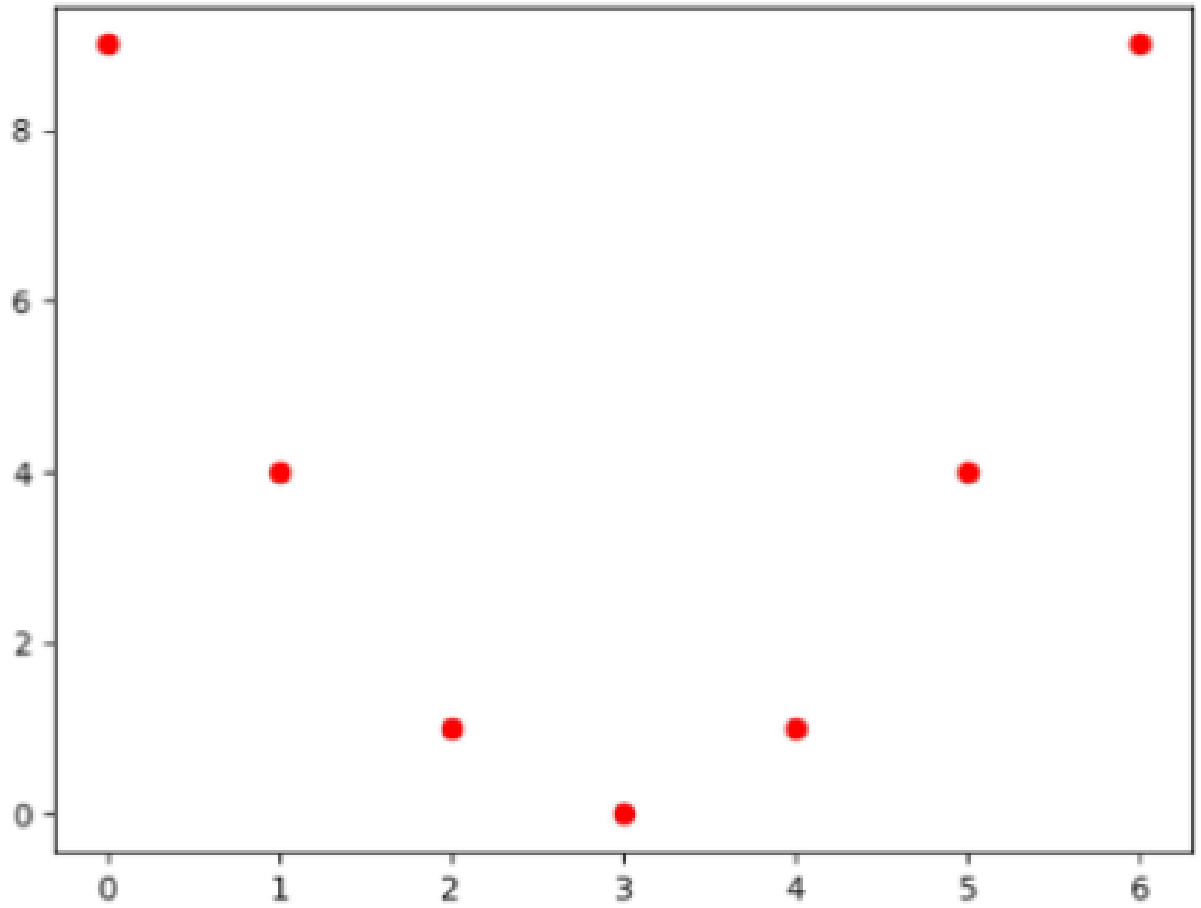
COMPUTAÇÃO GRÁFICA

Aula 2 –Fundamentos de Imagens

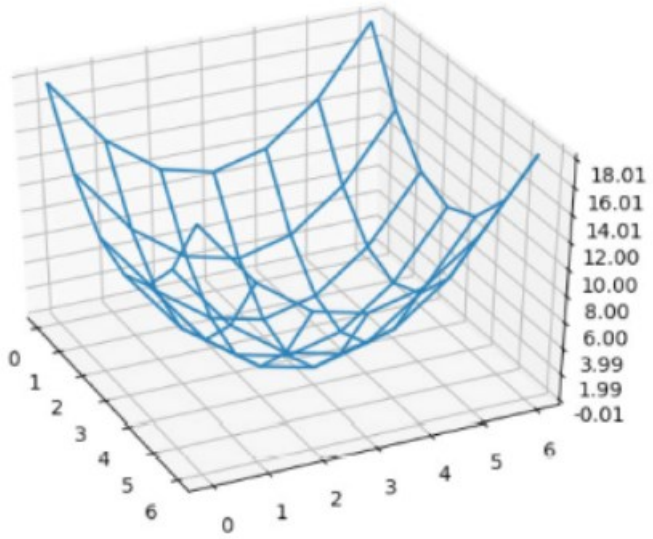
Curso de Ciência da Computação
Dr. Rodrigo Xavier de Almeida Leão
Cientista de Dados

Figura 1.8 | Funções discretas: a parábola $h(x)=(x-3)^2$ em (a) e o paraboloide $i(x,y)=(x-3)^2+(y-3)^2$ em (b), com os pontos conectados em um ***wireframe*** para melhor visualização

