Trabalho 1 Leonardo Monteiro Mastra Fontoura

In [26]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Leitura das informações

In [27]:

```
#usei os arquivos no mesma pasta onde salvei o meu ipy
path = ""
cie_file = path + "all_1nm_data.xls"
macbeth_file = path  + "ColorChecker_RGB_and_spectra.xls"
planilha = "spectral_data"
#print(cie_file,macbeth_file)
```

In [28]:

```
cie = pd.read_excel(cie_file,header=3)
cie
```

Out[28]:

	nm	CIE A	CIE D65	VM(l)	V'(l)	x bar	y bar	z bar	x bar.1	y bar.1
0	300	0.930483	0.03410	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	301	0.967643	0.36014	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	302	1.005970	0.68618	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	303	1.045490	1.01222	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	304	1.086230	1.33826	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
526	826	260.217000	59.16370	NaN	NaN	0.000002	5.980895e-07	0.0	0.000002	7.946400e-07
527	827	260.567000	59.45090	NaN	NaN	0.000002	5.575746e-07	0.0	0.000002	7.497800e-07
528	828	260.914000	59.73810	NaN	NaN	0.000001	5.198080e-07	0.0	0.000002	7.074400e-07
529	829	261.259000	60.02530	NaN	NaN	0.000001	4.846123e-07	0.0	0.000002	6.674800e-07
530	830	261.602000	60.31250	NaN	NaN	0.000001	4.518100e-07	0.0	0.000002	6.297000e-07

531 rows × 11 columns

In [29]:

```
#passando do dataframe para numpy array
nm = cie['nm'].to_numpy(dtype=np.int)
Aw = cie['CIE A'].to_numpy(dtype=np.float32)
D65 = cie['CIE D65'].to_numpy(dtype=np.float32)
V = cie['VM(l)'].to_numpy(dtype=np.float32)
xbar = cie["x bar"].to_numpy(dtype=np.float32)
ybar = cie["y bar"].to_numpy(dtype=np.float32)
zbar = cie["z bar"].to_numpy(dtype=np.float32)
```

In [30]:

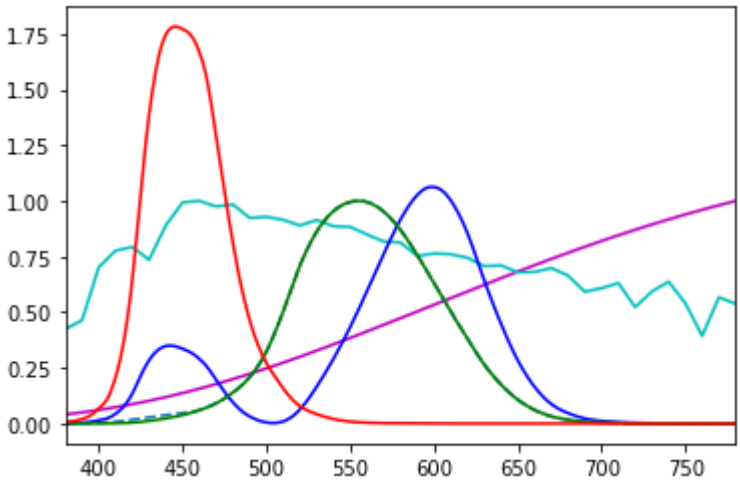
```
# selecionar a parte que interessa usando uma mascara
# ,ou seja, o comprimento de onda lambda de 380 a 780.
mascara = (nm>379)&(nm<781)

nm = nm[mascara]
Aw = Aw[mascara]
D65 = D65[mascara]
V  = V[mascara]
xbar = xbar[mascara]
ybar = ybar[mascara]
zbar= zbar[mascara]

# "Normalizando" a escala
Aw = Aw/np.amax(Aw)
D65 = D65/np.amax(D65)
```

In [31]:

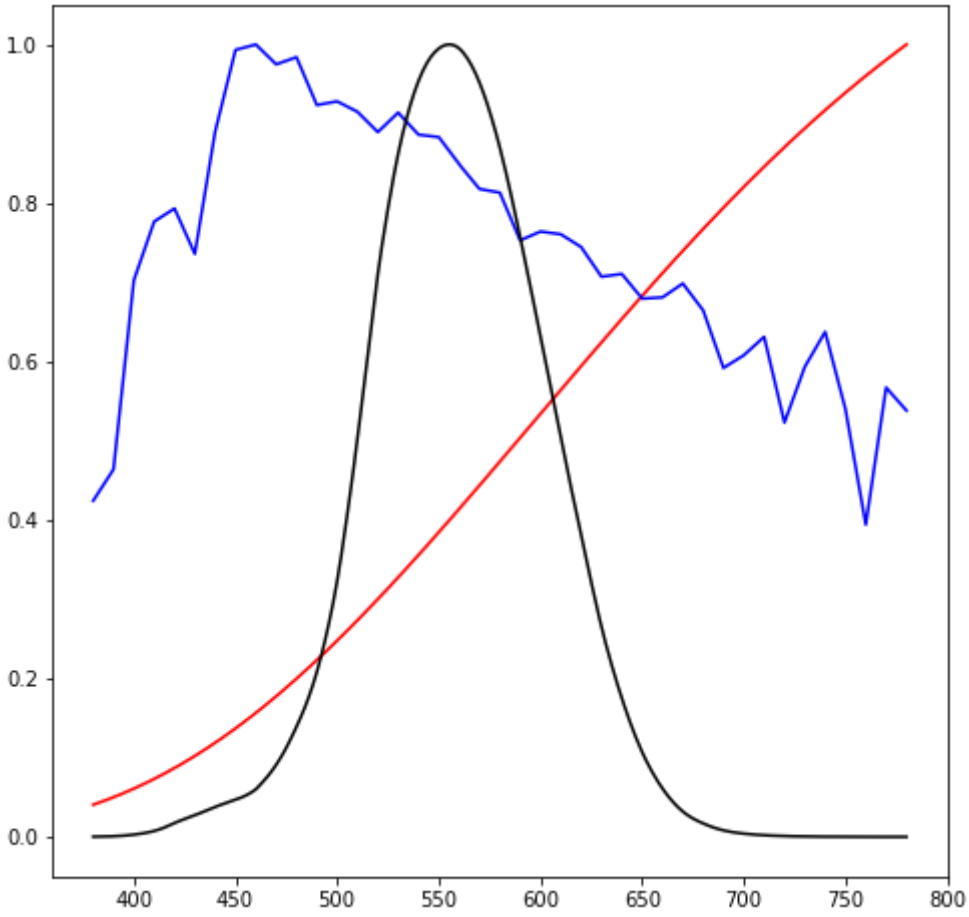
```
#plot das informações
plt.plot(nm,Aw,"m")
plt.plot(nm,D65,"c")
plt.plot(nm,V,"--")
plt.plot(nm,xbar,"blue")
plt.plot(nm,ybar,"green")
plt.plot(nm,zbar,"red")
plt.xlim(380, 780)
plt.show()
```



Eficiência Luminosa

In [32]:

```
# plot das luminosidades
fig = plt.figure(figsize=(8,8))
plt.plot(nm,Aw,"red")
plt.plot(nm,D65,"blue")
plt.plot(nm,V,"black")
plt.show()
```



Leitura do macbeth

In [33]:

```
# a variável "planilha" se refere a planilha "spectral data do arquivo excel"
checker = pd.read_excel(macbeth_file,planilha,skiprows=1,nrows=24)
checker.head()
```

Out[33]:

	No.	Color name	380	390	400	410	420	430	440	450
0	1	dark skin	0.054928	0.058196	0.060952	0.062206	0.062053	0.061716	0.061316	0.060896
1	2	light skin	0.121190	0.148141	0.180052	0.196914	0.201313	0.203969	0.208183	0.215898
2	3	blue sky	0.140803	0.184342	0.253856	0.306988	0.324564	0.331075	0.334410	0.333286
3	4	foliage	0.050885	0.053654	0.055276	0.056408	0.057415	0.058832	0.060468	0.061553
4	5	blue flower	0.158202	0.208656	0.300129	0.379623	0.412229	0.424513	0.429127	0.428518

