=(-0.7, slp_m))
plt.vlines(x = slp, ymin = 0, ymax = slp_m, linewidth = 1.0, linestyles = "dashed")
plt.hlines(y = slp_m, xmin = 0, xmax = slp, linewidth = 1.0, linestyles = "dashed")
plt.savefig("slope_medium.png", dpi = 300)
plt.show()
```



In [16]:
```
f'Membership to medium slope is: {slp_m:.3f}'
```

Out[16]:
```
'Membership to medium slope is: 0.500'
```

In [17]:
```
fig,ax1 = plt.subplots(nrows=1, figsize=(8, 4.5))
ax1.plot(x_slope, slp_stp1, 'r', linewidth=1.5, label='Steep')
ax1.set_title('Slope - sigmoid')
ax1.set_xlim([0, 10])
ax1.set_ylim([0,0.005])
ax1.legend()
lab1 = f'{slp_s:.4f}'
ax1.annotate(lab1, xy=(slp, slp_s), xytext=(-0.8, slp_s))
plt.vlines(x = slp, ymin = 0, ymax = slp_s, linewidth = 1.0, linestyles = "dashed")
plt.hlines(y = slp_s, xmin = 0, xmax = slp, linewidth = 1.0, linestyles = "dashed")
plt.savefig("slope_steep.png", dpi = 300)
plt.show()
```



In [18]:
```
f'Membership to steep slope is: {slp_s:.5f}'
```

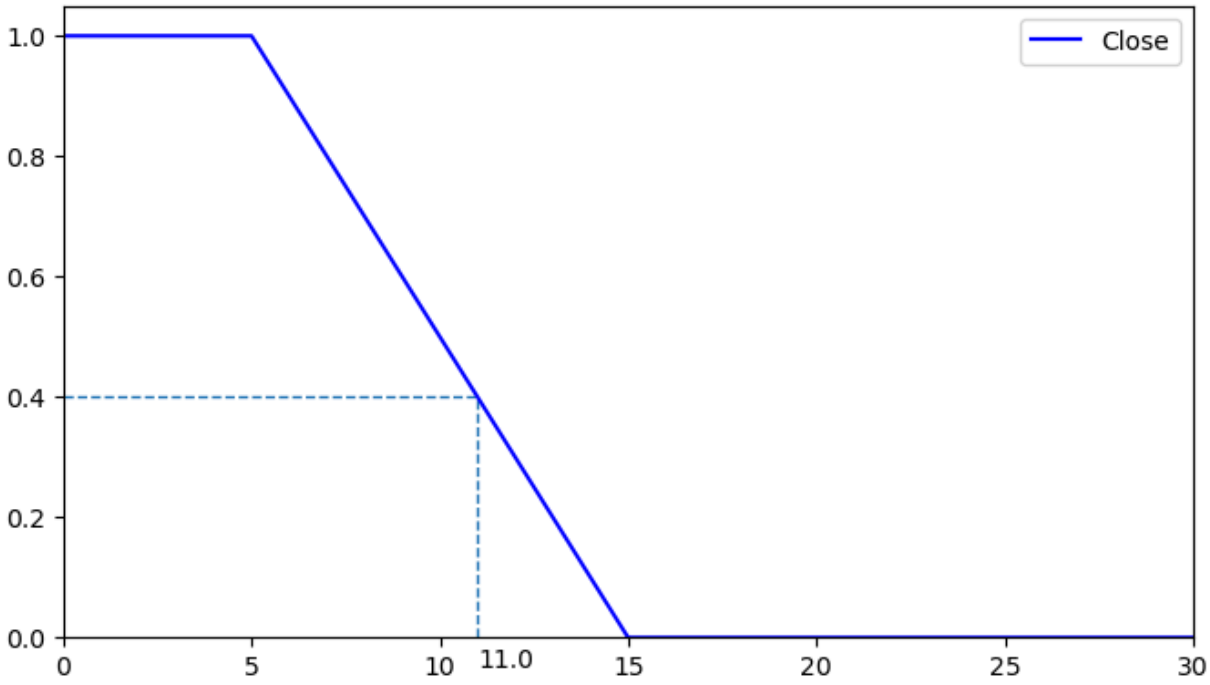
Out[18]:
```
'Membership to steep slope is: 0.00012'
```

In [19]:
```
dist_c = fuzz.interp_membership(x_dist, dclose, d2ut)
dist_f = fuzz.interp_membership(x_dist, dfar, d2ut)
```

In [20]:
```
f'Membership to close to UT is: {dist_c:.3f}'
```

Out[20]:
```
'Membership to close to UT is: 0.400'
```

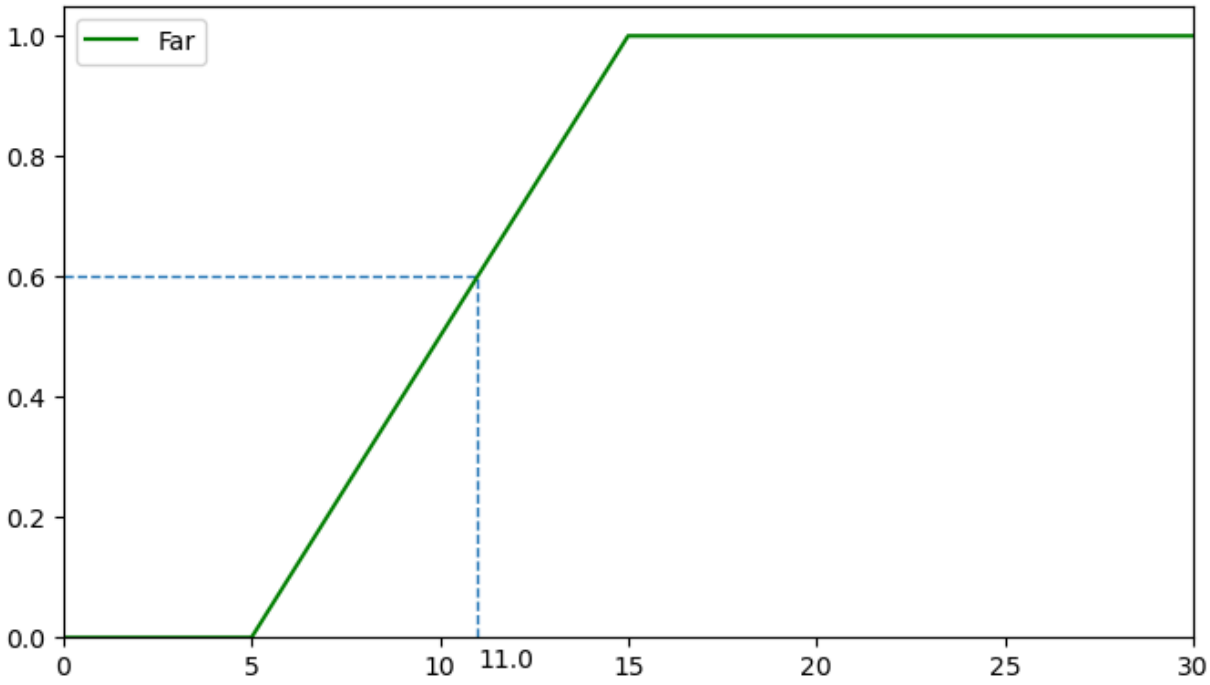
In [21]:
```
fig, (ax1) = plt.subplots(nrows=1, figsize=(8, 4.5))