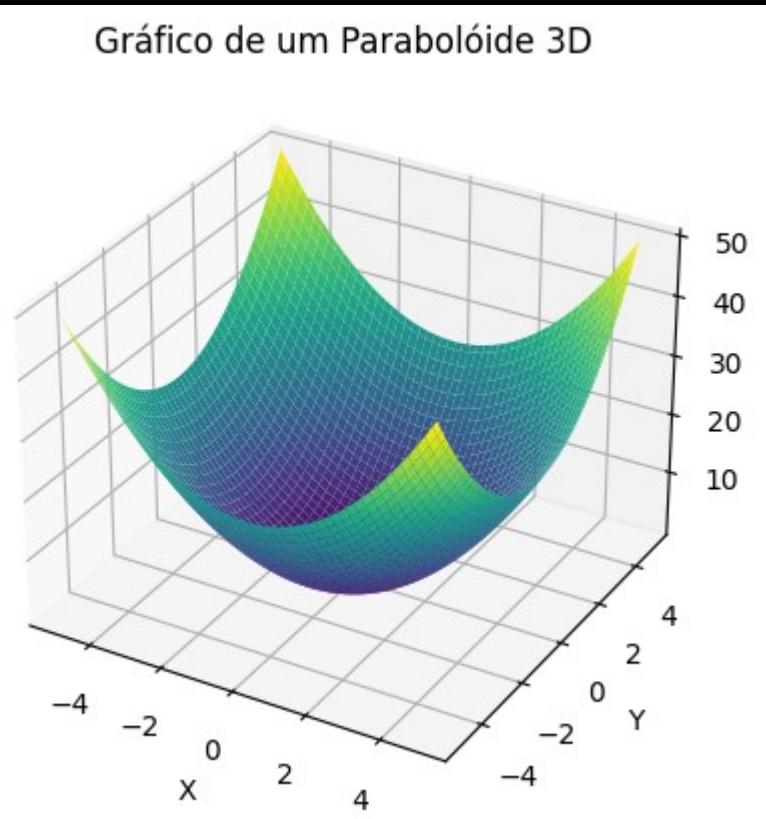
a)



```
[ ]  # prompt: desenhar grafico de uma função

     import numpy as np
     import matplotlib.pyplot as plt

     # Define a função que você quer plotar
     def f(x):
       return (x**2)+(4*x)-5  # Exemplo: função quadrática

     # Cria um array de valores de x
     x = np.linspace(-5, 5, 5)  # Valores de x de -5 a 5, com 100 pontos

     # Calcula os valores correspondentes de y
     y = f(x)

     # Plota o gráfico
     plt.plot(x, y)
     plt.xlabel("Eixo x")
     plt.ylabel("F(x)")
     plt.title("Título do Gráfico")
     plt.grid(True)
     plt.show()
```




```python
# prompt: desenhar o gráfico de um parabolóide 3d

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Criar os dados
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = X**2 + Y**2  # Equação do parabolóide

# Criar a figura e o eixo 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plotar o parabolóide
ax.plot_surface(X, Y, Z, cmap='viridis')

# Configurar os eixos
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
ax.set_title('Gráfico de um Parabolóide 3D')

# Mostrar o gráfico
plt.show()
```



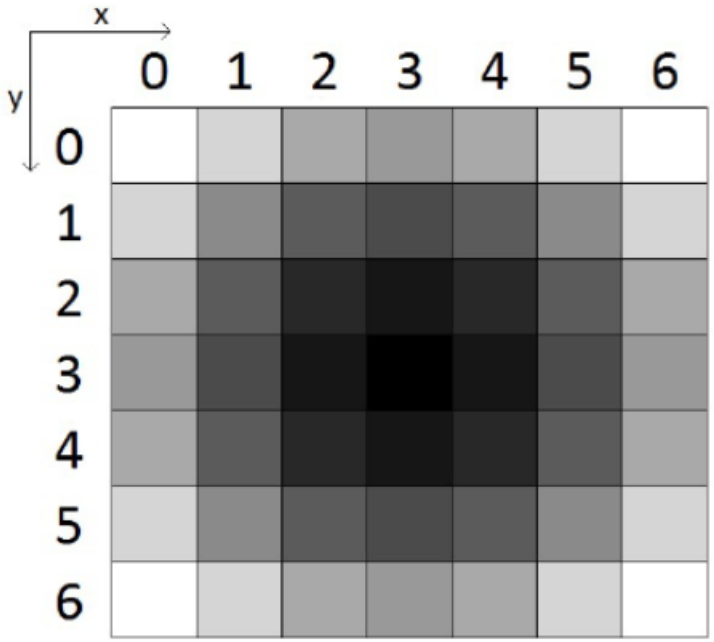
matriz bidimensional, na qual as colunas representam a coordenada x, e as linhas representam a coordenada y, como mostra a Figura 1.9 (a). Podemos também representar os números da matriz como tons de cinza, partindo do preto, para o valor 0, até o branco, para o valor 18, como na Figura 1.9 (b). A imagem representada por uma matriz é chamada de imagem **matricial**, **bitmap** ou, simplesmente, **imagem digital**. Cada elemento da imagem digital bidimensional é um pixel (acrônimo do inglês *picture element*) (HUGHES et al., 2013).