5 rows × 38 columns

In [34]:

```
# Pegando a reflexão das 2 cores que interessam ao trabalho
reflectance = checker.loc[:,checker.columns[2:]].to_numpy(dtype=np.float32)
beta_green = reflectance[13,:]
beta_cian = reflectance[17,:]
```

Adpatando os dados da curvas do CIE para o color checker

In [35]:

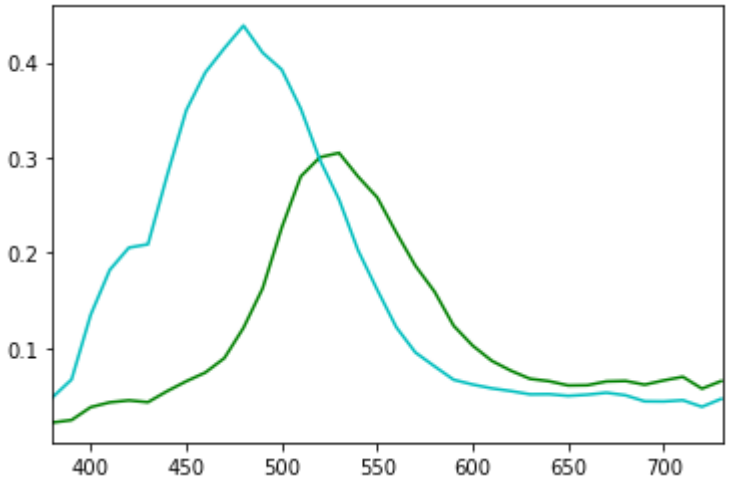
```
mask = (nm%10==0)&(nm<731)
```

In [36]:

```
# Aqui a variavel "mask" faz esse papel de pegar de 10 em 10
mask = (nm%10==0)&(nm<731)
nm_10 = nm[mask]
Aw_10 = Aw[mask]
D65_10 = D65[mask]
xbar_10 = xbar[mask]
ybar_10 = ybar[mask]
zbar_10 =  zbar[mask]
```

In [37]:

```
#plt.plot(nm_10,D65_10,'k--')
plt.plot(nm_10,D65_10* beta_green,'g')
plt.plot(nm_10,D65_10* beta_cian,'c')
plt.xlim(380, 731)
plt.show()
```



In [38]:

```
# Essa função faz o calculo em função de beta para o CIE
# Usando o iluminante D65 e a reflexão beta, e x,y,z de 10 em 10
def beta2XYZ(Lw, beta,x_,y_,z_):
    k = 1/np.sum(Lw * y_)
    X = k * np.sum(Lw* beta * x_)
    Y = k * np.sum(Lw* beta * y_)
    Z = k * np.sum(Lw* beta * z_)
    return np.array([X,Y,Z])
```

In [39]:

```
# import do scikit image para o cálculo
import skimage.color as sc
```

In [40]:

```
#Cálculo para o verde
XYZ_green = beta2XYZ(D65_10, beta_green,xbar[mask],ybar[mask],zbar[mask])
Lab_green = sc.xyz2lab(XYZ_green, illuminant='D65', observer='2')
sRGB_green = sc.xyz2rgb(XYZ_green)
```

In [41]:

```
#Cálculo para o ciano
XYZ_cian = beta2XYZ(D65_10, beta_cian,xbar[mask],ybar[mask],zbar[mask])
Lab_cian = sc.xyz2lab(XYZ_cian, illuminant='D65', observer='2')
sRGB_cian = sc.xyz2rgb(XYZ_cian)
```