ax1.plot(x_dist, dclose, 'b', linewidth=1.5, label='Close')
#ax0.plot(x_dist, dfar, 'g', linewidth=1.5, label='Far')
ax1.set_xlim([0, 30])
ax1.set_ylim([0,1.05])
ax1.legend()
lab1 = f'{dist_c:.3f}'
lab2 = str(d2ut)
ax1.annotate(lab2, xy=(d2ut, 0), xytext=(d2ut, -0.05))
plt.vlines(x = d2ut, ymin = 0, ymax = dist_c, linewidth = 1.0, linestyles = "dashed")
plt.hlines(y = dist_c, xmin = 0, xmax = d2ut, linewidth = 1.0, linestyles = "dashed")
plt.savefig("distance_close.png", dpi = 300)
plt.show()
```



In [22]:
```
f"Membership to far from UT is: {dist_f:.3f}"
```

Out[22]:
```
'Membership to far from UT is: 0.600'
```

In [23]:
```
fig, (ax1) = plt.subplots(nrows=1, figsize=(8, 4.5))
ax1.plot(x_dist, dfar, 'g', linewidth=1.5, label='Far')
ax1.set_xlim([0, 30])
ax1.set_ylim([0,1.05])
ax1.legend()
lab1 = f'{dist_f:.3f}'
lab2 = str(d2ut)
ax1.annotate(lab2, xy=(d2ut, 0), xytext=(d2ut, -0.05))
plt.vlines(x = d2ut, ymin = 0, ymax = dist_f, linewidth = 1.0, linestyles = "dashed")
plt.hlines(y = dist_f, xmin = 0, xmax = d2ut, linewidth = 1.0, linestyles = "dashed")
plt.savefig("distance_far.png", dpi = 300)
plt.show()
```



Apply Rules and use fuzzy algebra. Rule 1: Steep slope OR far from UT. OR is max value of either one. Fmax is an element wise operation (would matter if it were a higher dimensional array)

In [24]:
```
rule1 = np.fmax(slp_s, dist_f)
```

In [25]:
```