Figura 1.9 | Representações: matricial (a) e tons de cinza (b) do paraboloide $i(x,y)=(x-3)^2+(y-3)^2$

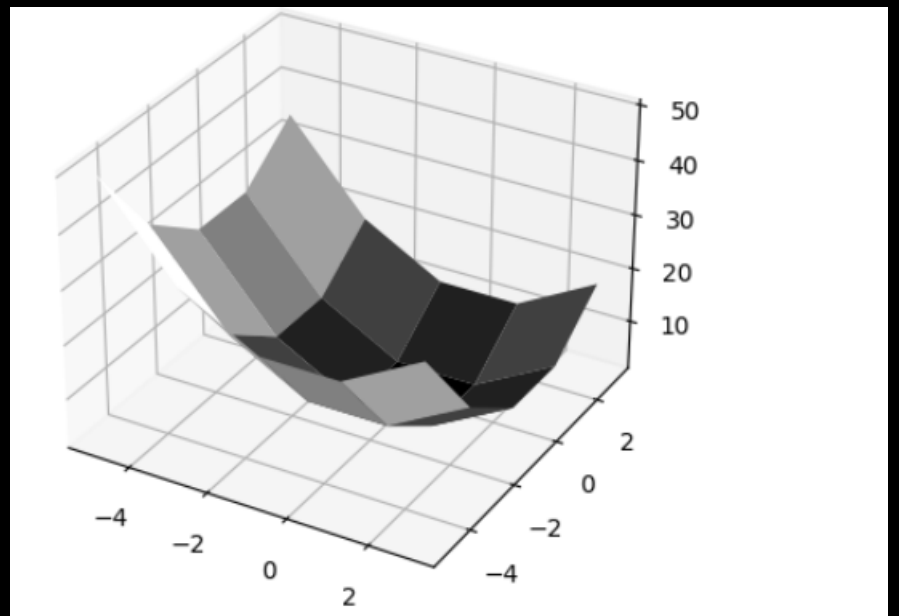
a)

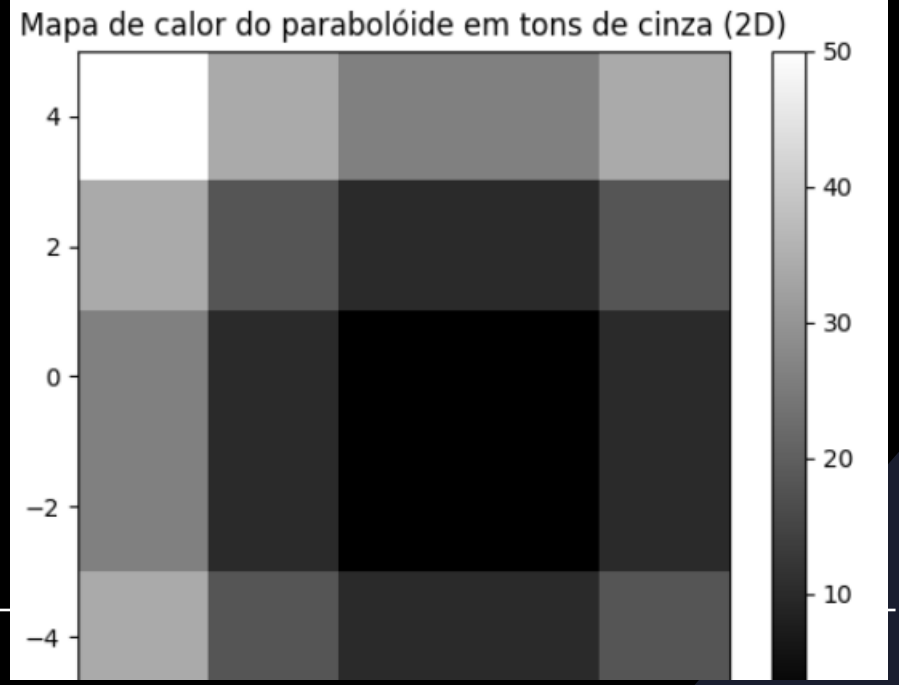
y \ x	0	1	2	3	4	5	6
0	18	13	10	9	10	13	18
1	13	8	5	4	5	8	13
2	10	5	2	1	2	5	10
3	9	4	1	0	1	4	9
4	10	5	2	1	2	5	10
5	13	8	5	4	5	8	13
6	18	13	10	9	10	13	18

b)



```
[ ]  # prompt: mostrar grafico 3d em tons de cinza e  o mapa de calor 2d
     import numpy as np
     import matplotlib.pyplot as plt
     from mpl_toolkits.mplot3d import Axes3D
     # Dados para o parabolóide
     x = np.arange(-5, 5, 2)
     y = np.arange(-5, 5, 2)
     x, y = np.meshgrid(x, y)
     z = x**2 + y**2

     # Cria a figura e o eixo 3D
     fig = plt.figure()
     ax = fig.add_subplot(111, projection='3d')

     # Plota o parabolóide em tons de cinza
     ax.plot_surface(x, y, z, cmap='gray')
     ax.set_title('Parabolóide em tons de cinza (3D)')

     # Mostra o gráfico 3D
     plt.show()

     # Cria a figura e o eixo 2D para o mapa de calor
     fig2 = plt.figure()
     ax2 = fig2.add_subplot(111)

     # Plota o mapa de calor em tons de cinza
     im = ax2.imshow(z, cmap='gray', extent=(-5, 5, -5, 5))
     ax2.set_title('Mapa de calor do parabolóide em tons de cinza (2D)')
     fig2.colorbar(im)

     # Mostra o mapa de calor 2D
     plt.show()
```





Xavier de Almeida Leão

```
[ ]  # prompt: gere um vetor raster numérico em forma de matriz para a imagem

     # Importa a biblioteca PIL para manipulação de imagens
     from PIL import Image

     # Salva a figura do mapa de calor como um arquivo temporário
     fig2.savefig('temp_heatmap.png')

     # Abre a imagem do mapa de calor
     image = Image.open('temp_heatmap.png').convert('L')  # Converte para escala de cinza

     # Converte a imagem em um array NumPy
     raster_array = np.array(image)
```

```
raster_array
```