In [42]:

```
def gamma_sRGB(x):
    t = x if x>0 else -x
    if t>0.0031308:
        gamma = 1.055*pow(t,1.0/2.4)-0.055
    else:
        gamma = 12.92*t
    return gamma if x>0 else -gamma
def XYZTosRGB(XYZ):
    M1 = np.array([[ 3.2404542, -1.5371385, -0.4985314],
                   [-0.9692660,  1.8760108,  0.0415560],
                   [ 0.0556434, -0.2040256,  1.0572252]])
    sRGB = np.dot(M1,XYZ)
    sRGB[0] = gamma_sRGB(sRGB[0])
    sRGB[1] = gamma_sRGB(sRGB[1])
    sRGB[2] = gamma_sRGB(sRGB[2])
    return sRGB
```

In [43]:

```
# Uma segunda maneira de calcular com uma questão a ser considerada nas coordenadas
sRGB_green_2 = XYZTosRGB(XYZ_green)
sRGB_cian_2 = XYZTosRGB(XYZ_cian)
```

Print de todas as informações com o debate sobre a coordenada negativa ¶

In [44]:

```
print("Verde")
print(f'Verde pelo skimage:\n Verde pelo Lab={Lab_green}\n sRGB={sRGB_green}\n Formula\n sR
print("\nCiano")
print(f'Ciano pelo skimage:\n Ciano pelo Lab={Lab_cian}\n sRGB={sRGB_cian}\n Formula\n sRGB
```

```
Verde
Verde pelo skimage:
 Verde pelo Lab=[ 55.42095702 -40.4905476   33.53996665]
 sRGB=[0.27909113 0.58478831 0.28054948]
 Formula
 sRGB2=[0.2790902 0.5847912 0.2805381]


Ciano
Ciano pelo skimage:
 Ciano pelo Lab=[ 51.6183847  -24.10756678 -25.70953849]
 sRGB=[0.         0.53416594 0.65170428]
 Formula
 sRGB2=[-0.1866621   0.53416846  0.65168038]
```

Como uma das variáveis é negativa, no sciimage ele não a representa pois os valores são apenas entre 0 e 1, já na fórmula temos a indicação que de aquela cor não pode ser representada no espaço sRGB. Ou seja, fica fora das cores representáveis pelas cores primárias no SRGB. O mesmo não acontece no Lab pois o lab é um espaço no qual não há restrição. Ele pode ser de qualquer tamanho.Todas as cores podem ser representadas.

Imagens do Verde e Ciano

In [45]:

```
def clip(x):
    if x<0:
        x=0
    elif x>255:
        x=1
    return x


def rgbString(sRGB):
    r= round(255*sRGB[0])
    g= round(255*sRGB[1])
    b= round(255*sRGB[2])
    r=clip(r)
    g=clip(g)
    b=clip(b)
    cor = f'rgb({r:.0f},{g:.0f},{b:.0f})'
    return cor
```