f"rule 1 outcome is: {rule1:.3f}"
```

Out[25]:
```
'rule 1 outcome is: 0.600'
```

Rule 2: Slope is average (nothing to do here, just pass on the value)

In [26]:
```
rule2 = slp_m
```

In [27]:
```
f"rule 2 outcome is: {rule2:.3f}"
```

Out[27]:
```
'rule 2 outcome is: 0.500'
```

Rule 3: Slope is gentle OR close to UT. Again, OR calls for max.

In [28]:
```
rule3 = np.fmax(slp_f, dist_c)
```

In [29]:
```
f"rule 3 outcome is: {rule3:.3f}"
```

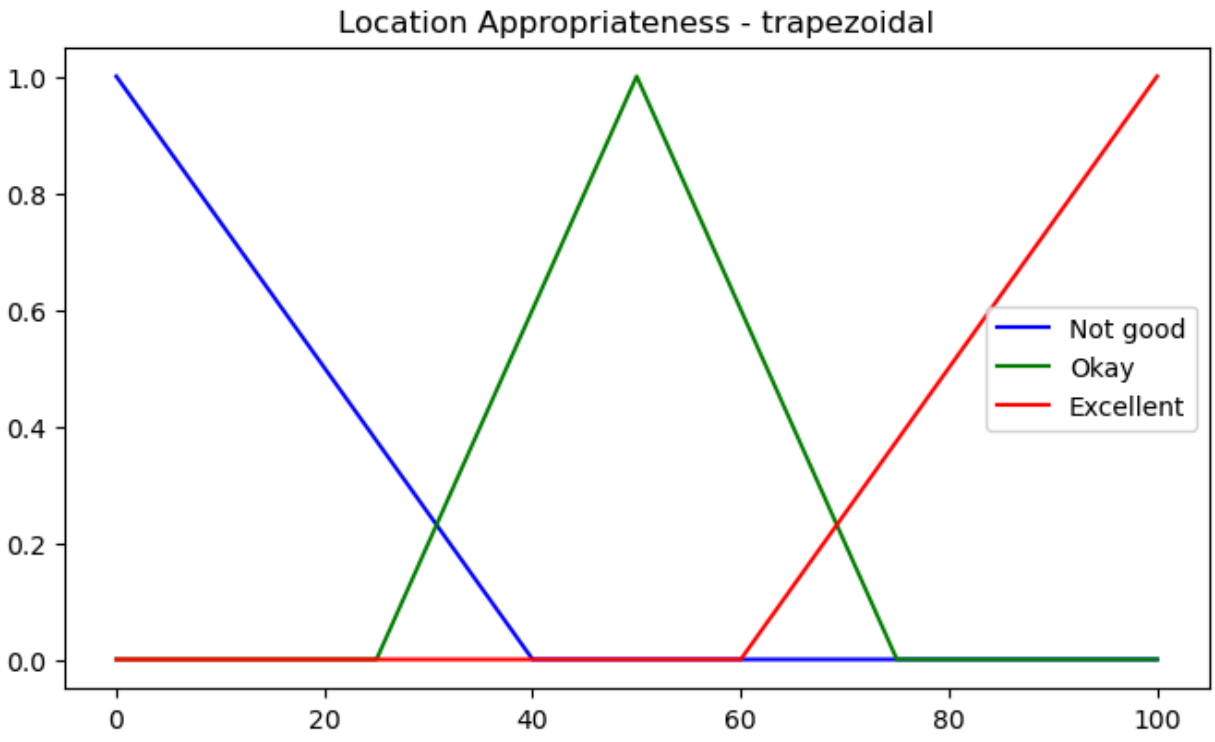
Out[29]:
```
'rule 3 outcome is: 0.400'
```

Now we have to activate the rule, which means that we have to translate each rule into the verbal outcome not good, okay, and excellent. To to that, we have to also create membership functions for that. Remember that appropriateness ranges from 0-100.

In [30]:
```
x_appr = np.arange(0,101,1)
```

Let's use triangular membership functions

In [31]:
```
nogood = fuzz.trimf(x_appr, [0, 0,40])
okay = fuzz.trimf(x_appr, [25, 50, 75])
excl = fuzz.trimf(x_appr, [60, 100, 100])
fig, (ax0) = plt.subplots(nrows=1, figsize=(8, 4.5))
ax0.plot(x_appr, nogood, 'b', linewidth=1.5, label='Not good')
ax0.plot(x_appr, okay, 'g', linewidth=1.5, label='Okay')
ax0.plot(x_appr, excl, 'r', linewidth=1.5, label='Excellent')
ax0.set_title('Location Appropriateness - trapezoidal')
ax0.legend()
outFig = 'defuzzification_fct.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