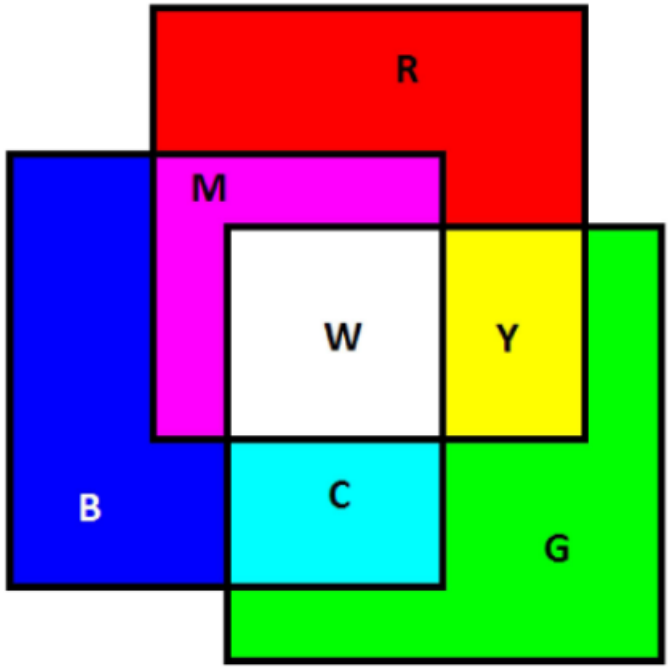
```
array([[255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
        255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 159,   0, 203,
         58,  93, 255, 255, 176,   2, 229,  31,  75, 255, 255, 255,  28,
        106, 148, 142,  54,   0, 180, 255, 255, 255,   0,  44,  20, 130,
        137,  31,   9, 211, 255, 255, 171,  47, 126, 154, 122,  17,  31,
        245, 255, 255, 255, 255, 255, 255, 255, 254,  62,   2, 106, 147,
         66,  46,   0, 159, 255, 255, 212,  15,  42, 137, 148,  62,   1,
        174, 255, 255, 255, 255, 255, 255, 255, 255,  96,   0,  73, 144,
        140,  79, 147, 255, 255,  53,  96, 143, 147,  70,   0, 148, 255,
        255, 255,  63,  63, 255, 255, 246,  41,   5, 113, 146,  61,   0,
        144, 255, 255, 223,   0,  42,  34, 136, 143, 225, 255, 255, 255,
        255, 255, 255,  92,   0,  90, 148,  82,  46,   0, 127, 255, 255,
        212,  11,  25, 131, 136,  33,   7, 200, 255, 255, 255, 255, 255,
        255, 255, 223,   0,  45,  32, 137, 129,  18,  21, 232, 255, 255,
        140,  61, 130, 154, 111,   9,  54, 254, 255, 255, 127,   0,  42,
```

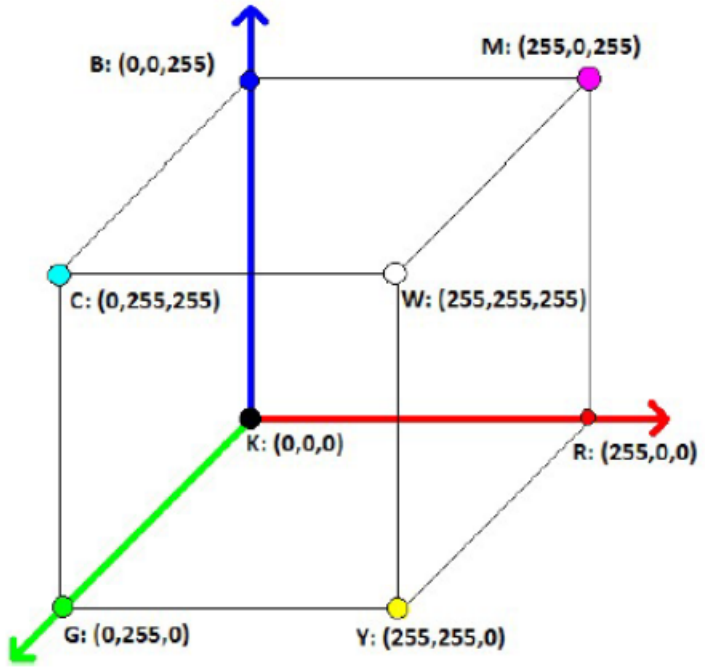
Na memória, uma imagem digital nada mais é do que um vetor unidimensional de bytes. Os elementos da matriz bidimensional são dispostos no vetor segundo uma ordem *raster*. A ordem *raster* tem como primeiro elemento o ponto de coordenada (0,0). Todos os elementos da linha 0 são adicionados ao vetor unidimensional, em seguida os elementos da linha 1, e assim por diante. O elemento da linha y, coluna x, se encontrará na posição $y \times xs + x$, onde *xs* é o tamanho de cada linha. Para a Figura 1.9 (a), os elementos no vetor estarão na sequência {18, 13, 10, 9, 10, 13, 18, 13, 8, 5, 4, 5, 8, 13, 10, 5, 2, 1, **2**, 5, 10, ...}. O elemento da linha 2, coluna 4, é o elemento na posição $2 \times 7 + 4 = 18$, destacado em negrito no vetor unidimensional.

Imagens digitais (do tipo bitmap ou matriciais) são armazenadas em arquivos que contêm todos os pixels. Os arquivos do tipo BMP são arquivos sem compactação. Se cada pixel ocupar 1 byte (imagem em tons de cinza), uma imagem de 12 megapixels (12MP) ocupará 12MB de um arquivo BMP, e se cada pixel ocupar 4 bytes (imagem colorida com transparência), o arquivo terá 48MB.