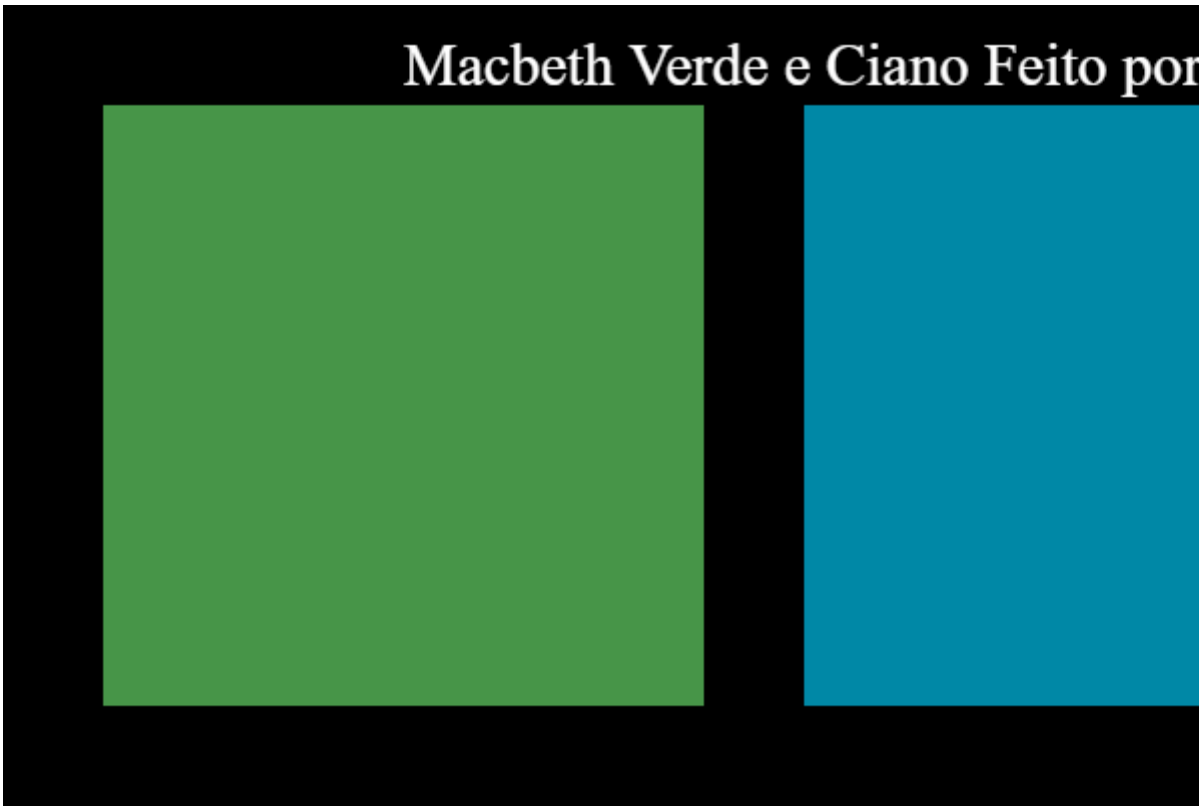
In [46]:

```
from ipycanvas import Canvas

border = 50
size = 300
canvas_width  = 3*border+2*size
canvas_height = 2*border+size

canvas = Canvas(width=canvas_width,height=canvas_height,sync_image_data=True)
canvas.fill_style='black'
canvas.fill_rect(0,0,canvas_width, canvas_height)
canvas.fill_style = rgbString(sRGB_green)
canvas.fill_rect(border,border,size,size)
canvas.fill_style = rgbString(sRGB_cian)
canvas.fill_rect(2*border+size,border,size,size)
canvas.fill_style='white'
canvas.font = '30px serif'
canvas.fill_text("Macbeth Verde e Ciano Feito por Leonardo Monteiro Mastra Fontoura",200,40

canvas
```



In [48]:

```
#salvando imagens
canvas.to_file("Leonardo_imagem_macbeth.png")
```

Gerando a imagem dos lambdas

In [49]:

```
beta_spec = np.zeros(D65.shape,dtype=np.float)
beta_spec[0] = 1

XYZ_spec = np.zeros(shape=(D65.shape[0],3),dtype=np.float)
sRGB_spec = np.zeros(shape=(D65.shape[0],3),dtype=np.float)
for i in range(D65.shape[0]):
    beta = np.roll(beta_spec,i)
    XYZ_spec[i,:]= beta2XYZ(D65, beta,xbar,ybar,zbar)
    soma = XYZ_spec[i,0]+XYZ_spec[i,1]+XYZ_spec[i,2]
    if(soma > 0):
        x = XYZ_spec[i,0]/soma
        y = XYZ_spec[i,1]/soma
        Y = 1
        X = x*(Y/y)
        Z = (1-x-y)*(Y/y)
        sRGB_spec[i,:] = sc.xyz2rgb(np.array([X,Y,Z]))
```

In [50]:

```
ones = np.ones(nm.shape,dtype=np.float)
plt.scatter(nm,ones,color=sRGB_spec)
imagem = plt.show()
```