Now translate the rules into a member of the not good, okay, excellent (called 'activate rule')

In [32]:
```
appr_nogood = np.fmin(rule1, nogood)
```

In [33]:
```
appr_nogood
```

Out[33]:
```
array([0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.6  ,
       0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.6  , 0.575,
       0.55 , 0.525, 0.5  , 0.475, 0.45 , 0.425, 0.4  , 0.375, 0.35 ,
       0.325, 0.3  , 0.275, 0.25 , 0.225, 0.2  , 0.175, 0.15 , 0.125,
       0.1  , 0.075, 0.05 , 0.025, 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
       0.   , 0.   ])
```

In [34]:
```
appr_okay = np.fmin(rule2, okay)
```

In [35]:
```
appr_exc = np.fmin(rule3, excl)
```

Final step is to defuzzify. To do that, we have to aggregate the outputs by picking the max value

In [36]:
```
appragg = np.fmax(appr_nogood, np.fmax(appr_okay, appr_exc))
```

And then picking an algorith to translate the array into a single appropriatness output (back to crisp value)

Centroid algorithm

In [37]:
```
app_per1 = fuzz.defuzz(x_appr, appragg, 'centroid')
```

Mean of maximum algorithm

In [38]:
```
app_per2 = fuzz.defuzz(x_appr, appragg, 'mom')
```

Max of maximum algorithm

In [39]:
```
app_per3 = fuzz.defuzz(x_appr, appragg, 'lom')
```

In [40]:
```
f"Using the centroid rule, appropriateness is: {app_per1:.3f}"
```

Out[40]:
```
'Using the centroid rule, appropriateness is: 46.448'
```

In [41]:
```
f"Using the mean of maximum rule, appropriateness is: {app_per2:.3f}"
```