Para evitar arquivos muito grandes, as imagens digitais são comprimidas. Exemplos de imagens comprimidas são TIFF, JPEG e PNG. Para um

software utilizar uma imagem BMP, basta copiar sequencialmente os bytes do arquivo para a memória. Para utilizar uma imagem comprimida, é preciso ler todo o arquivo e fazer a descompressão para obter a imagem em bitmap na memória. Diferentes formatos de arquivos comprimidos podem usar diferentes algoritmos de compressão.



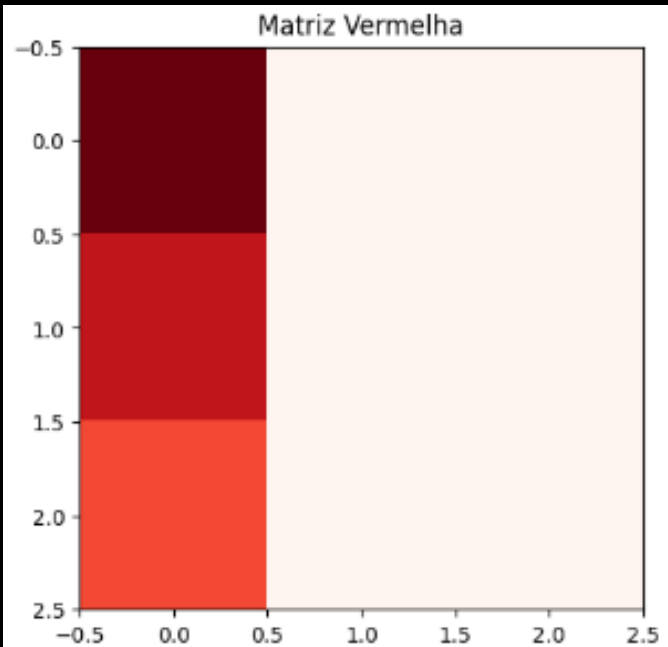


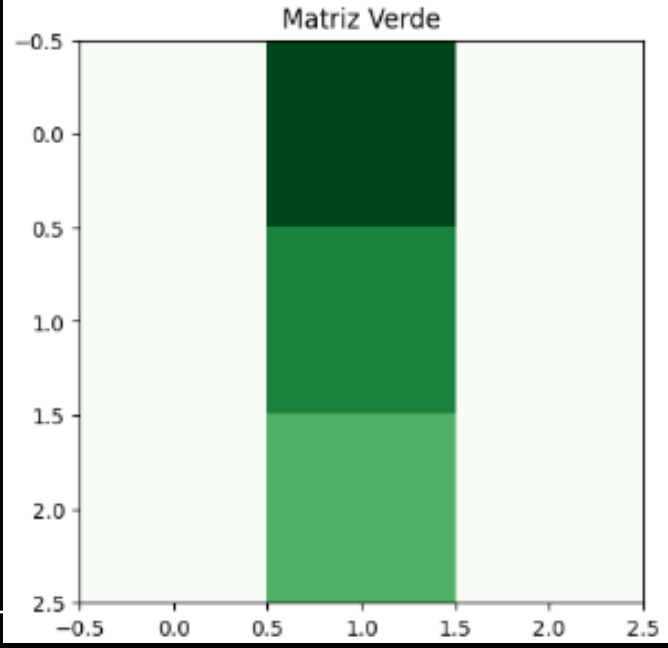
Fonte: elaborada pelo autor.

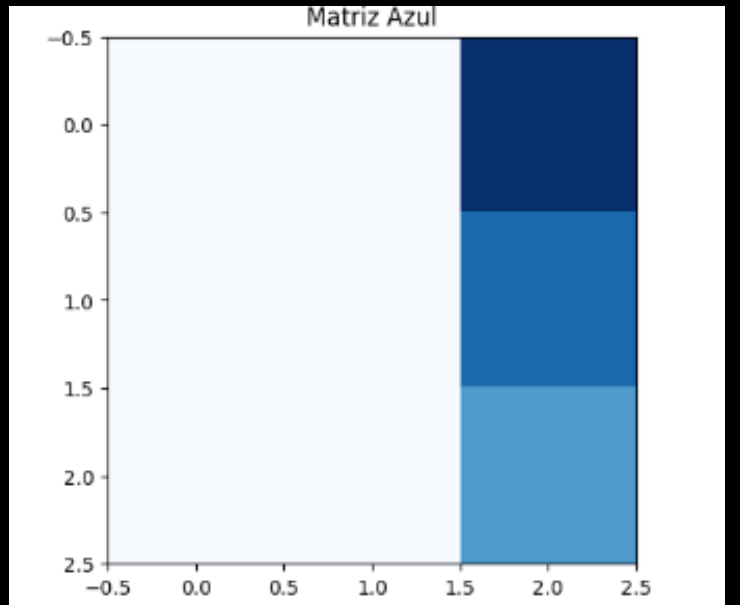
O modelo RGB surgiu a partir de estudos de composição de luzes, que demonstraram que a partir de apenas três cores primárias é possível, ajustando-se a intensidade de cada uma delas, obter todas as outras cores do espectro da luz visível. Os estudos também mostraram que as três cores primárias que

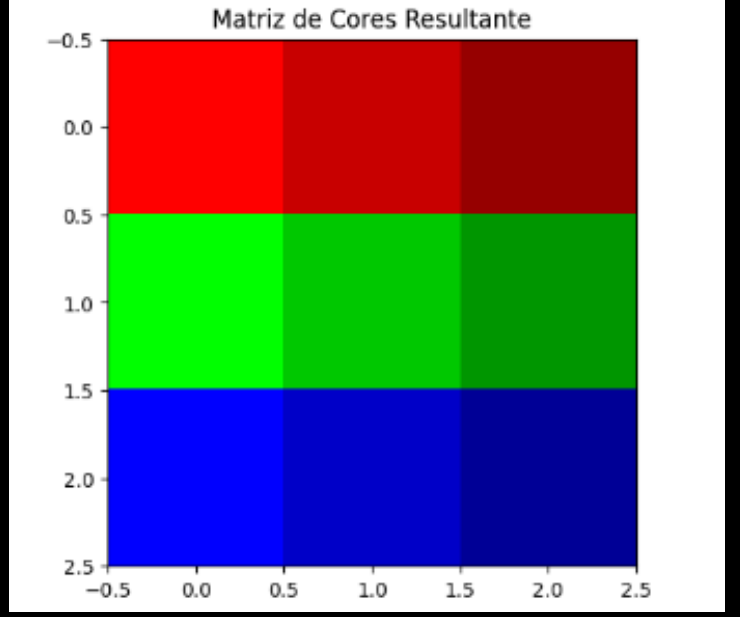
```
[ ]  # prompt: criar uma matriz de cor escolendo a aprtir de 3 matriz de cores RBG

     # Cria três matrizes de cores RGB (vermelho, verde e azul)
     red_matrix = np.array([[255, 0, 0], [200, 0, 0], [150, 0, 0]])
     green_matrix = np.array([[0, 255, 0], [0, 200, 0], [0, 150, 0]])
     blue_matrix = np.array([[0, 0, 255], [0, 0, 200], [0, 0, 150]])

     # Empilha as matrizes de cores ao longo de um novo eixo para criar uma matriz de cores 3x3x3
     color_matrix = np.stack((red_matrix, green_matrix, blue_matrix), axis=0)

     # Exibe a matriz de cores resultante
     print(color_matrix)
```

```
[[[255   0   0]
  [200   0   0]
  [150   0   0]]

 [[  0 255   0]
  [  0 200   0]
  [  0 150   0]]

 [[  0   0 255]
  [  0   0 200]
  [  0   0 150]]]
```

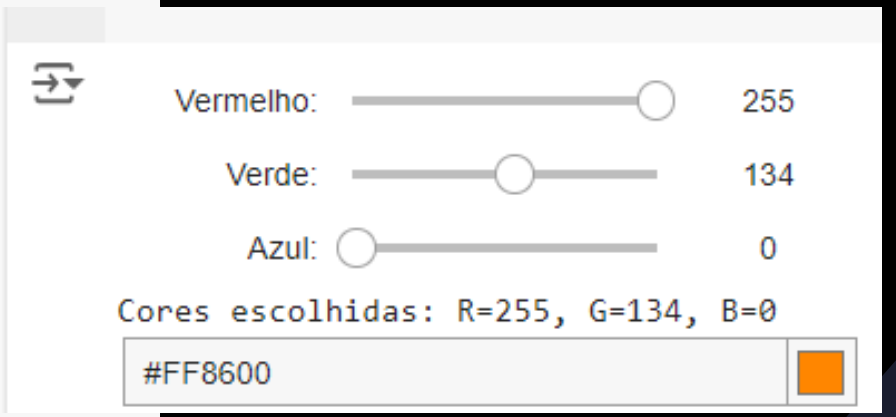

Matriz Vermelha
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Matriz Verde
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Matriz Azul
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Matriz de Cores Resultante
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

```python
#@title CRIADOR DE CORES
# prompt: mostrar a criação de uma cor a aprtir da escolha de RGB, mostrar as cores ecolh

import ipywidgets as widgets
from IPython.display import display

# Cria sliders para os valores RGB
red_slider = widgets.IntSlider(value=0, min=0, max=255, description='Vermelho:')
green_slider = widgets.IntSlider(value=0, min=0, max=255, description='Verde:')
blue_slider = widgets.IntSlider(value=0, min=0, max=255, description='Azul:')

# Cria um widget de saída para exibir a cor resultante
output_color = widgets.Output()

# Define a função para atualizar a cor resultante
def update_color(change):
  with output_color:
    output_color.clear_output()
    color = (red_slider.value, green_slider.value, blue_slider.value)
    print("Cores escolhidas: R={}, G={}, B={}".format(*color))
    display(widgets.ColorPicker(value='#' + ''.join('{:02X}'.format(c) for c in color)))

# Registra a função de atualização para os sliders
for slider in [red_slider, green_slider, blue_slider]:
  slider.observe(update_color, names='value')

# Exibe os widgets
display(red_slider, green_slider, blue_slider, output_color)
```