Out[41]:
```
'Using the mean of maximum rule, appropriateness is: 8.000'
```

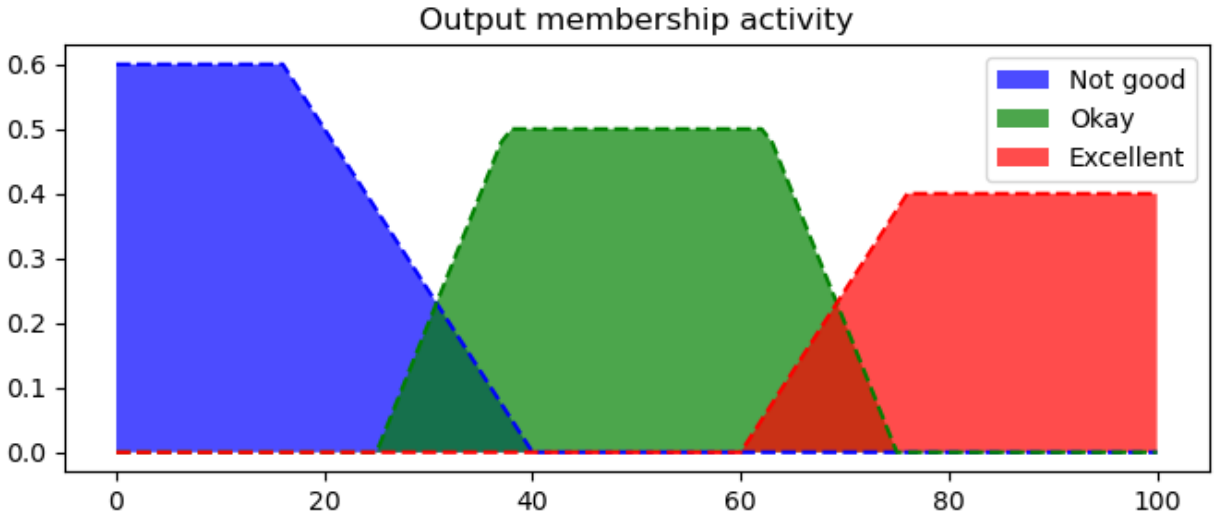
In [42]:
```
f"Using the max of maximum rule, appropriateness is: {app_per3:.3f}"
```

Out[42]:
```
'Using the max of maximum rule, appropriateness is: 16.000'
```

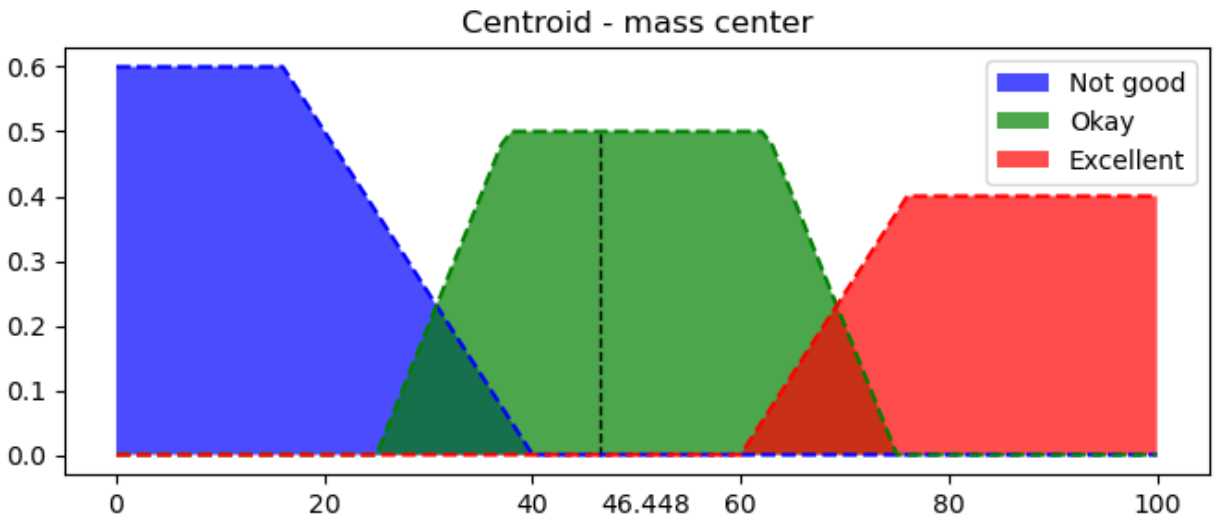
To visualize where these numbers come from, plot the graph of activation values