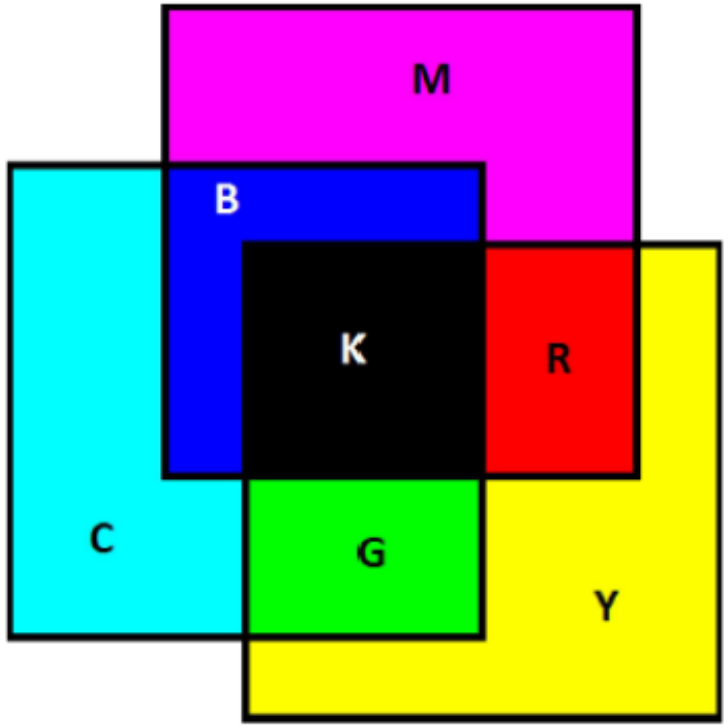


que é a composição de todas as cores. Uma cor é identificada por uma tupla *(r,g,b)*. Com $r = 0$, não há contribuição da cor vermelha, e com $r = 255$, há o máximo de contribuição da cor vermelha. O mesmo vale para as cores verde (g) e azul (b). Uma cor é, portanto, um ponto no interior do cubo de cores da Figura 1.11 (b), cujos vértices representam as cores primárias (R, G e B), as cores secundárias (C, M e Y), a ausência de cor (K) e a composição de todas as cores (W).

serve para TVs, monitores, projetores, etc. Quando as cores precisam ser impressas no papel, o papel é branco e, para ser lido, precisa estar iluminado. Para a impressão, portanto, é utilizado um modelo complementar ao RGB, o modelo CMY, que é um modelo de cor **subtrativo**. As cores básicas do CMY são as cores secundárias do RGB: o ciano (C), o magenta (M) e o amarelo (Y). As combinações dessas cores básicas são mostradas na Figura 1.12.

A combinação de todas as cores secundárias produz a ausência de cores, que é o preto (K). A combinação das cores secundárias, duas a duas, gera as cores primárias. O amarelo combinado com o ciano produz o verde, o ciano combinado com o magenta produz o azul, e o magenta combinado com o amarelo produz o vermelho. O modelo CMY é usado nas impressoras e, por esse motivo, os cartuchos de pigmentos são ciano, magenta ou amarelos. É comum, porém, que as impressoras possuam um quarto cartucho, com o pigmento preto, formando o modelo CMYK. Como a obtenção da cor preta pela mistura das cores secundárias requer grandes quantidades de pigmentos coloridos, a presença do pigmento preto gera economia, visto que a cor preta é muito utilizada nas impressões.