In [43]:
```
ap0 = np.zeros_like(x_appr)
```

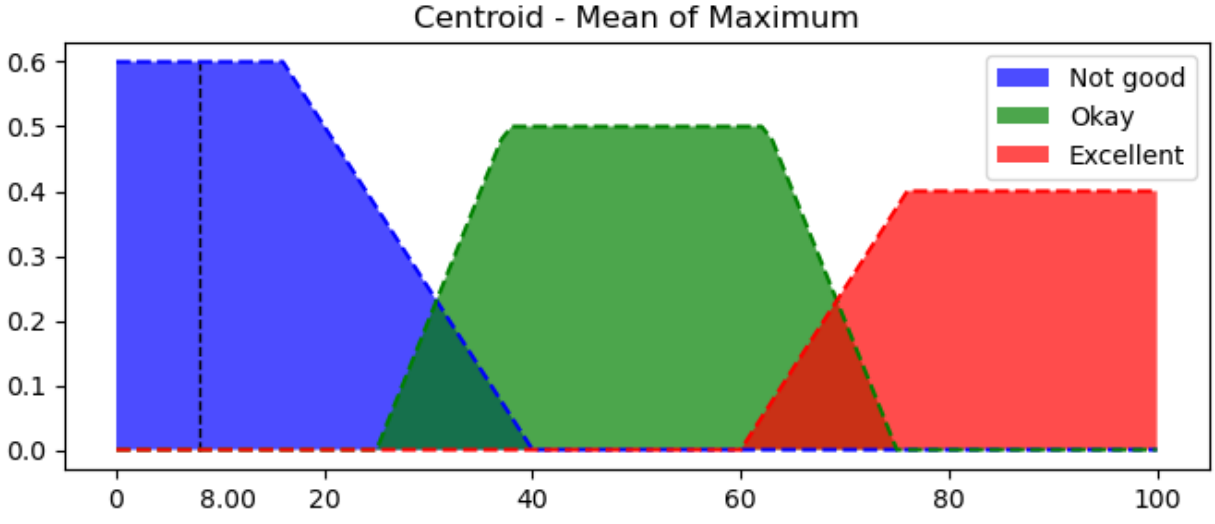
```
fig, ax0 = plt.subplots(figsize=(8,3))

ax0.fill_between(x_appr, ap0, appr_nogood, facecolor = 'b', alpha= 0.7, label = "Not g
ax0.plot(x_appr, appr_nogood, 'b', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_okay, facecolor = 'g', alpha= 0.7, label = "Okay")
ax0.plot(x_appr, appr_okay, 'g', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_exc, facecolor = 'r', alpha= 0.7, label = "Excellen
ax0.plot(x_appr, appr_exc, 'r', linestyle = '--')
ax0.set_title('Output membership activity')
ax0.legend()
outFig = 'defuzzification_activity.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



In [44]:
```
ap0 = np.zeros_like(x_appr)

fig, ax0 = plt.subplots(figsize=(8,3))

ax0.fill_between(x_appr, ap0, appr_nogood, facecolor = 'b', alpha= 0.7, label = "Not g
ax0.plot(x_appr, appr_nogood, 'b', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_okay, facecolor = 'g', alpha= 0.7, label = "Okay")
ax0.plot(x_appr, appr_okay, 'g', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_exc, facecolor = 'r', alpha= 0.7, label = "Excellen
ax0.plot(x_appr, appr_exc, 'r', linestyle = '--')
ax0.set_title('Centroid - mass center')
lab1 = f'{app_per1:.3f}'
ax0.annotate(lab1, xy=(app_per1, 0), xytext=(app_per1, -0.09))
ax0.legend()
plt.vlines(x = app_per1, ymin = 0, ymax= 0.5, linewidth = 1.0, color = "k", linestyles
outFig = 'defuzzification_activity_centroid.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```