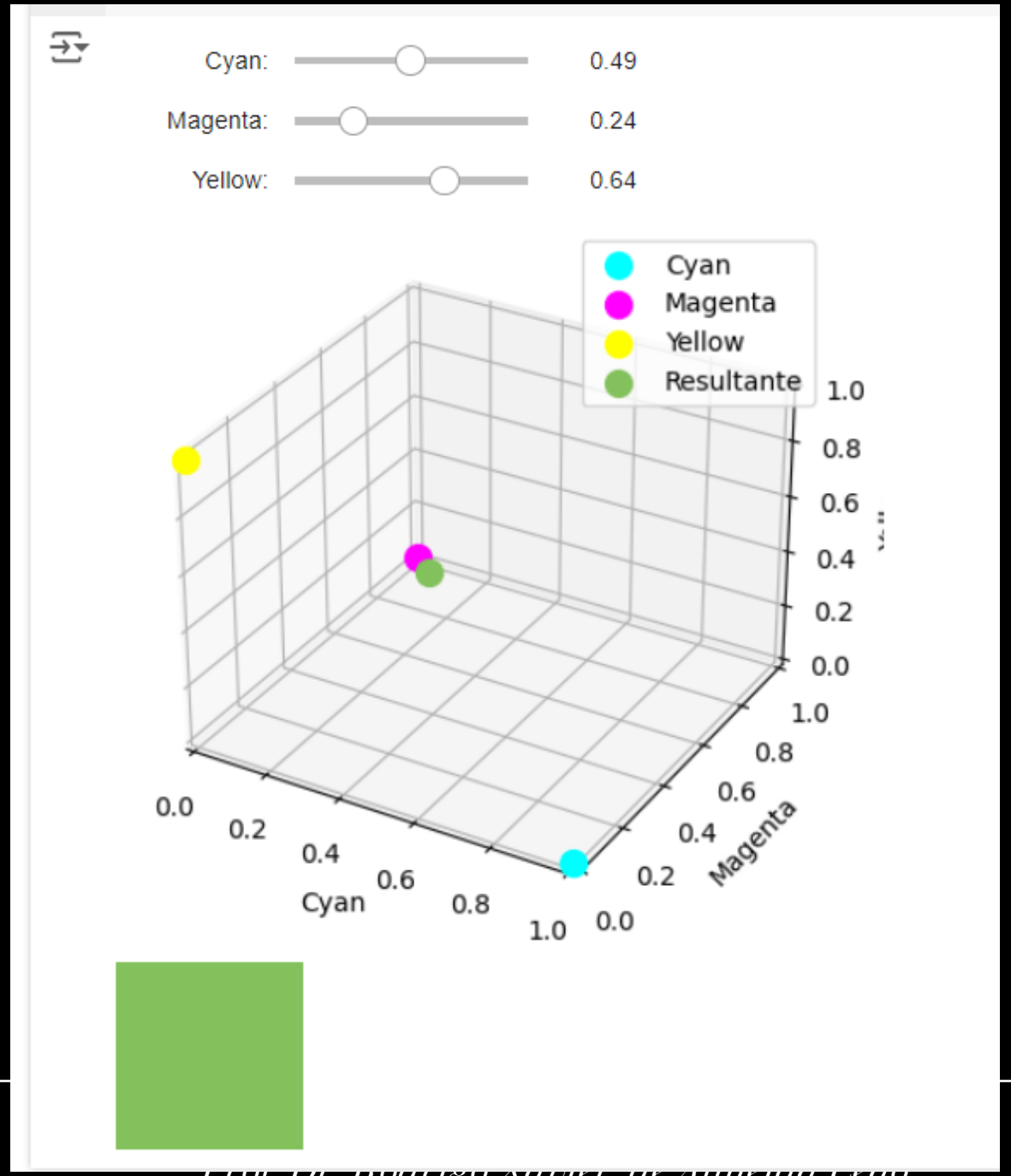
Figura 1.12 | Modelo CMY



Fonte: elaborada pelo autor.

CORES CMY

```python
#@title CORES CMY
# prompt: criar código similar ao anterior para o sistema

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image
import ipywidgets as widgets
from IPython.display import display

def cmy_to_rgb(c, m, y):
  r = 255 * (1 - c)
  g = 255 * (1 - m)
  b = 255 * (1 - y)
  return (int(r), int(g), int(b))

def plot_cmy_color(c, m, y):
  rgb = cmy_to_rgb(c, m, y)

  # Criando a figura e os eixos 3D
  fig = plt.figure()
  ax = fig.add_subplot(111, projection='3d')

  # Plotando os pontos das cores primárias
  ax.scatter(1, 0, 0, c='cyan', s=100, label='Cyan')
  ax.scatter(0, 1, 0, c='magenta', s=100, label='Magenta')
  ax.scatter(0, 0, 1, c='yellow', s=100, label='Yellow')

  # Plotando o ponto da cor resultante
  ax.scatter(c, m, y, c=[np.array(rgb)/255], s=100, label='Resultante')

  # Configurando os limites dos eixos
  ax.set_xlim([0, 1])
  ax.set_ylim([0, 1])
  ax.set_zlim([0, 1])

  # Configurando os rótulos dos eixos
  ax.set_xlabel('Cyan')
  ax.set_ylabel('Magenta')
  ax.set_zlabel('Yellow')

  # Adicionando a legenda
  ax.legend()

  # Exibindo o gráfico
  plt.show()

  # Criando a imagem da cor resultante
  img_array = np.zeros((100, 100, 3), dtype=np.uint8)
  img_array[:, :, :] = rgb
  img = Image.fromarray(img_array)
  display(img)

# Criando os sliders interativos
c_slider = widgets.FloatSlider(value=0, min=0, max=1, step=0.01, description='Cyan:')
m_slider = widgets.FloatSlider(value=0, min=0, max=1, step=0.01, description='Magenta:')
y_slider = widgets.FloatSlider(value=0, min=0, max=1, step=0.01, description='Yellow:')

# Criando a função de interação
widgets.interact(plot_cmy_color, c=c_slider, m=m_slider, y=y_slider);
```


Cyan: 0.49
Magenta: 0.24
Yellow: 0.64
Cyan
Magenta
Yellow
Resultante
0.0
0.2
0.4
0.6
0.8
1.0
Cyan
Magenta

Os modelos de cor RGB e CMY são modelos que usam combinação de cores. Um outro conjunto de modelos de cor representa as cores em três componentes, sendo um componente de luminância ou intensidade e dois componentes de cromaticidade (SOLOMON; BRECKON, 2013). O modelo HSL representa a cor pelos componentes de cromaticidade matiz (H, do inglês *hue*) e saturação (S), e pela componente de luminância (L). É um modelo bastante utilizado no processamento de imagens, isso porque alguns algoritmos de processamento de imagens requerem uma relação de ordem entre os elementos do contradomínio. Uma imagem digital em RGB é uma função $f : \mathbb{Z}^2 \rightarrow \mathbb{Z}^3$, na qual as três dimensões do contradomínio são o cubo de cores RGB. Não existe uma relação de ordem entre as cores do cubo RGB. O contradomínio da componente luminância (L) do HSL contém todos os contornos da imagem (como uma fotografia monocromática) e possui a relação de ordem (número inteiro). Assim, a imagem deve ser convertida do RGB para o HSL, o processamento em tons de cinza, aplicado sobre a componente L, e a imagem resultante, convertida de volta para RGB. Há outros modelos de cor que se assemelham ao HSL, com uma componente de luminância e duas de cromaticidade, como o HSI, HSV, Lab e YCbCr (GONZALEZ; WOODS, 2011).

```
]  # prompt: plotar mais de uma euqação no mesmo gráfico

   # Dados para a primeira equação
   x1 = np.linspace(-5, 5, 100)
   y1 = x1**2

   # Dados para a segunda equação
   x2 = np.linspace(-5, 5, 100)
   y2 = np.sin(x2)

   # Plotando as duas equações no mesmo gráfico
   plt.plot(x1, y1, label='x^2')
   plt.plot(x2, y2, label='sin(x)')

   # Adicionando legenda, título e rótulos dos eixos
   plt.legend()
   plt.title('Gráfico com Duas Equações')
   plt.xlabel('x')
   plt.ylabel('y')

   # Exibindo o gráfico
   plt.show()
```