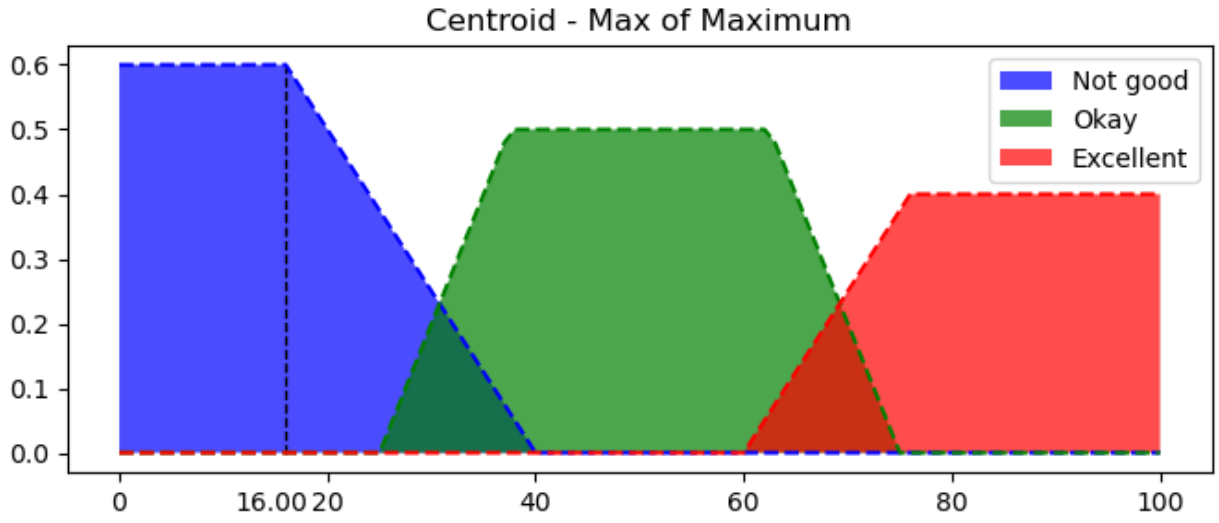


In [45]:
```
ap0 = np.zeros_like(x_appr)

fig, ax0 = plt.subplots(figsize=(8,3))

ax0.fill_between(x_appr, ap0, appr_nogood, facecolor = 'b', alpha= 0.7, label = "Not g
ax0.plot(x_appr, appr_nogood, 'b', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_okay, facecolor = 'g', alpha= 0.7, label = "Okay")
ax0.plot(x_appr, appr_okay, 'g', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_exc, facecolor = 'r', alpha= 0.7, label = "Excellen
ax0.plot(x_appr, appr_exc, 'r', linestyle = '--')
ax0.set_title('Centroid - Mean of Maximum')
lab1 = f'{app_per2:.2f}'
ax0.annotate(lab1, xy=(app_per2, 0), xytext=(app_per2, -0.09))
ax0.legend()
plt.vlines(x = app_per2, ymin = 0, ymax= 0.6, linewidth = 1.0, color = "k", linestyles
outFig = 'defuzzification_activity_mean_max.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



In [46]:
```
ap0 = np.zeros_like(x_appr)

fig, ax0 = plt.subplots(figsize=(8,3))
```

```
ax0.fill_between(x_appr, ap0, appr_nogood, facecolor = 'b', alpha= 0.7, label = "Not g
ax0.plot(x_appr, appr_nogood, 'b', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_okay, facecolor = 'g', alpha= 0.7, label = "Okay")
ax0.plot(x_appr, appr_okay, 'g', linestyle = '--')
ax0.fill_between(x_appr, ap0, appr_exc, facecolor = 'r', alpha= 0.7, label = "Excellen
ax0.plot(x_appr, appr_exc, 'r', linestyle = '--')
ax0.set_title('Centroid - Max of Maximum')
lab1 = f'{app_per3:.2f}'
ax0.annotate(lab1, xy=(app_per3, 0), xytext=((app_per3-5), -0.09))
ax0.legend()
plt.vlines(x = app_per3, ymin = 0, ymax= 0.6, linewidth = 1.0, color = "k", linestyles
outFig = 'defuzzification_activity_max_max.png'
plt.savefig(outFig, dpi = 300)
plt.show()
```



In [ ]: